Sigma Clermont
Université Clermont Auvergne
Clermont-Ferrand

MEMOIRE DE FIN D'ETUDE
MASTER ROBOTIQUE

— — ——

BEN CHARRADA Akila

Génération d'une séquence de postures pour la manipulation des objets.

Soutenu le 07 Septembre 2017
Institut Pascal

Remerciements

Je tiens tout d'abord à exprimer mes sincères et chaleureux remerciements à mon encadrant M. LENGAGNE Sébastien, pour sa constante disponibilité, son aide et son encouragement ainsi que pour m'avoir fait bénéficier amplement de sa rigueur scientifique, de ses critiques objectives et de ses conseils avisés, aussi pour ses qualités humaines qui suscitent mon admiration et mon profond respect.

J'adresse mes sincères remerciements aux membres de jury M. MEZOUAR Youcef et M. CHATEAU Thierry pour avoir accepté d'examiner et évaluer ce travail et pour leurs précieuses remarques qui vont améliorer ce mémoire.

Je tiens à témoigner ma profonde gratitude à tous les enseignants qui ont participé à l'évolution scientifique de ma formation à l'Université Clermont Auvergne.

Enfin, mes plus grands remerciements à ma famille et mes amis.

Résumé

Nous proposons une approche de planification unifiée pour les robots manipulateurs réalisant des tâches de manipulation nécessitant une compétence propre aux systèmes multidisciplinaire. Ces tâches sont basées sur des transitions de contacts ; entre les extrémités de l'organe effecteur (les doigts) et l'objet manipulé. Nous planifions ces transitions de contacts pour des systèmes constitués de plusieurs robots. Nous introduisons notre paradigme de planification de manipulation dans l'espace des configurations de systèmes pour lesquels nous résolvons le problème en comparant des méthodes de planification de chemin, de mouvement et des séquences. Nous présentons ensuite l'algorithme de planification de contacts basé sur une arbre d'exploration. Cet algorithme fait appel à un générateur de posture qui prend en compte des configurations de contacts générales dans l'espace pouvant être établis entre robots, objets, et environnement dans toutes les combinaisons possibles, le tout sous contraintes d'équilibre statique, collision et de respect des limitations mécaniques des robots.

Mots clés : Contrainte ; Optimisation ; Planification de Contacts ; Génération de Mouvement ; Manipulation.

Abstract

We propose a unified planning approach for robot manipulators performing manipulation tasks requiring competence specific to multidisciplinary systems. These tasks are based on contact transitions ; Between the ends of the effector (the fingers) and the manipulated object. We plan these contact transitions for systems made up of several robots. We introduce our planning paradigm in the space of system configurations for which we solve the problem by comparing path, motion and sequence planning methods. We then present the contact planning algorithm based on an exploration tree. This algorithm uses a posture generator which takes into account general spatial contact configurations that can be established between robots, objects, and environment in all possible combinations, all under static, collision and Compliance with mechanical limitations of robots.

Keywords. : Constraint ; Optimisation ; Planning Contacts ; Movement Generation ; Manipulation.

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Contexte

La robotique est un domaine de recherche qui se situe au carrefour de l'automatique, l'informatique, le mécanique et l'intelligence artificielle. Cette pluridisciplinarité est à l'origine d'une certaine complexité.

La conception des humanoïdes est un choix intuitif qui découle d'un désir de l'être humain pour recréer un avatar d'eux-mêmes. Ce désir inné suffit pour nous faire effectuer des recherches sur le sujet. Mais ce choix a également été justifié rationnellement par un certain nombre d'arguments cités ci-après, nécessaires à l'effort de recherche pour survivre dans la structure économique de la société.

Tout d'abord, un robot humanoïde nous permet d'étudier le mouvement humain en essayant de le reproduire. Une autre justification possible est que l'un des objectifs de la construction de robots est de faire remplacer les humains dans des tâches trop dangereuses ou fastidieuses pour eux. Dans ce cas, le design humanoïde est le plus approprié pour se déplacer et opérer dans un environnement qui a été initialement conçu et optimisé pour les opérateurs humains. Un dernier argument est que les robots humanoïdes évoluent et interagissent avec d'autres êtres humains réels, qui supposent ne pas avoir nécessairement le contexte scientifique pour comprendre les limites techniques ni la connaissance précise des capacités mécaniques et cognitives du robot.

Cela nous amène à ce sujet que nous essayons d'établir dans la présente dissertation. La planification multi-contact peut être l'un de ces concepts unifiants qui améliorent l'autonomie, contribuant à fournir aux robots un certain niveau d'autonomie pour accomplir des tâches de manière humaine dans un environnement opérationnel optimisé. Pour répondre à l'exigence énoncée ci-dessus, nous identifions une tâche principale : la **manipulation**, qui est réalisée à partir d'une configuration initiale et d'un objectif. La généralité est assurée en ne nous limitant pas aux séquences cycliques et en considérant l'utilisation de tous les points de contact et configurations possibles. L'atteinte de l'autonomie peut donc être décomposée pour planifier

ces séquences de contacts pour la manipulation de la manière la plus simple, donc la plus unifiée possible. C'est notre sujet traité.

Sans doute, les robots les plus utilisés dans l'industrie sont les robots manipulateurs. Ces bras mécaniques fixés (humanoïde à base fixe) au sol permettent une manipulation rapide et précise. Cependant, l'utilisation de plusieurs robots dans les usines implique une meilleure capacité de répartition spatiale, de meilleurs résultats d'exécution et d'un meilleur partage de tâche.

1.2 Plan du rapport

Ce projet expose la construction d'un planificateur d'appuis (cf. Figure 1.1).

Le chapitre 1, il introduit la définition de problème et les notations utilisées. Ensuite, il classe les différentes méthodes de planification comprenant la planification de mouvement, la planification de chemin et la planification des séquences. Puis, il décrit la locomotion et la manipulation des robots en expliquant les approches de mouvement précédant les appuis et les appuis précédant le mouvement. Enfin, il donne les applications des techniques citées.

Le chapitre 2, il présente la description et l'implémentation d'un générateur de posture qui permet de trouver une configuration du robot respectant une série de contraintes comme le contact, les collisions ou le maintien de l'équilibre. Ce générateur de posture est un outil de base pour la suite du travail. Il est aussi générique.

Chapitre 3 présente la description et l'implémentation d'un algorithme de planification d'appui pour plusieurs agents. Il se base sur un arbre de recherche qui lui permet de générer une séquence de posture en partant d'une posture initiale jusqu'à une posture désirée. En outre, il présente les résultats de visualisation obtenus.

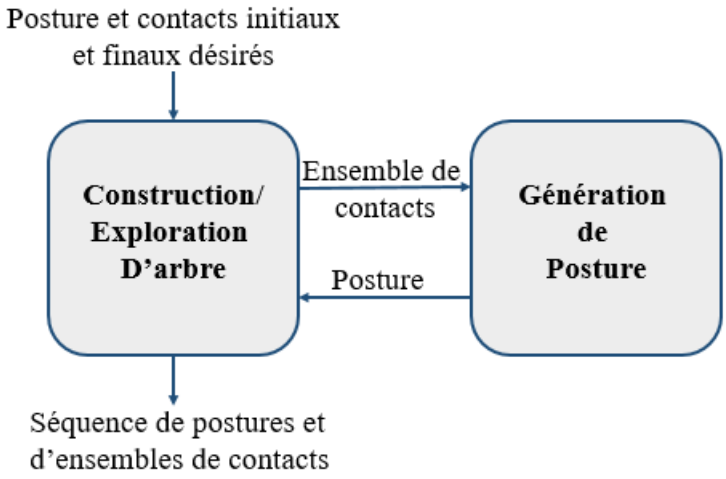


FIGURE 1.1 – Structure de planificateur.

1.3 Organisme d'accueil

L'Institut Pascal, est une unité mixte de recherche et de formation interdisciplinaire de 370 personnes placée sous la triple tutelle de l'Université Clermont Auvergne (UCA), du CNRS et de SIGMA Clermont. Le CHU de Clermont-Ferrand est également partenaire du laboratoire.

L'Institut Pascal est né de la fusion successive (2012, puis 2017) à vocation structurante de six laboratoires couvrant les disciplines des Sciences de l'Ingénierie et des Systèmes du site clermontois : Génie des Procédés, Mécanique, Robotique, Physique des Sciences de l'Information, Santé.

Le laboratoire développe des connaissances et des technologies contribuant à trois domaines d'application : l'usine (incluant les écosystèmes), les transports et l'hôpital du futur.

L'Institut Pascal est membre de FACTOLAB, laboratoire commun avec MICHELIN. Il est porteur du laboratoire d'excellence IMobS3 et membre du réseau CNRS EquipEx ROBOTEX et du LabEx GaNeX (PIA1). L'unité est membre des pôles de compétitivité Céréales Vallée et ViaMéca.

L'unité est structurée en cinq Axes de recherche comme présenté dans la Figure 1.2 :

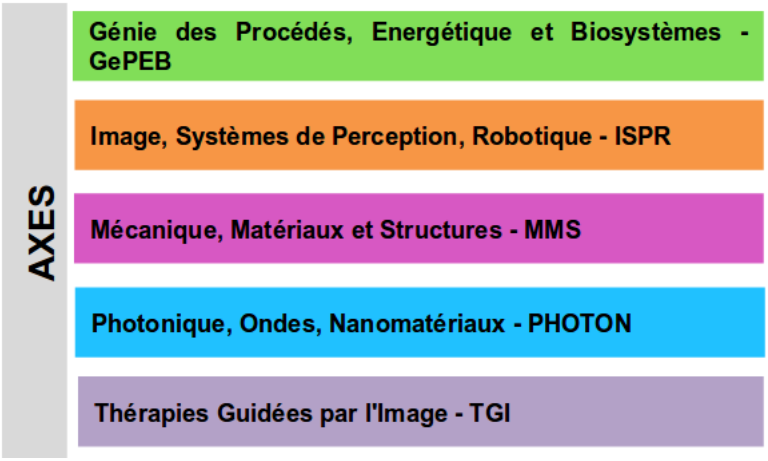


FIGURE 1.2 – Les axes de recherche de l'institut Pascal.

Ce sujet de stage est inclus dans l'axe ISPR qui a comme responsable M. Thierry CHATEAU et M. Youcef MEZOUAR. Cet axe est composé de 4 équipes qui sont :

- Vision artificielle (ComSee)
- Modélisation, Identification et Commande (MACCS)
- Systèmes de perception multi sensorielle (PerSyst)
- Architecture matérielle et logicielle pour la perception (DREAM)

Précisément, nous réalisons ce sujet dans l'équipe MACCS qui a comme responsable M. Youcef MEZOUAR et M. Lounis Adouane. Les recherches de cette équipe portent principalement sur la modélisation et le contrôle de robots mobiles et manipulateurs, la vision des robots, la vision active, la maîtrise visuelle et les comportements d'anticipation. Les applications de leurs recherches sont liées à la fabrication robotisée et aux systèmes de transport intelligents.

Chapitre 2

Cadre du projet et état de l'art

Sommaire

2.1 Introduction

Ce chapitre d'ouverture démontre la possibilité de résoudre des problèmes de planification de mouvements, particulièrement la manipulation et la génération de posture qui possèdent une abondante littérature.

Les robots sont traditionnellement catégorisés en manipulateurs à base fixe [9] et robots de navigation mobiles (roues [10] ou jambières [11]). Beaucoup d'entre eux, cependant, ne tombent pas strictement dans l'une de ces deux catégories car ils comportent à la fois des capacités de locomotion et de manipulation et sont conçus pour effectuer de manière indifférente les deux types de tâches, tombant ainsi dans une troisième catégorie de locomotion et de manipulation. Les robots humanoïdes, qui constituent la motivation initiale qui ont inspiré ce travail, sont des exemples typiques de tels systèmes intégrés de locomotion et de manipulation. Du point de vue de la planification et du contrôle des mouvements, la locomotion et la manipulation sont conceptuellement les mêmes problèmes [24]. Leur caractère commun est issu de leur sous-actionnement inhérent qui est résolu par les forces de contact : un système de locomotion est sous-actionné en ce sens que la position de la base mobile n'est pas contrôlée directement par des couples d'actionneurs, mais résulte plutôt des couples d'actionnement à travers des forces de contact avec l'environnement de soutien ; un

système de manipulation est également sous-actionné d'une manière strictement équivalente : les degrés de liberté de l'objet manipulé ne sont pas actionnés et sa position est un résultat indirect de l'actionnement du manipulateur à travers les forces de contact qu'il établit avec l'objet manipulé.

Le reste de ce chapitre est structuré comme suit : Section 2.2 présente la définition des principaux éléments de l'espace de travail, la section 2.3 explique l'approche de planification en classifiant ses différentes catégories en 3 sous-sections. La section 2.4 définie la théorie de la locomotion et manipulation en indiquant la relation entre le mouvement et les appuis. Enfin, la section 2.5 qui synthétise l'application des théories cités dans les sections précédentes en introduisant implicitement notre problématique.

2.2 Définition

Un robot R à n articulations évoluant dans un espace de travail W de dimension 3 peut être représenté par n+6 variables correspondant aux n valeurs des angles des articulations q (degrés de liberté internes), et aux 6 paramètres de position et orientation d'un de ses corps relativement à un repère W. Le robot est donc classiquement représenté par un point dans un espace de n+6 dimensions. Cet espace est appelé espace de configuration C. Pour une configuration donnée q, R(q) est la représentation du robot dans l'espace de travail.
Un obstacle O dans l'espace de travail W devient un volume O^C dans C défini comme

$$O^C = \{q \in C | R(q) \cap O \neq \oslash\} \tag{2.1}$$

soit l'ensemble des configurations pour lesquelles le robot et l'obstacle sont en collision dans l'espace de travail. Par la suite on peut construire $C_{obs} = \cup O_i^C$, l'ensemble des configurations où le robot est en collision avec un obstacle.
Soit r_i le $i^{ème}$ corps du robot, et $r_i(q)$ son image dans W à la configuration q, alors l'ensemble des auto-collisions du robot est défini par

$$C_{auto} = \{q | \exists i \neq j, r_i(q) \cap r_j(q) \neq \oslash\} \tag{2.2}$$

On définit alors le sous-ensemble de C libre de toute collision et auto-collision par

$$C_{free} = C \setminus \{C_{obs} \cup C_{auto}\} \tag{2.3}$$

Les algorithmes de planification classiques recherchent un chemin dans ce dernier ensemble.

Le robot est néanmoins sous-actionné, ses 6 degrés de liberté externes n'étant pas contrôlables par des moteurs dédiés. Il ne peut donc se déplacer que par le biais de ses interactions avec l'environnement : il doit s'appuyer sur des objets pour pouvoir évoluer. Dans le cadre de la formalisation de la planification, cela veut dire que **le robot ne peut rester dans C_{free} , mais doit faire des incursions dans ∂C_{obs} qui est la frontière de C_{obs}** . Hors de ∂C_{obs} , le robot n'est soumis qu'à la gravité, et se trouve dans une phase balistique. Tout au long de ce travail, on se restreindra au cas ou le robot reste en contact avec l'environnement, c'est-à-dire qu'il n'y a pas de phase balistique. Le robot évolue donc sur ∂C_{obs} . Plus précisément, il doit se trouver dans $C_c = \partial C_{obs} \backslash C_{auto}$, l'espace des configurations de contact sans auto-collision. Il représente l'ensemble des configurations pour lesquelles le robot a au moins un point en contact avec l'environnement, sans avoir aucune autre collision avec celui-ci et sans être en auto-collision .

2.3 Planification

Les robots sont conçus pour aider ou remplacer l'être humain dans des environnements irrégulier en réponse aux contraintes géométriques et physiques tel que l'évitement de collision et le maintien d'équilibre. Afin d'accomplir leurs tâches, ils doivent être capables de planifier leurs mouvements et leurs déplacements dans cet environnement. La figure 2.1 montre un exemple de déplacement d'une position initiale q_i à une position finale q_f dans un espace de configuration. La planification des déplacements est un axe de recherche où l'on peut trouver différents méthodes probabilistes au rang desquelles : la planification de mouvement, la planification de chemin et la planification des séquences [23].

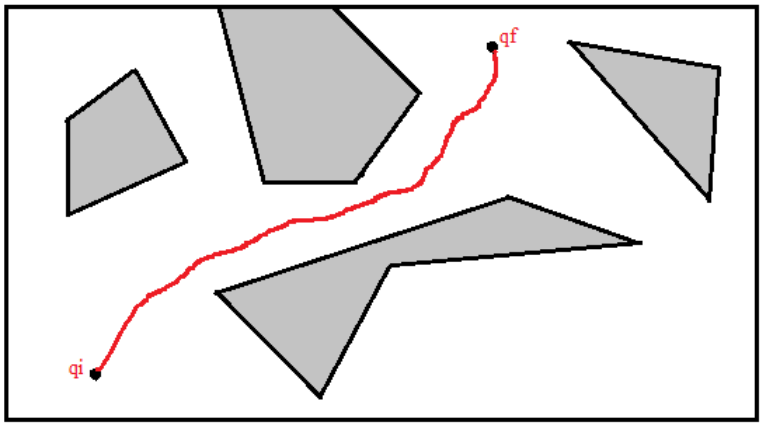


FIGURE 2.1 – Déplacement de q_i à q_f, les zones d'obstacle en gris et le chemin en rouge

La planification de chemin est défini par un processus qui permet à un robot de déplacer d'un point initial à un point final tout en évitant les obstacles. Le chemin trouvé définira le déplacement global du robot comme le déplacement d'un seul corps, il sera continu et défini par une posture dans un espace de 3 dimensions (deux positions et une orientation) pour un mouvement plan ou dans un espace de dimensions 6 (3 positions et 3 rotations) pour un mouvement dans l'espace 3D.

La planification de mouvement est défini par un processus qui détermine, d'une manière continue, les grandeurs articulaires qui permettent au robot d'effectuer un mouvement qui sera défini dans un espace à n dimensions (n est le nombre d'articulations actives du robot). Les deux méthodes de planification doivent prendre en comptes les mêmes contraintes, telles que la non-collision avec les obstacles mais la planification de mouvement doit prendre en compte aussi les contraintes qui traduisent les limites physiques du robot comme les couples maximum, les auto-collisions, l'équilibre ...

La planification de séquence est différente de deux autres catégories. Contrairement à ces derniers, la planification de séquences ne définit pas le déplacement de manière continue mais par des points de passage du robot qui devront être reliés par des mouvements.

Pour pouvoir naviguer dans son environnement, le robot a besoin de planifier le chemin sous la forme d'une séquence de postures entre sa position courante et la position finale désirée ainsi que les mouvements qui lui permettront de rallier ses postures.

2.3.1 Planification de chemin

Un chemin est l'évolution dans l'espace d'un point de robot en fonction du temps. Ce point est défini comme étant la configuration du robot à un instant t. Dans [13], le problème de planification de chemin est défini par :

- un environnement $W \subseteq \Re^3$
- des régions d'obstacle $O \subset W$
- un robot R défini dans W avec m degrés de libertés
- l'espace de configuration C qui est de dimensions 3 est défini comme l'ensemble des configurations possibles
- l'image de robot R dans une configuration q se note R(q)
- à partir de C on obtient :
  - $C_{free} = \{q \in C | R(q) \cap O = \emptyset\}$
  - $C_{obs} = C \backslash C_{free}$
- une configuration initiale et finale : $(q_i, q_f) \in C^2_{free}$
- un algorithme de planification de chemin doit calculer un chemin contenu $\tau : [0, 1] \longrightarrow C_{free}$ tel que $\tau(0) = q_i$ et $\tau(1) = q_f$

Les principales études sur la planification de chemin sont destinées aux robots à base mobile dont la plus grandes parties (Robot mobiles à roues) a la spécificité d'être des systèmes non holonomes, ce qui leur impose de se déplacer selon la perpendiculaire de leur axes de roues.

Généralement, pour relier une position initiale à une position finale, il peut exister un grand nombre de chemins possibles. Arechavaleta montre que l'être humain aurait tendance à minimiser la variation d'orientation de son torse durant son mouvement [2]. Mettler met en avant le fait que l'homme essaierait de diminuer la durée de déplacement lorsqu'il gère les déplacements d'un véhicule tel qu'un hélicoptère miniature [14].

Latombe [15] constitue un ouvrage de référence qui donne un état de l'art des travaux effectués jusqu'à 1990. LaValle [13] reprend le travail des années suivantes qui ont notamment vu l'avènement des méthodes probabilistes au rang desquelles la famille de PRM (Probabistic RoadMaps) [17] et celle des RRT (Rapidly exploring Random Trees) [16] qui sont les deux classes d'algorithmes les plus utilisées et adaptées.

Probabilistic Roadmap Method - PRM : Un algorithme PRM construit un graphe dans l'espace de configuration libre C_{free} dont les sommets sont des configurations tirées aléatoirement, et les arêtes sont des chemins de C_{free} . Ce graphe permet d'appréhender la structure de C_{free} en trouvant ses différentes composantes connexes. Un chemin entre la configuration initiale q_i et celle finale q_f du problème consiste en un chemin sans collision, de q_i à q_f . Les chemins entre deux points sont trouvés par un planificateur local simple (qui dans le cas le plus simple se limite à chercher si la ligne droite entre ces deux points est bien dans C_{free}) (cf. Figure 2.2).

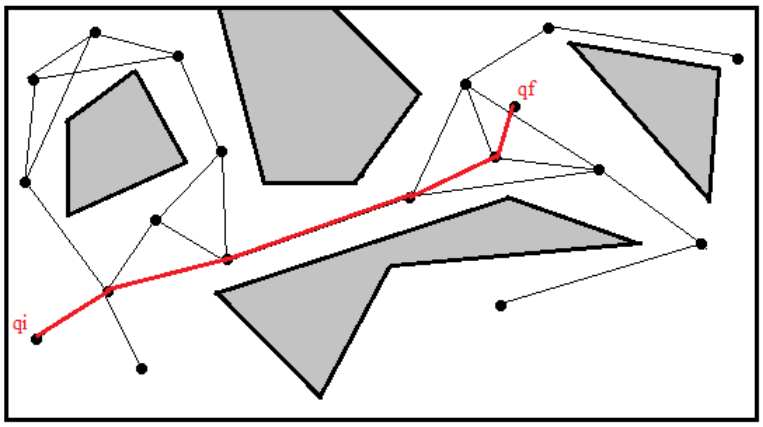


FIGURE 2.2 – Probabilistic Roadmap Method - PRM

Rapidly-Exploring Random Trees - RRT : Un algorithme RRT est une méthode rapide d'exploration d'arbre aléatoire qui a été presenteé en 1998 par LaValle [16] (cf. Figure 2.3). Cet algorithme permet de construire un arbre dont la racine est la configuration initiale, les nœuds des configurations et les arêtes des chemins de C_{free} , jusqu'à arriver à la configuration finale du problème. Pour ce faire, il tire à chaque itération une configuration aléatoire q et à partir du nœud de l'arbre le plus proche q_i construit une nouvelle branche en direction de q_f, c'est-à-dire un nœud q sur le segment $q_i q_f$ et une arête de q_i à q.

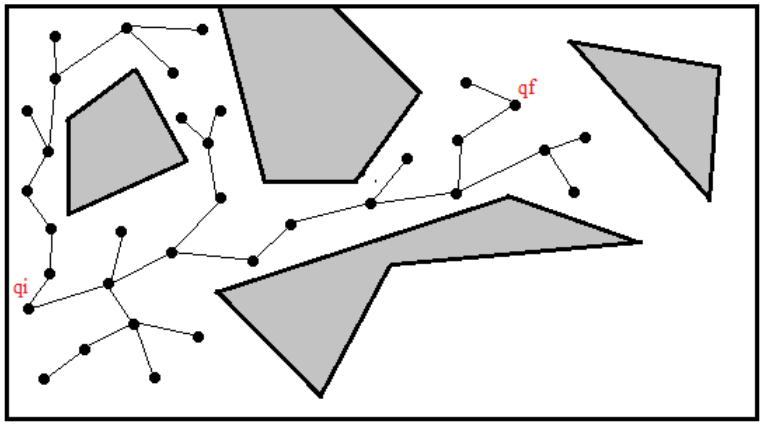


FIGURE 2.3 – Rapidly-Exploring Random Trees - RRT

2.3.2 Planification de mouvement

Les méthodes citées précédemment permettent de déterminer le chemin à suivre par le robot, néanmoins comment déterminer l'évolution des valeurs angulaires des articulations de telle façon à générer des mouvements qui vont permettre d'effectuer le déplacement ? La planification de chemin ne prend en compte que les contraintes géométrique. Cependant, un mouvement doit évidement respecter un certain nombre de contraintes dynamiques qui ne sont pas aisément prises en compte par les méthodes de planification de chemin.

Ces mouvements peuvent être générés par des processus d'optimisation hors ligne ce qui permet de considérer des modèles complexes où l'environnement est supposé connu et fixe à

une application. Dans ce cas, un algorithme d'optimisation permet de trouver l'évolution des trajectoires articulaires optimales qui va satisfaire un ensemble de contraintes et minimiser un critère.

Les méthodes basées sur un processus d'optimisation sont principalement utilisées pour des mouvements complexes, l'optimisation permet de gérer des systèmes ayant un grand nombre de degré de liberté et un grand nombre de contraintes (collision, équilibre, couples, ...)

En se basant sur un processus d'optimisation, Miossec a implémenté un algorithme d'optimisation de mouvement qui permet de calculer le modèle dynamique et sa dérivée par rapport aux paramètres d'optimisation de façon analytique, afin d'obtenir des mouvements en simple support [1] [18] et un mouvement en double support [19]. De la même manière, Suleiman a utilisé les crochets de Lie pour le calcul du modèle et de sa dérivé par rapport aux paramètres d'optimisation afin d'augmenter la stabilité de mouvements [20].

2.3.3 Planification de séquence

La planification de séquence diffère de la planification de chemin et de la planification de mouvements. Elle ne génère pas les déplacements de manière continues mais elle spécifie les postures (points de contact) pour atteindre la position finale. Elle est généralement utilisée pour des déplacements complexes. La séquence peut être composée par de postures spécifiées par le programmeur ou de postures prédéfinies dans une base de donnée [12].

Postures prédéfinies : à partir d'un ensemble de posture calculés hors ligne, il est possible de planifier le déplacement du robot entre sa position courante et sa position finale [21]. Pour trouver la séquence de posture qui permettra d'atteindre l'objectif, la plupart des algorithmes se basent sur la construction d'un arbre où chaque noeud représente une configuration (position et orientation) du robot et chaque branche correspond à un pas qui ramène le robot à une nouvelle configuration (cf. Figure 2.4).

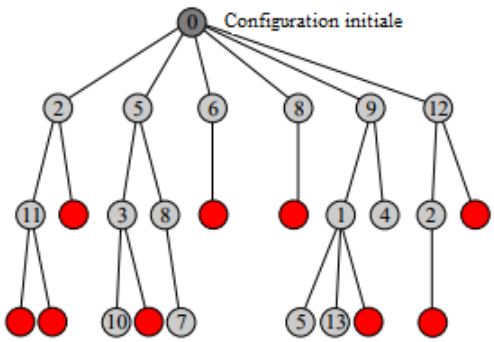


FIGURE 2.4 – Représentation d'un arbre [21]

Les configurations non faisables sont rejetées de l'arbre (en rouge sur la Figure 2.6) et la progression en aval de ces noeuds n'est pas étudié. L'algorithme doit donc déplier cet arbre jusqu'à ce que une branche mène à la posture finale.

Néanmoins, la navigation du robot à partir d'un ensemble de contact prédéfinis génère un arbre de très grande dimension et donc long à calculer ; Pour obtenir une solution un peu

1. Un mouvement en simple support est un mouvement pendant lequel une seule partie de robot est en contact permanent avec l'environnement

plus rapide, Chestnutt a pris en compte un critère heuristique afin de favoriser le chemins en ligne droite avec un minimum de pas [6] alors que Ayaz propose d'adopter une méthode comportementale [3] en limitant le nombre de choix face à un obstacle à trois : passer à gauche, passer à droite ou passer par dessus. Ce qui permet de ne considérer que certaines branches de l'arbre et donc de diminuer le temps de calcul.

Ces méthodes permettent donc de planifier rapidement un déplacement à partir d'un ensemble de mouvements de base dans un environnement peu encombré [7]. Pour un environnement très encombré, il faut disposer d'un grand nombre de mouvements, ce qui augmenterait la taille de l'arbre et donc le temps de calcul, dans ces conditions il faut pouvoir déterminer une séquence de posture adaptées à chaque étape de déplacement.

Postures non définies : Dans des environnements très encombrés où un ensemble de pas prédéfinis pourrait ne pas convenir, le robot ne doit plus considérer les obstacles comme des zones interdites, mais plutôt comme des zones de points d'appuis potentiels, par exemple, dans [22] le robot prend appui sur la table devant lui pour se lever de la chaise dans le but d'atteindre un objet placé sur la table (cf. Figure 2.5).

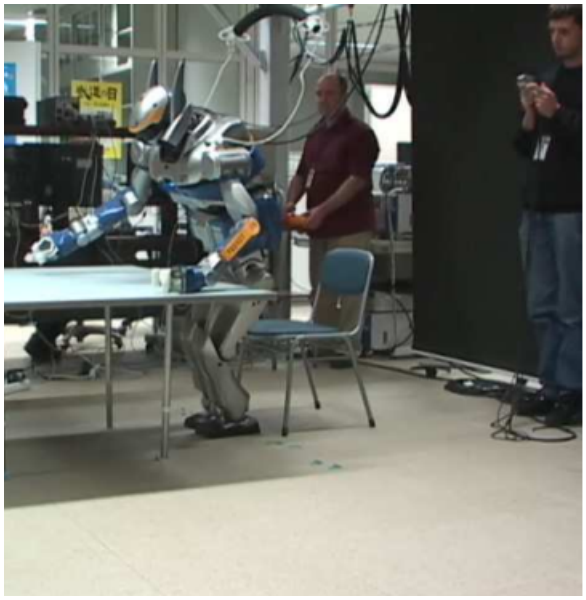


FIGURE 2.5 – Représentation de lever de chaise avec appuis sur la table [22].

Pour résoudre ce problème, Hauser propose une méthode de planification de mouvement où le robot prend appui sur son environnement avec certaines parties de son corps définies à l'avance [26]. Par contre, Escande a considéré des points d'appuis entre toute les parties du robot et son environnement pour générer une succession de posture [27]. Bouyarmanne a généré un mouvement statique entre deux de ces postures en contraignant la recherche de la solution optimale dans un espace guide qui est proche de l'espace de contact [5].

En récapitulant, dans cette section nous avons vu qu'il existe différents types de planification (chemin, mouvement, séquence) adaptés à différents type d'application. Toutes ces méthodes ont pour but d'amener le robot d'une configuration initiale à une configuration finale, en assurant la sécurité du robot et évidement son équilibre.

2.4 Locomotion et manipulation

La planification de support, qui est une partie de la locomotion/manipulation de robot multi-membres, donne un chemin solution consiste en une succession de chemins entre des instants de création ou de suppression de contact des corps terminaux des membres avec l'environnement. Le cas le plus étudié et le plus utilisé de ce cadre est la marche d'un humanoïde qui est une instance particulière du problème posé. Ce problème est pareil à la manipulation d'objet, les contacts y deviennent des préhensions entre lesquelles le robot suit un chemin. Nous pouvons trouver une succession d'appuis comme une manipulation de l'environnement en considérant celui-ci comme mobile, et la base du robot comme fixe.

Les deux types de problèmes partagent une même structure de leur espace de configuration, trouver un chemin met en jeu à la fois des déplacements et des choix de contact. Les algorithmes de planification résolvant ces problèmes se répartissent en deux catégories selon leur façon d'approcher le problème, soit planifier une trajectoire générale de robot en premier lieu puis trouver les appuis qui lui correspondent, soit commencer par le choix des appuis puis trouver le trajectoires entre les contacts successifs. Notre travail tombe dans le deuxième groupe.

2.4.1 Définition d'un pas

Un pas, au sens du problème de planification, est un déplacement sur une sous-variété. Trouver un pas revient à résoudre un problème de planification où le chemin est contraint à un sous-ensemble de l'espace de configuration (cf. Figure 2.6).

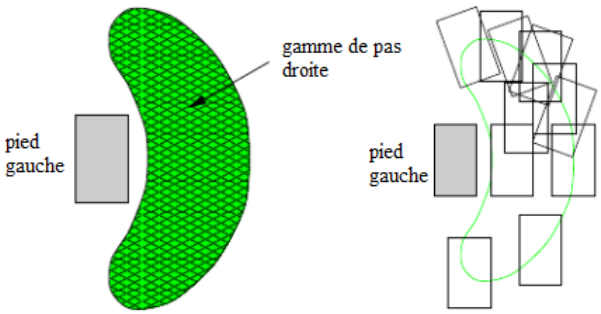


FIGURE 2.6 – Représentation des pas possible utilisées dans [21]

Par exemple un robot avec un seul corps en contact peut être considéré comme un robot dont ce corps est la base fixe. Les sous-variétés sont cependant souvent trop complexes pour cela il faut les regarder comme des variétés plongées dans C. Utiliser un PRM ou RRT classique dans ce cadre veut dire construire un graphe ou un arbre dont les nœuds et arêtes sont sur la sous-variété considérée. De ce fait, la probabilité de tirer une configuration dessus est nulle. Il faut donc adapter ces algorithmes pour la planification dans un sous-espace. Divers travaux s'intéressent à la question. Si par exemple les contraintes traduisent la présence de boucles, ce qui est le cas lorsqu'un robot humanoïde a ses deux pieds au sol (ou deux corps quelconques en contact avec l'environnement), [28] propose une méthode pour biaiser le tirage aléatoire, dénommée RLG (Random Loop Generator). L'idée est de décomposer une boucle en deux parties, la première pour laquelle on tire l'une après l'autre les valeurs articulaires de façon à rester dans une zone atteignable grossièrement définie de la seconde, dont il définit

ensuite la position par cinématique inverse. Les valeurs des articulations hors des boucles sont déterminées de manière complètement aléatoire. Si en revanche les contraintes sont quelconques, Stilman [30] a proposé une adaptation plus générale des RRT dont le principe est de projeter un point nouvellement construit q' soit directement sur la sous-variété, soit sur son espace linéarisé tangent en q0.

2.4.2 Mouvement précédant les appuis

Tant que le placement précis n'est pas nécessaire, il est possible de laisser ce choix de coté pour se concentrer sur le chemin global du robot. Une fois ce chemin est trouvé, alors les points d'appuis nécessaire se génèrent. En générale, ce type de planification à deux niveaux est très rapide puisque le calcul du chemin se fait en petite dimension, et le placement des contacts est donné par des heuristiques simples, ces dernières sont basées lorsque cela est possible sur des mouvement cycliques.

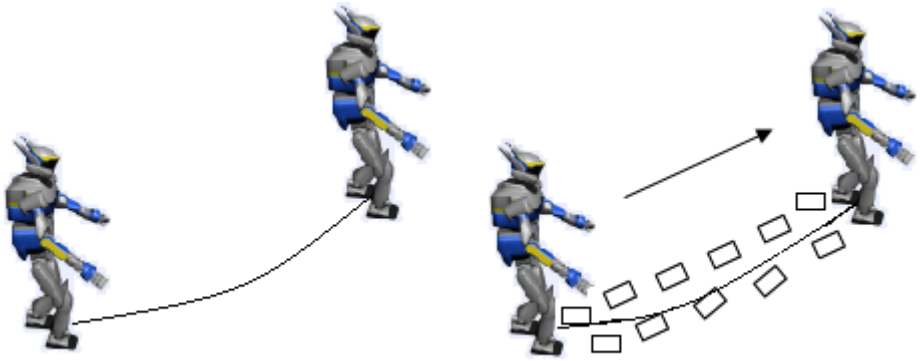


FIGURE 2.7 – Exemple de mouvement précédent les appuis.Trouver un chemin global (à gauche) pour lequel on génère une séquence d'appuis (à droite)

Ce type d'approche est souvent utilisée pour la marche (d'un humanoïde par exemple) sur un sol plat ou assez régulier. Le principe est de prendre un volume (un cylindre ou une boite) englobant le robot ou sa partie inférieure (cf. Figure 2.8) et de planifier un chemin horizontal par rapport au sol sans collision.

FIGURE 2.8 – Volume englobant la partie inférieure du robot [23].

En manipulation, cette approche consiste à planifier le chemin de l'objet manipulé seul puis d'en déduire une succession de prises et relâchement de l'objet par le robot manipulateur. Il faut néanmoins s'assurer lors de la génération du chemin qu'il existe toujours une possibilité de maintenir l'objet en tout point.

Cette méthode s'étend à des sols plus complexes [31] en 2.5D. Dans ce cas, le choix devient plus compliqué et il n'existe parfois pas de solution pour suivre la trajectoire. Si le sol devient plus complexe et peut tenir à la présence de peu d'endroit sur lequel prendre appui alors l'obtention de chemin devient plus difficile.

Enfin l'approche du mouvement avant le choix des contacts a été utilisé dans le cadre d'un humanoïde qui peut se déplacer en utilisant ses mains et ses pieds. Pour cela des positions de contact sont prédéfinies qui correspondent soit aux pieds, soit aux mains, soit aux deux, et le robot se déplace sous l'effet d'un champ de potentiel qui agit sur son centre de masse [15].

2.4.3 Appuis précédant le mouvement

Lorsque le choix de contact est important, par exemple lorsque l'environnement est trop complexe pour s'assurer de pouvoir suivre un chemin, il faut commencer par le choix des contacts dont on déduit ensuite un chemin.
Généralement, cette méthode est très lente par rapport à la précédente mais elle est plus générale autant que les simplifications qui étaient faites dans la première méthode (mouvements cycliques, boite englobante, etc.) sont autant de contraintes sur le mouvement qu'il n'est pas toujours possible de suivre. Les problèmes de planification

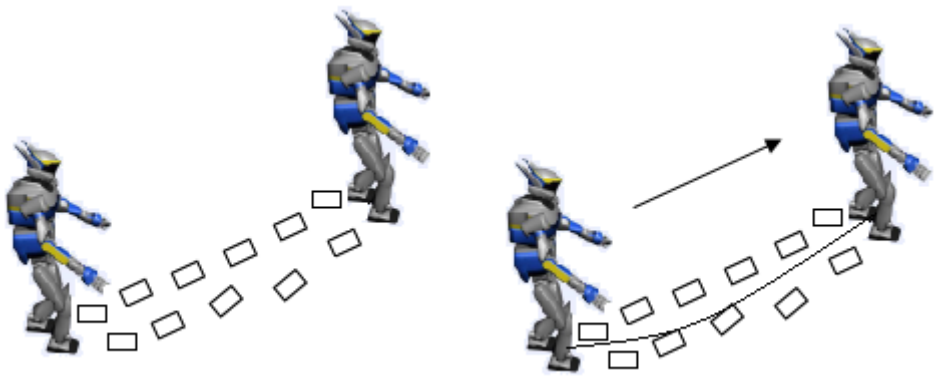


FIGURE 2.9 – Exemple d'appuis précédant le mouvement. Trouver les appuis (à gauche) pour lequel on génère un chemin global (à droite)

d'appuis/manipulation se divisent en deux groupes qui se distinguent par la structure de leur espace de configuration :

— L'espace est une collection de sous-variété. C'est le cas lorsque les emplacements de contacts prises sont discrets.

— L'espace est une collection de feuilletage [2]. c'est le cas lorsque le choix de la position des contacts est continu.

2. Un feuilletage est une partition en sous-variété connexe par un arc et disjointes d'une variété.

Contacts discrets

En raison de la structure de l'espace, il est nécessaire naturellement d'utiliser un graphe d'adjacence entre les sous-variétés [3] ce qui se réduit à trouver une configuration commune entre les deux.
Dans cette classe, il existe le travaux de Bretl [4] sur la planification pour un robot grimpeur. La planification se fait en deux étapes, commencer par la construction d'un graphe d'adjacence de manière progressive à partir de la variété contenant la configuration initiale jusqu'à atteindre la configuration finale. Dés que le graphe contient la variété finale, un chemin de ce graphe est suggéré. Il faut alors vérifier la possibilité de passer entre les configurations intermédiaires q_i de ce chemin. Cela se fait au travers d'un algorithme PRM appliqué sur les sous-variétés correspondant à chaque paire de q_i successifs. En cas d'insuccès la construction du graphe se rétablir de façon à proposer un autre chemin (cf. Figure 2.10).

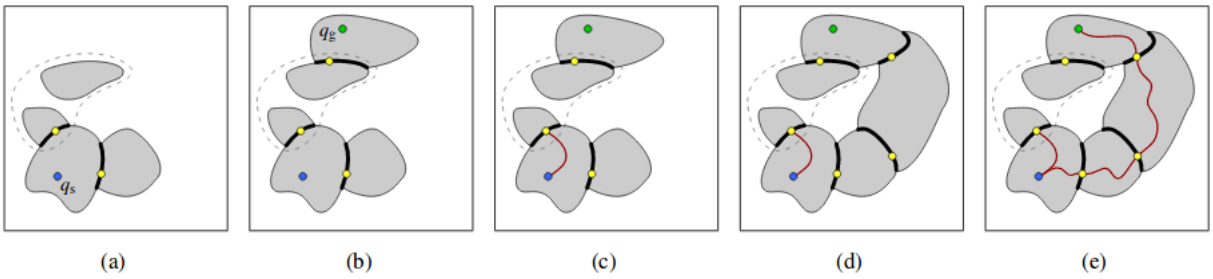


FIGURE 2.10 – Schéma d'une recherche multi-étapes simple en utilisant une stratégie en deux étapes. (a)-(b) Le graphique de la position est exploré jusqu'à ce qu'un chemin candidat multi-étapes soit généré du début au but. (c) Le graphique de transition est exploré pour vérifier que chaque étape candidate est faisable ; en cas d'échec, la recherche du graphe de position reprend. (d) Un nouveau chemin candidat à plusieurs étapes se trouve dans le diagramme de position. (e) Un chemin correspondant se trouve dans le graphique de transition, en vérifiant que le mouvement en plusieurs étapes est réalisable [4].

La force de cet algorithme consiste en la diminution de dimension obtenue grâce au graphe, qui permet une bonne recherche de l'espace en évitant les appels coûteux aux PRM à chaque fois qu'on y ajoute un nœud. Cela est réalisable car la probabilité de trouver un chemin entre deux q_i est très forte.

Plus tard, ce travail a été adapté pour les robots humanoïdes [25] en permettant à ces derniers de prendre contact grâce à des emplacements prédéfinis, et en améliorant la manière de tirer les configurations dans l'utilisation de la PRM. Les positions possibles des contacts restent connues à l'avance. L'utilisation de trajectoires prédéfinies qui sont adaptées aux contacts utilisés permet d'avoir des chemins plus naturels d'un point de vue visuel [26]. Ces trajectoires peuvent aussi aider à sélectionner les points de contact.

Contacts continus

La discrètisation des emplacements de contact pose quelques problèmes lorsqu'elle n'est pas dictée par la nature même du problème mais par un souci de simplification. La question qui se pose dés le début est comment discrétiser.

3. Deux variétés sont adjacentes si il existe un chemin pour passer de l'une à l'autre.

Quand le choix des contacts est continu, on est en présence de feuilletages, ce qui implique que dans un espace donné, le robot ne peut déplacer que dans certaines directions. Dans [1] Il existe un résultat qui dit que pour un feuilletage F qui est l'intersection de deux autres F_1 et F_2 , si il existe un chemin quelconque entre deux configurations de feuilles de F, alors il existe une succession finie de chemins sur des feuilles de F_1 et F_2 qui permet d'aller d'une configuration à l'autre. Cette propriété utilisée peut être très utile car elle permet de considérer une composante connexe d'un feuilletage comme un seul élément et d'accélérer la planification en fournissant la preuve de l'existence d'un chemin sans avoir à le chercher. C'est en quelque sorte une réduction de l'espace. Cette méthode sert dans [29] à la construction d'un graphe de manipulation.
Cependant, l'identification des composantes connexes des feuilletages (par l'utilisation de PRM) a un coût qui peut être grand quand celui-ci n'a pas une paramétrisation simple. Dans notre cas, nous aurons beaucoup de feuilletage à étudier, rendant cette méthode impraticable. Néanmoins, elle reste une méthode élégante et puissante lorsque le nombre d'espaces est petit. Cependant, la propriété de réduction utilisée dans [1] ne dispense pas de trouver par la suite la séquence dont on connaît l'existence.

Pour trouver une succession de pas il existe plusieurs méthodes dans le cadre de la locomotion ou la manipulation. [6] propose d'utiliser un algorithme A*, comme un ensemble de placement de pieds possible. Cette méthode est suffisamment rapide pour planifier en ligne dans un environnement variable. [8] utilise la méthode PRM dont les nœuds sont des placements de pieds et les arêtes des trajectoires obtenues par capture de mouvement et adapté pour commencer à un placement et finir à l'autre. Les placements de pieds sont retravaillés une fois qu'un chemin complet a été trouvé, pour lisser le résultat.

Tous ces travaux ne sont cependant pas facilement adaptables au cadre de ce projet car ils restent dans un nombre étroit de sous-espaces. Le nombre qu'il nous faut, explose avec le nombre d'objets sur lesquels prendre contact et le nombre d'emplacements de contact possibles sur le robot.

Le cadre de ce projet tombe dans une jonction entre [29] avec lequel nous partageons le type de structure d'espace (la réduction de l'espace composé d'un feuilletage connexe) et [4] pour le grand nombre d'espace considéré. La façon de résoudre ce problème est plus proche du second travail avec lequel nous partageons l'idée de la construction d'un arbre. Les grandes différences sont dans la continuité des positions de contact.

2.5 Application

Des nombreuses applications existantes se concentrent souvent sur la locomotion et la manipulation des robots humanoïdes. Les systèmes sous-actionnés tel que les humanoïdes ne peuvent se déplacer que par des contacts avec l'environnement. Le déplacement se fait par alternance entre les corps terminaux des membres avec le milieu extérieur, plusieurs approches sont avérés efficace pour planifier ce déplacement [34].

Plus récemment, les comportements de marche impressionnant sont générés automatiquement [32] même pour des terrains irréguliers. D'autres travaux traitent l'approche dédié à des situations où le passage sous la table ou le passage à travers un tunnel étroit sont demandés [33]. Il existe cependant la nécessité de planificateurs plus généraux. Il est possible de rassembler les différentes approches ci-dessus en contact avec les postures et les primitives de transitions de contact, lesquelles combinaisons seraient utilisées pour planifier des mouvements pour différentes situations. Cependant, il est difficile d'unifier ces approches car elles utilisent différentes stratégies de contrôle et nécessitent d'énormes efforts et des astuces pour

le réglage. Le robot doit être capable de planifier le mouvement sans restreindre les choix de contacts possibles qui peuvent être réalisés, c'est-à-dire envisager des contacts entre une partie de son corps et toutes les parties de l'environnement . C'est ce que nous abordons dans ce travail.

Travailler avec des contacts n'est pas simple : il y a des contacts pris sur les surfaces des objets qui sont également des obstacles sur lesquels nous ne voulons pas que le robot puisse entrer en collision (et même son propre corps) et d'autres contacts qui sont nécessaires pour manipuler des objets. Pour ces raisons, les algorithmes classiques de planification ne peuvent pas être utilisés facilement. Nous proposons un algorithme de planification qui réalise une manipulation multi-contacts générale. Il combine un premier algorithme de planification adapté pour couvrir un arbre de contacts possibles et un générateur de posture qui vérifie la faisabilité d'une configuration de contact. L'algorithme est entièrement re-visité pour planifier dans l'espace des contacts et le générateur de posture est formulé comme un problème d'optimisation des contraintes. Notre planificateur appartient à la classe des planificateurs de contact avant mouvement en ce sens que la trajectoire entière du corps entier est la conséquence d'une séquence de contacts (alors que dans les planificateurs de mouvement avant contact, les contacts sont choisis après la planification d'une trajectoire). Les contacts peuvent être pris n'importe où sur des surfaces prédéfinies et les postures et les choix de contacts sont obtenus en minimisant une fonction de coût (métrique) qui est la distance par rapport à la configuration (posture) finale. Cela fait respecter le naturel tout en conservant la généralité. Nous proposons un algorithme pour le cas général et donnons une description détaillée et complète de tous ses composants ainsi que détails de mise en œuvre.

2.6 Conclusion

Dans ce chapitre, nous avons démontré les différentes méthodes existantes pour la planification de chemins, de mouvements et de séquences. Les méthodes de planification de chemin sont généralement utilisées pour les robots mobiles à roues. La planification de mouvement s'appuie sur des processus d'optimisation. Cette méthode est utilisée pour des robots manipulateurs. la planification de séquences de mouvement s'appuie sur la construction d'une arbre de recherche.

Nous avons mis en évidence les difficultés de problème de planification auquel se traite par ce travail. Ces difficultés se trouvent dans le choix des contacts faisables en satisfaisant des contraintes et minimisant un ou plusieurs critères qui se posent tout au long de la planification. Nous avons fait l'état de l'art en locomotion et en manipulation et conçu l'approche dans laquelle tombe notre travail : la planification "appuis précédant le mouvement".

A présent que notre problème est posé, nous allons recoder l'état de l'art en introduisant un outil de base du planificateur proposé dans ce projet : ***le* générateur de posture**.

Chapitre 3

Génération d'une posture

Sommaire

3.1 Introduction

La posture est un ensemble de configurations corporelles, y compris les positions de contact. Le choix des postures peut avoir une incidence sur la possibilité d'accomplir une tâche avec succès. Les postures optimales varient habituellement d'une tâche à l'autre. Le problème de génération de posture est formulé comme un problème d'optimisation non linéaire sous contraintes.
Le calcul des configurations du robot pour répondre aux exigences d'un ensemble de tâches données, dans un état viable, est un problème récurrent dont la complexité augmente avec celle du robot. Les travaux précédents cherchent une configuration pour laquelle le robot remplit les tâches sous contraintes de limite d'articulation, d'auto-collision et d'évitement de collision non souhaité, l'équilibre, la limite de couple, etc. Un tel problème se rencontre tant dans la planification que dans le contrôle. Dans les deux cas, le temps de calcul et la robustesse sont des problèmes critiques.

Nous détaillons dans ce chapitre une méthode pour générer sur un sous espace une configuration qui respecte de telle contraintes, par le biais de l'écriture et la résolution d'une

optimisation sous contrainte. Suivant cette méthodologie, le reste du chapitre est structuré comme suit : nous proposons au début une définition de problème en exprimant les entrées et la sortie (section 3.2). Ensuite, nous expliquons le critère à optimiser par notre travail (section 3.3). Puis, nous donnons les contraintes à satisfaire (section 3.4). Après, nous donnons la formulation qui exprime les contraintes d'égalité et d'inégalité (section 3.5). Enfin, nous écrivons l'algorithme d'optimisation et l'architecture globale du générateur de posture (section 3.6).

3.2 Définition du problème

Dans cette section, nous allons présenter le formalisme théorique pour générer une posture d'un système robotisé (plusieurs robots) poly-articulé pour lequel les contacts peuvent se produire entre une partie quelconque du corps et toute partie de l'environnement. À partir d'une description d'un ensemble d'entrée, le générateur de posture calcule une posture composée de plusieurs contacts permettant à notre système d'atteindre son objectif.

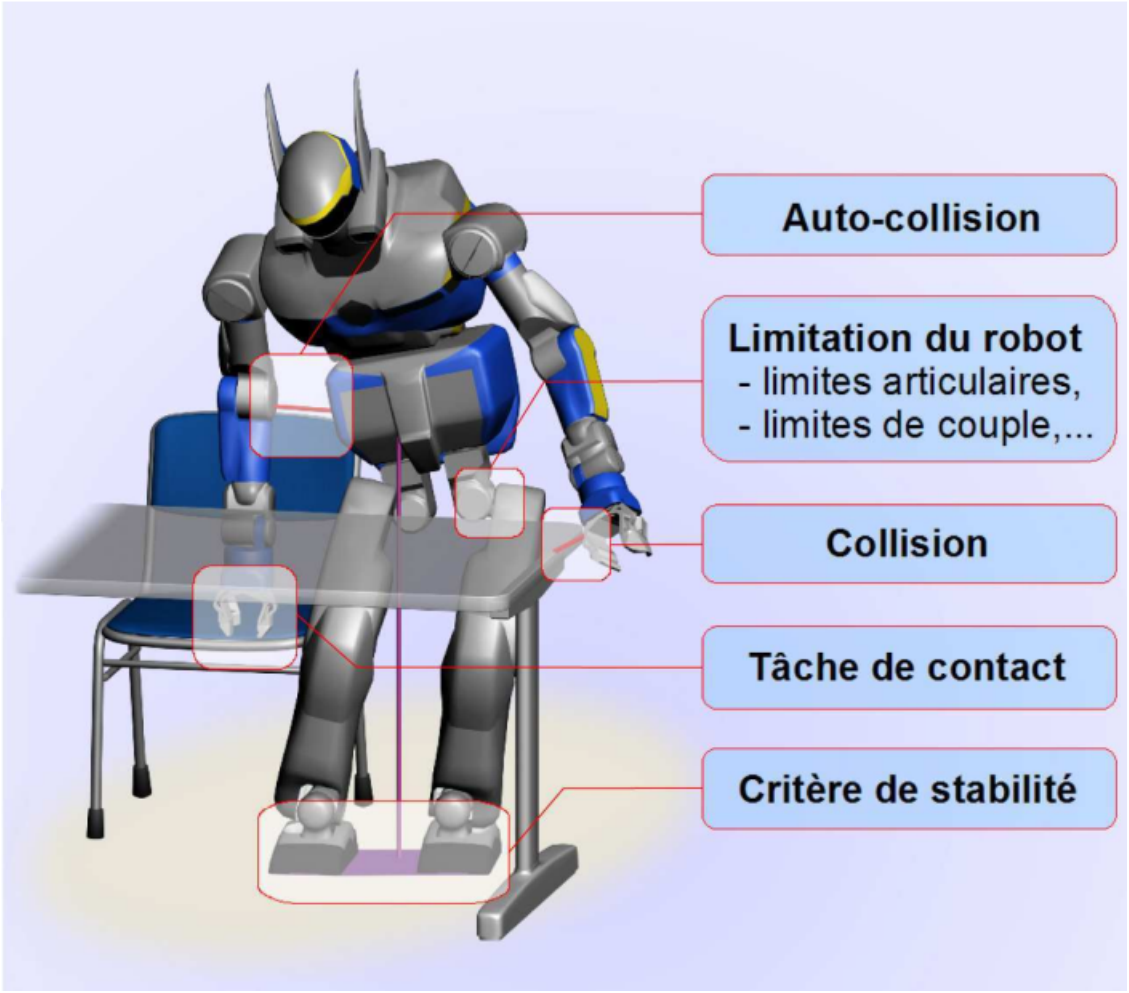


FIGURE 3.1 – Les différentes contraintes à respecter par le robot dans le cadre de la génération de posture pour la projection sur un sous-espace de contact [22].

Principalement, nous donnons au générateur comme **entrées** ; le système robotisé en définissant leurs articulations et leurs corps, l'ensemble des contacts qu'ils peuvent créer

avec l'environnement, les critères qu'ils doivent optimiser et l'ensemble des contraintes qu'ils doivent satisfaire.
Chaque robot doit satisfaire : les contraintes de collision c'est à dire garder une marge de sécurité strictement positive entre un corps et un objet et les contraintes d'auto-collision entre deux corps du même robot, les contraintes d'équilibre qui permet d'assurer l'équilibre d'un objet à manipuler, les contraintes des couples qui assurent que les limites des couples du robot ne dépasse pas les valeurs maximales et les contraintes de contact qui permet de créer un contact entre deux objets en appliquant une force. La figure 3.1 montre clairement les contraintes citées.

Par conséquent, nous aurons comme **sorties** la position précise des points de contact et la posture créée par ces contacts.

Ces entrées/sorties seront bien détaillées dans les section suivantes de ce chapitre. En plus, nous allons décrire une architecture très générique de ce générateur et nous allons présenter ensuite des résultats.

3.3 Critères

Le critère d'optimisation permet de faire le choix d'une configuration dans le sous-ensemble solution de l'espace de configuration lorsque celui-ci est non vide.

Le choix du critère dépend de l'utilisation faite du générateur de posture. Si l'on utilise comme un test de compatibilité d'un ensemble de tâches, il n'est même pas utile d'avoir un critère. Soit l'optimisation trouve un point faisable, soit l'ensemble des solutions est déclaré vide. Si on veut adopter les postures trouvées par un robot réel, on pourra préférer maximiser une marge de stabilité.

Si l'on s'attache à l'apparence du robot, il est classique de rechercher une posture ressemblant à celle d'un être humain. De nombreux critères ont été proposés pour cela. On pourra citer la minimisation de distance entre la position courante Pos_i et la position finale Pos_{finale} de l'objet mobile O qui donne de bons résultats, ou des notions de confort de la posture. Plus simple et plus rapide, la distance à une position de référence Pos_{final} s'écrit sous la forme

$$obj_1 = \|Pos_i(O) - Pos_{final}(O)\|^2 \tag{3.1}$$

La minimisation de distance est le critère que nous appliquons en premier lieu dans ce travail. En outre, nous adoptons aussi le critère appliqué aux couples qui assure le confort du robot. Ce critère s'écrit sous cette forme

$$obj_2 = \|\tau(q)\|^2 \tag{3.2}$$

mais avec un poids w inférieur à celui de distance. Le critère distance est plus intéressant pour nous puisqu'il assure le rapprochement du posture souhaitée. Notre critère fonctionne comme suit : chaque critère donne une valeur d'évaluation comme retour et après nous faisons la somme cumulée des deux

$$obj = w_1 obj_1 + w_2 obj_2 \tag{3.3}$$

Le générateur de posture doit trouver la meilleur posture qui optimise ce critère.

3.4 Contraintes

3.4.1 Collision

Pour éviter les collisions entre le robot et un obstacle, nous avons utilisé un volume englobant les corps par des boîtes arrondies. Cette approche de volume englobant permet d'assurer une marge de sécurité ϵ entre le corps de robot et l'obstacle.
Soient deux objets quelconques O_1 et O_2 d'un corps du robot ou parties de l'environnement, on souhaite pouvoir écrire une contrainte g, telle que

$$\begin{cases} g(q) > 0 & si O_1(q) \cap O_2(q) = \emptyset \quad (3.4) \\ g(q) = 0 & si O_1(q) \cap O_2(q) = \partial O_1(q) \cap \partial O_2(q) \quad (3.5) \\ g(q) < 0 & sinon \quad (3.6) \end{cases}$$

et vérifiant les propriétés nécessaires, exposées au paragraphe 3.6.1. La distance euclidienne d classique est la base pour les contraintes de collision. Cette distance est signée, cela signifie qu'elle devient négative lorsque les objets sont en interpénétration. Sa valeur absolue mesure alors la norme de la plus petite translation possible nécessaire pour séparer les objets. On a alors l'écriture suivante pour la contrainte de non collision :

$$d(O_1, O_2) \geq \epsilon \quad (3.7)$$

on note P_1(q) et P_2(q) les points témoins de la distance, c'est-à-dire les points de O_1(q) et O_2(q) respectivement pour lesquels la distance est réalisée : d(O_1(q), O_2(q)) = $\|P_1(q)P_2(q)\|$. On note plus simplement d(q) cette distance. A q fixé, les coordonnées $\mathbf{P_1}$ et $\mathbf{P_2}$ de P_1 et P_2 dans les repères liés à O_1 et O_2 sont les solutions du problème d'optimisation suivant, quand les objets *ne sont pas en pénétration* :

$$\min_{\mathbf{P_1},\mathbf{P_2}} \|Trans1(q)\mathbf{P_1} - Trans2(q)\mathbf{P_2}\| \quad (3.8)$$

Si les deux objets sont *en cas de pénétration* la distance s'écrit alors sous la forme d'un problème de maximisation.

Le pseudo-code du contrainte collision est donné par l'algorithme 1. en utilisant la

Algorithm 1 BoxCollisiontConstraint

```
1: Data : robot1, robot2, body1, body2
2: Result : retourner la contrainte optimisée
3: for i ∈ {1, ..., Body1.size()} do
4:     for j ∈ {1, ..., Body2.size()} do
5:         getFrameCoordinate(body1, Trans1)
6:         getFrameCoordinate(body2, Trans2)
7:         ComputeDistance(trans1, trans2, O1, O2);
8:     end for
9: end for
```

transformation (*Trans1, Trans2*) entre le repère de corps et le repère de l'environnement ;

Cet algorithme permet de parcourir les corps de deux objets (qui sont considérés comme des robots sans articulations) pour obtenir leurs coordonnés dans le repère de l'environnement W à travers la transformation de repère (*Trans1, Trans2*) et après calculer la distance entre ces deux corps.

3.4.2 Équilibre

Compte tenu des lois de la dynamique rigide du corps appliquée à R, la condition de stabilité statique est simplement écrite

$$\sum_{f \in F} f + mg = 0 \tag{3.9}$$

Où F est l'ensemble de toutes les forces de contact appliquées au robot, m est la masse du robot et g le vecteur de gravité.
Pour simplifier, nous avons modélisé tout contact de surface en tant que jeu discret de contacts ponctuels appliqués sur des points choisis répartis sur la surface de contact (nous ne l'expliquons pas intentionnellement dans nos formules à des fins de lisibilité). Chaque force de contact $f \in F$ appliquée sur le robot à un point P avec un N normal est dans un cône de frottement $\mathscr{C}_{P,N,\theta}$ (cf. Figure 3.2), θ étant l'angle du cône qui dépend du coefficient de frottement entre le corps et le corps obstacle, P est le sommet du cône, et N définit l'axe de révolution du cône.

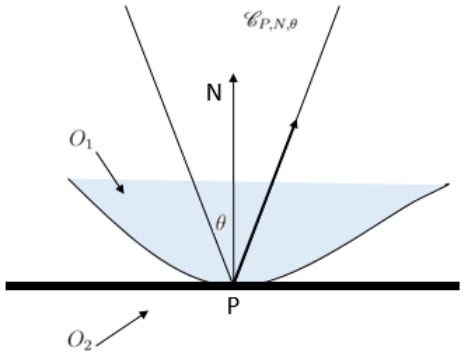


FIGURE 3.2 – Cône de frottement.

3.4.3 Contacts

La contrainte contact est composée d'une contrainte collision avec une distance nulle (exprimé par l'équation (3.10)) entre les deux corps en contact et un effort appliqué dans ce point de contact .

$$d(O_1, O_2) = 0 \tag{3.10}$$

La création d'une contrainte de contact ajoute des variables et des contraintes à notre problème d'optimisation. Pour chaque contact il faut rajouter le point d'application P des forces (qui doit appartenir aux 2 surfaces) ainsi que l'effort 3D qui doit appartenir au cône de frottement ($f_x, f_y, f_z \in \mathscr{C}_{A,N,\theta}$).

Par conséquent, chaque contact est défini par 3 contraintes mathématiques.

Le pseudo-code du contrainte contact est donné par l'algorithme 2.

Cette contrainte a besoin comme entrées les noms des robots (*robot1, robot2*) et les noms de corps de chaque robot (*body1, body2*) où on va créer le contact. Les paramètres utilisés sont *F, P, N, ContactPoint1, ContactPoint2* qui sont des vecteurs de tailles 3 contenant les coordonnées

Algorithm 2 BoxContactConstraint

Data : robot1, robot2, body1, body2
Result : Satisfaire la contrainte
Parameters : F, P, N, ContactPoint1, ContactPoint2
for $i \in \{1, \ldots, Body1.size()\}$ **do**
  for $j \in \{1, \ldots, Body2.size()\}$ **do**
    for $k \in \{1, \ldots, 3\}$ **do**
      $P[k] \leftarrow$ coordonnées de contact
      $F[k] \leftarrow$ coordonnées de force de contact
    end for
    ComputeDistance(Trans1, P, ContactPoint1)
    ComputeDistance(Trans2, P, ContactPoint2)
    ComputeForce(trans1, trans2, N) ;
  end for
end for

en x, y et z. Pour chaque contact body1-body2, on calcule les cordonnées de point de contact, la force et la distance entre les deux en utilisant la transformation (*Trans1, Trans2*) entre le repère de corps et le repère de l'environnement ; de même on calcul la force appliquée à cet point de contact.

3.4.4 Couple

La contrainte des couples se traduit par les équations suivantes :

$$|\tau| \leq \tau_{max} \tag{3.11}$$

$$\tau = G(q) + J(q)^T F \tag{3.12}$$

Où :

- τ_{max} est le couple maximal
- G(q) le vecteur de gravité
- J(q) la matrice jacobienne des articulations
- F désigne l'ensemble des forces de contact appliquée au robot

Les valeurs de couple doivent être limitées par la valeur maximale τ_{max} et les forces de contact f appliquées au robot aux points de contacts doivent être inclus dans la cône de frottement $\mathscr{C}$.

3.5 Formulation

Généralement, le robot doit réaliser un certain nombre de tâches et nous cherchons une configuration répondant à ce problème. Une tâche est un objectif à atteindre, qui est décrit par un ensemble d'égalité et d'inégalité, écrite dans un espace lié au robot. Par exemple, éviter une collision consiste à garder la distance entre un corps et un objet strictement positive, ou entre deux corps (auto-collision), dans ce même espace. Un contact consiste à imposer une distance nulle entre une partie d'un corps du robot et une partie de corps de l'environnement. On considère les forces **F** appliquées sur le robot et les couples τ [22].

Le problème se écrit comme suit

$$
\begin{aligned}
&trouver \quad q, F, P \\
&tel\ que \quad min\ obj(q, F, P) \\
&et \begin{cases}
q_i^- \leq q_i \leq q_i^+, \forall i \in \{1, \ldots, n\} & (3.14)\\
\tau_i^- \leq \tau_i \leq \tau_i^+, \forall i \in \{1, \ldots, n\} & (3.15)\\
\epsilon_{ij} \leq d(r_i(q), r_j(q)), \forall (i,j) \in \Gamma_{auto} & (3.16)\\
\epsilon_{ik} \leq d(r_i(q), O_k), \forall (i,k) \in \Gamma_{coll} & (3.17)\\
\tau + J(q)^T F = G(q) & (3.18)\\
F \in \mathscr{C}_{P,N,\theta} & (3.19)\\
P \in (O_1\ et\ O_2) & (3.20)
\end{cases}
\end{aligned}
$$

- n est le nombre des articulations dans un robot.
- ϵ_{ij} et ϵ_{ik} désignent les distances de sécurité à respecter pour un couple d'objets donné. Ordinairement, les ϵ_{ij} et les ϵ_{ik} sont égaux entre eux et positifs.
- Γ_{auto} et Γ_{coll} sont respectivement des sous-ensembles d'auto-collision et de collision.
- d(A,B) désigne la distance entre deux objets.

Les inégalités (3.14) sont les limites sur les valeurs des articulations. On considère dans notre cas que toutes les articulations n'ont qu'un seul degré de liberté (liaison glissière ou pivot). Pour des articulations plus complexes (par exemple liaison rotule), il faudrait écrire les valeurs limites des articulations sous une forme plus générale.
Les inégalités (3.15) donnent les limites sur les couples. L'équation (3.16) représente les contraintes de non auto-collision et (3.17) représente les contraintes de non collision.
L'équation (3.18) décrit la relation entre les forces et les couples dans le cas statique.
L'équation (3.19) impose que l'ensemble des forces doit appartenir au cône de frottement.
L'équation (3.20) impose que le point de contact P doit appartenir aux deux objets en contact.

Le nombre de paramètre du problème N_{param} est la somme des articulations n multiplié par le nombre de robot N_R et le nombre de contact N_{ctc} multiplié par 6 (la position et la rotation du point de contact) :

$$N_{param} = n \times N_R + 6 \times N_{ctc} \tag{3.20}$$

Chaque contrainte physique possède N_{ctr}^i contraintes mathématiques. Par conséquent, le nombre total des contraintes N_{ctr} est la somme de tout les contraintes mathématiques N_{ctr}^i :

$$N_{ctr} = \sum N_{ctr}^i \tag{3.21}$$

3.6 Implémentation

3.6.1 Algorithme d'optimisation

La possibilité de présence de liaison pivot sur le robot, la géométrie de ce dernier dépend d'angles, implique qu'au moins une partie des contraintes sont non linéaires. Parmi les différentes méthodes existantes pour résoudre un problème avec de telles contraintes, nous avons choisi l'IPOPT.

IPOPT (Interior Point OPTimizer) est un logiciel open source pour l'optimisation non linéaire à grande échelle. Ipopt a été conçu pour être flexible pour une grande variété d'applications, et il existe plusieurs façons d'interagir avec Ipopt qui permettent des structures de données spécifiques et des techniques de résolution linéaire. Néanmoins, les auteurs ont inclus une représentation standard qui devrait répondre aux besoins de la plupart des utilisateurs. Il peut être utilisé pour résoudre des problèmes généraux de programmation non linéaire de la forme [36] :

$$\min_{x \in \Re^n} \quad f(x) \tag{3.22}$$

$$s.t. \quad g^L \leq g(x) \leq g^U \tag{3.23}$$

$$x^L \leq x \leq x^U \tag{3.24}$$

avec $x \in \Re^n$ sont les variables d'optimisation (éventuellement avec des limites inférieures et supérieures x_L et x_U), f : $\Re^n \Rightarrow \Re$ est la fonction objective, g : $\Re^n \Rightarrow \Re^m$ sont les contraintes non linéaires générales. Les fonctions f (x) et g (x) peuvent être linéaires ou non linéaires et convexes ou non convexes (mais doivent être deux fois continûment différentiable). Les contraintes, g(x), ont des limites inférieures et supérieures. Les contraintes d'égalités sont de la forme $g_i(x) = \bar{g}_i$ peut être spécifié en définissant $g_i^L = g_i^U = \bar{g}_i$.

Pour résoudre un problème, Ipopt a besoin de plus d'informations que la définition du problème (par exemple, l'information dérivée). Les informations que l'Ipopt a besoin pour résoudre un problème sont :

1. Dimensions du problème
   - nombre de variables
   - nombre de contraintes
2. Limites de problème
   - les limites des variables
   - les limites des contraintes
3. Point de départ initial
   - Valeurs initiales pour les variables primitives de x
4. La structure du problème
   - Nombre des variables non-nulles dans le jacobien des contraintes
   - Structure de sparcité [1] du jacobienne des contraintes
5. Évaluation des fonctions du problème
   - La fonction objective, f(x)
   - Le gradient de l'objective, $\bigtriangledown f(x)$
   - Les valeurs de la fonction de contrainte, g(x)
   - La jaccobienne des contraintes, $\bigtriangledown g(x)^T$

Les dimensions et les limites du problème sont simples et proviennent uniquement de la définition du problème. Le point de départ initial est utilisé par l'algorithme lorsqu'il commence à itérer pour résoudre le problème. Si Ipopt a de la difficulté à converger, ou s'il converge vers un point localement impossible, le réglage du point de départ peut aider. Selon

1. Indices de ligne et de colonne d'une matrice, le terme en anglais : *sparsity*

le point de départ, Ipopt peut également converger vers différentes solutions locales. Fournir la structure de sparsité des matrices dérivées est une partie un peu plus compliquée. Ipopt est un solveur de programmation non linéaire conçu pour résoudre des problèmes à grande échelle. Avant de résoudre le problème, Ipopt doit connaître le nombre d'éléments non nuls et la structure de sparcité (indices de ligne et de colonne de chacune des entrées non nulles) de la contrainte jacobienne.

Une fois défini, cette structure non nulle doit rester constante pour toute la procédure d'optimisation. Cela signifie que la structure doit inclure des entrées pour tout élément qui pourrait être non nul, non seulement ceux qui sont non-zéro au point de départ. Comme Ipopt l'indique, il faudra les valeurs pour l'étape 5 évaluée à des points particuliers. Avant que nous puissions commencer à coder l'interface, cependant, nous devons analyser les détails de ces équations symboliquement.

L'algorithme 3 présente les fonctions de notre optimisateur :

Algorithm 3 Ipopt algorithm

1: bool NLP-FAD-1-4 : :get-nlp-info ()
2: bool NLP-FAD-1-4 : :get-bounds-info ()
3: bool NLP-FAD-1-4 : :get-starting-point ()
4: bool NLP-FAD-1-4 : :eval-f ()
5: bool NLP-FAD-1-4 : :eval-grad-f ()
6: bool NLP-FAD-1-4 : :eval-g ()
7: bool NLP-FAD-1-4 : :eval-jac-g ()
8: void NLP-FAD-1-4 : :finalize-solution ()

3.6.2 Architecture globale du générateur de posture

Classiquement, la résolution d'un problème d'optimisation se fait par le biais d'un couple optimisateur-simulateur, cela du point de vue informatique. L'optimisateur s'occupe de la partie résolution numérique, alors que le simulateur se charge du calcul de la valeur du critère ou des contraintes pour chaque point choisi par l'optimisateur. Le découplage n'est pas facile, parce que l'optimisateur n'a aucune idée du problème qu'il résout et de même le simulateur n'a aucune idée du calcul réalisé par l'optimisateur. L'échange entre les deux se fait par le biais d'une demande par l'optimisateur de la fonction du contrainte en fournissant un critère en un point x, et l'optimisateur répond à cette requête. Dans notre cas, x désigne les valeurs articulaires du robot q, les forces appliquées à un point de contact et P le point de contact. Un certain nombre des hypothèses sont établis sur ce que chacun doit attendre de l'autre, tel que par exemple le bon conditionnement du problème.

Cette approche est conservée par le générateur de posture (cf Figure 3.3). Notre optimisateur ici l'ipopt que nous encapsulons dans une boite OptimSolver. Le simulateur de l'autre coté se nomme MogsPosture dans notre architecture et rassemble l'ensemble des contraintes ainsi que l'ensemble de critères, qui sont des fonctions se basant sur des calculs de géométrie directe de robot.

Nous détaillons les différents niveaux de la partie simulateur :

MogsCore : c'est une classe en charge de calcul lié à la géométrie directe du robot, à sa modélisation dynamique et cinématique, etc. On peut par exemple l'utiliser pour mettre à jour la dynamique du robot c'est à dire récupérer les valeurs courantes des articulations et par la suite mettre à jour la position du robot.

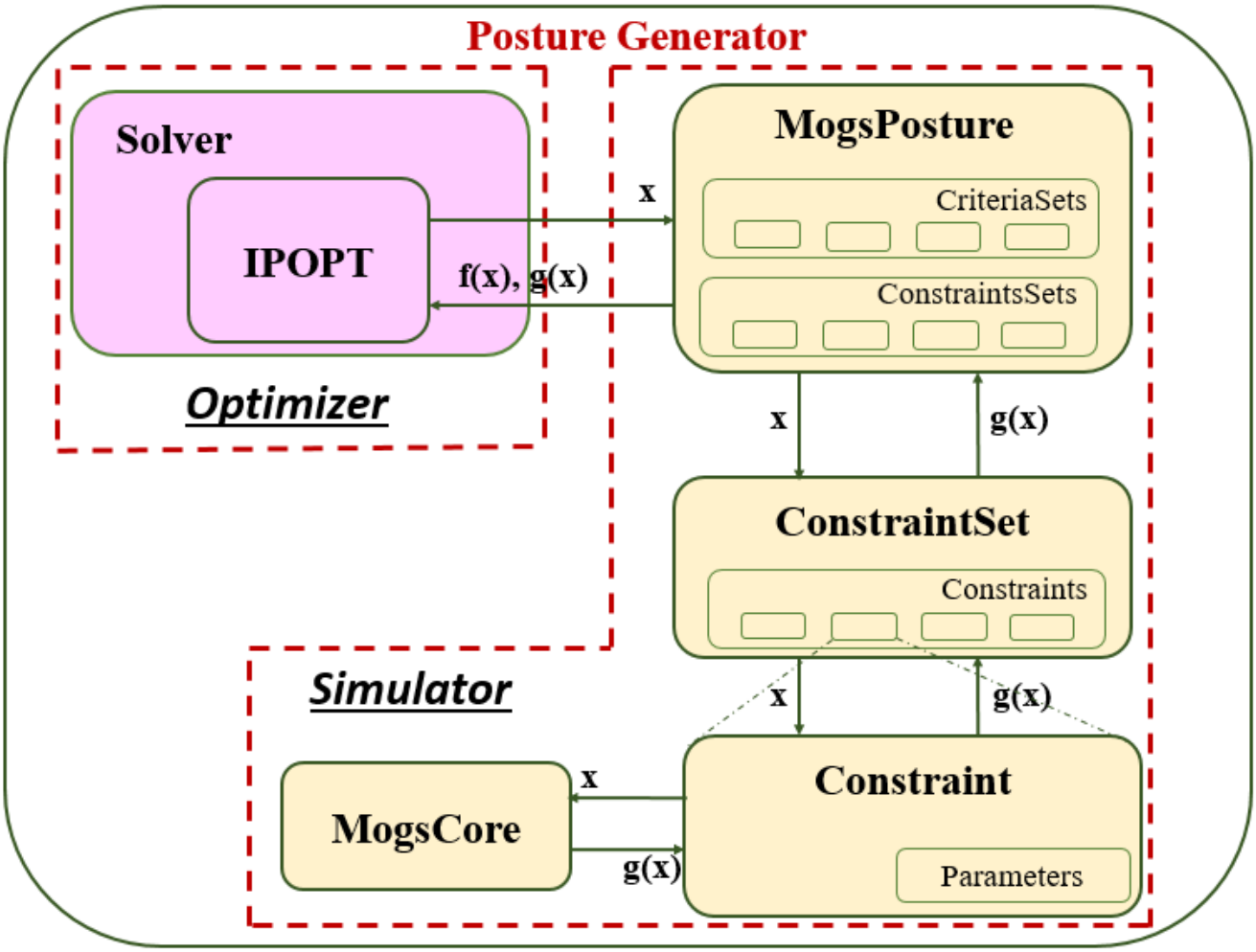


FIGURE 3.3 – Architecture du générateur de posture. L'optimisation se fait par un dialogue entre l'IPOPT et le MogsPosture, dans lequel l'optimisateur (en rose) demande l'évaluation d'une fonction (contrainte ou objectif) et reçoit une réponse du simulateur (en jaune). La demande est transmise à la base de la hiérarchie à la contrainte, qui repose sur les calculs de géométrie directe faits par le MogsCore et fournis via la fonction g.

Constraint : une contrainte physique est une fonction de $\Re^n$ dans $\Re^m$ où m est le nombre des contraintes mathématiques. Chaque classe contrainte contient son propre fonction qui permet de spécifier ses propres informations les limites et les conditions initiales. Dans notre cas, un objet contraint doit pouvoir renvoyer la valeur de la dynamique pour chaque articulation q. La fonction implémentée est choisie parmi une liste de fonctions et spécifiée par une série de paramètres.

ConstraintsSet : un ensemble de contraintes. Typiquement, une configuration, ou plus généralement une posture, est un ensemble des contraintes en ajoutant un critère. Ce groupe permet à l'utilisateur de faire des manipulations de haut niveau sans rentrer dans les détails. Dans le cadre d'un contact par exemple, il suffit à l'utilisateur de donner les noms des robots (un objet est considéré comme un robot sans articulation) et les noms de leurs corps où il veut créer le contact ainsi que leurs configurations courantes (position et rotation). Une fois c'est fait, un contact se génère automatiquement.

MogsPosture : c'est le simulateur. Il regroupe l'ensemble des contraintes et maintient lui

même à jour en fournissant un critère pour donner une posture.

Le processus de fonctionnement d'un générateur de posture est décrit par l'algorithme 4

Algorithm 4 SolveOnePosture

```
1: Data : dyns, robots, bodies, Position
2: Result : σ
3: Crit is a vector of Criteria
4: nb ← criteria.size()
5: for i ∈ {1, . . . , robots.size()} do
6:     TorqueCriteria()
7:     PositionCriteria()
8:     ContactConstraint()
9:     CollisionConstraint()
10:    BalanceConstraint()
11:    TorqueConstraint()
12: end for
13: for i ∈ {1, . . . , dyns.size()} do
14:    dyns[i] = UpdateStaticDynamic()
15: end for
16: for i ∈ {1, . . . , nb} do
17:    tmp ← crit[i].compute(dyns)
18:    eval + = tmp
19: end for
```

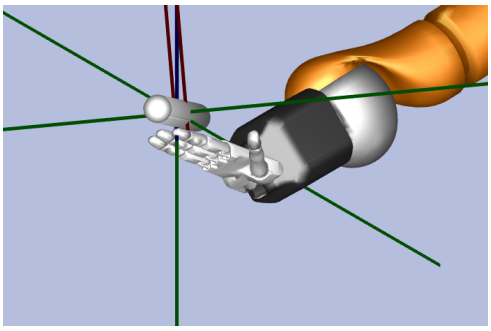
Cet algorithme a besoin comme entrés le dynamique de robot, les noms du robots utilisés et les noms de leurs corps et la position souhaitée. Il donne comme sortie une posture composée d'un ou plusieurs contacts. Cri est vecteur rempli par des critères, dans notre cas on a deux. Ensuite, pour chaque robot, on doit optimiser les contraintes.

La fonction **UpdateStaticDynamic** assure la mise à jour de la dynamique du robot et ce qui nous permet de mettre à jour la position PosCurr de chaque posture.

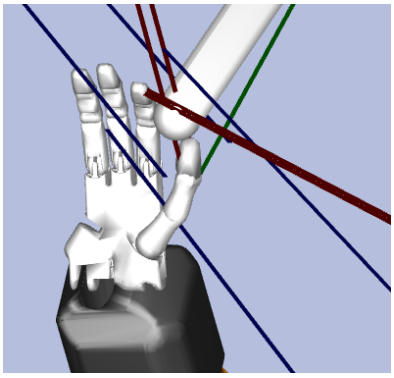
Enfin , calculer la somme des valeurs retournés par chaque critère.

3.7 Résultats

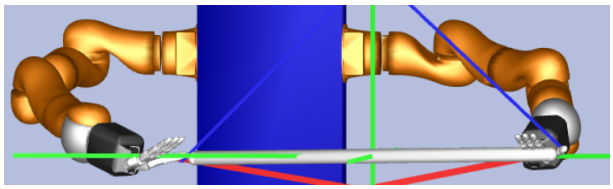
Nous avons tester notre algorithme pour générer des postures contenant un ou plusieurs contacts entre des différents corps du robot et un bâton. Nous avons trouvé les résultats présentés par la figure 3.4.



(a) Contact entre un doigt du robot et un bâton



(b) Contact entre deux doigts du robot et un bâton



(c) Contact entre deux bras d'un même plateforme et un bâton

FIGURE 3.4 – Tester des postures en utilisant des différents contacts

3.8 Conclusion

Dans ce chapitre, nous avons donné une méthode pour générer des postures stables, sans collision et réalisant des tâches. Elle repose sur l'écriture et la résolution d'un problème d'optimisation sur des variables liées au robot (les degrés de liberté, les articulations, les forces de contact,...). Nous avons présenté la façon mathématique qui décrit ce problème et ses diverses composantes, ainsi que la forme et l'algorithme de son implémentation. Cette méthode nous a permis d'optimiser les principales contraintes qui lie le robot à son espace de configuration. Le temps d'exécution pour générer une seule posture est de l'ordre de seconde ou dizaine de seconde. Nous avons utilisé l'architecture logicielle MOGS pour implémenter notre problème.

Nous exposons dans le chapitre suivant une méthode généralisée qui permet de générer une séquence de posture en utilisant une arbre de décision. Celui ci est basée sur une métrique qui va permettre au robot de se rapprocher de son but.

Chapitre 4

Génération d'une séquence de postures

Sommaire

4.1 Introduction

Nous passons aux systèmes à grande échelle à partir d'ici et nous proposons dans ce quatrième chapitre un cadre généralisé avec un algorithme pour planifier une séquence de positions multi-contacts qui concerne un ensemble de robots collaborateurs et un objet manipulé à partir d'une configuration initiale spécifiée à un but désiré. L'approche proposée est basée sur la planification d'une séquence possible de positions avant de planifier le mouvement continu qui suit cette séquence de positions planifiée.

L'algorithme que nous proposons ici est de créer un arbre de recherche. Cet arbre permet de sélectionner les postures faisables, parmi touts les postures possibles, qui peuvent accomplir une tâche demandée par l'utilisateur. Le principe de cette méthode est d'utiliser une métrique pour comparer et trouver les meilleures postures.

Le reste de de ce chapitre est organisé comme suivant. Nous proposons d'abord une formulation du problème en définissant le système (Section 4.2). Ensuite, nous écrivons notre algorithme dans le langage synthétique en donnant deux méthode (Section 4.3). Puis, nous démontrons certains résultats obtenus par notre planificateur sur différentes classes de problèmes (Section 4.4). Enfin, nous évaluons les résultats obtenus (section 4.5).

4.2 Planification multi-contacts pour plusieurs agents

Pour chaque tâche de manipulation, un problème d'optimisation des contraintes est résolu hors ligne pour trouver une séquence de postures optimales associées à un chemin de manipulation désiré, au voisinage d'une posture initiale donnée. Les contraintes basées sur les tâches devraient tenir compte des contraintes physiques liées à la structure du corps du robot et aux relations géométriques entre lui et l'environnement.

Le contact avant la mise en mouvement est une approche de planification qui prévoit d'abord la séquence de contacts qui sera ajoutée ou supprimée à chaque étape du mouvement, puis planifie le mouvement continu qui sera constitué de mouvements d'étapes élémentaires sur des configurations de contacts fixes. Les deux étapes de la planification du contact avant le mouvement peuvent être entrelacées ou découplées. Dans notre cas, en suivant la stratégie découplée [24], nous nous concentrons sur la première étape : la planification de la séquence des positions et des postures. Notre étude de ce problème réside dans la généralisation du cadre aux systèmes multi-robots. Cette généralisation permet une description et un traitement des problèmes de manipulation, soit pour un seul robot, soit pour plusieurs robots collaborateurs.

4.2.1 Système

Supposons que nous ayons un système de N robots indexé dans l'ensemble $\{1, \ldots, N\}$. Un robot ici est soit un mécanisme articulé entièrement ou sous-actionné, soit un objet manipulé non actionné. L'environnement peut également être considéré comme un cas particulier de robot (non actionné).

Ainsi, l'ensemble d'indice {0, …, N} contient tous les mécanismes articulés. Chaque robot

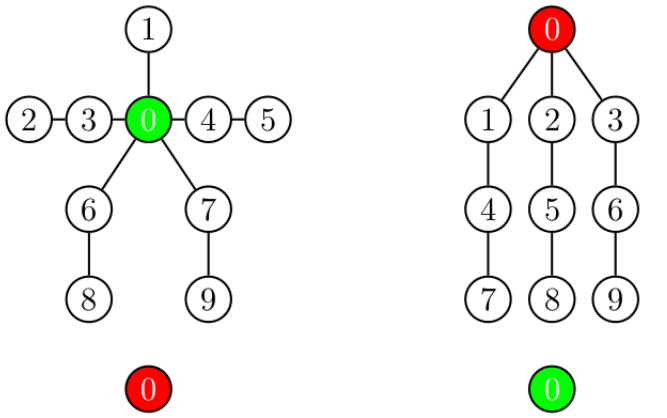


FIGURE 4.1 – Exemples des 4 types d'arbres cinématiques donnant un espace de configuration. En haut à gauche : robot à base flottant. En haut à droite : robot à base fixe. En bas à gauche : l'environnement. En bas à droite : un objet rigide. En rouge : base fixe. En vert : base flottante libre. Un système de robots se compose d'un nombre arbitraire de l'un de ces 4 types d'arbres cinématiques [24].

$r \in \{0, \ldots, N\}$ peut être représenté comme un arbre cinématique (cf. Figure 4.1) constitué de corps b_r corps (noeuds de l'arbre) indexés dans {0, …, b_r - 1}, liés par j_r joints actionnés (bords de l'arbre) indexés dans $\{1, \ldots, j_r\}$ (ou $\emptyset$ si j_r = 0).

— b_r = 1 et j_r = 0 si r se réfère à l'environnement ou à un objet manipulé.

— L'indice 0 dans l'ensemble $\{0, \ldots, B_r - 1\}$ se réfère au corps racine de l'arbre cinématique représentant le robot r.

La configuration q du système est un élément de l'espace de configuration

$$C = \prod_{r=1}^{N} C_r \tag{4.1}$$

le produit cartésien des espaces de configuration des robots du système.

Par conséquent, un élément de C est définit par :

$$q = (q_1, \ldots, q_N), \tag{4.2}$$

Où, pour $r \in \{1, \ldots, N\}$,

- $Q_r = (\theta_{r,1}, \ldots, \theta_{r,j_r})$, si r se réfère à un bras manipulateur à base fixe tel que le bras Kuka, par exemple.
- $Q_r = (x_r, y_r, z_r, \alpha_r, \beta_r, \gamma_r)$, si r se réfère à un objet rigide non articulé manipulé.
- Q_r n'est pas défini pour r = 0 (l'environnement).

4.2.2 Formulation de problème

Un contact est établi entre deux surfaces s_1 et s_2 appartenant à deux liaisons différentes (les entités r_1 et r_2 peuvent être égales). Nous désignons c un tel contact, et E_{ctc} l'ensemble de tout les contacts possible. (x_c, y_c, ϑ_c) désigne la position relative et l'orientation des deux surfaces s_1 et s_2 lorsque le contact c est établi (Voir figure 4.2).

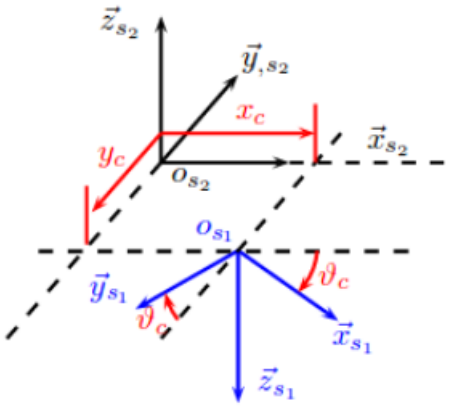


FIGURE 4.2 – Paramètres d'un contact

Une posture σ est un ensemble fini de contacts $\sigma = \{c_1, \ldots, c_{n_\sigma}\}$ de sorte que chaque surface apparaît au plus une fois. Nous désignons Σ l'ensemble de toutes les postures possibles (ce qui est ainsi un sous-ensemble de $2^{E_{ctc}}$). Pour chaque posture σ nous désignons $n_\sigma = card(\sigma)$ le nombre de contact dans σ. Une posture σ est dite adjacente à une posture σ' si elle diffère de σ par un seul contact (ajout ou une suppression d'un contact). Nous désignons $Adj^+(\sigma)$ l'ensemble des postures σ' où nous ajoutons un contact i.e $n_{\sigma'} = n_\sigma + 1$ et $\sigma \subset \sigma'$, et nous désignons $Adj^-(\sigma)$ l'ensemble des postures σ' où nous supprimons un contact i.e

$n_{\sigma'} = n_\sigma - 1$ et $\sigma' \subset \sigma$. Finalement, nous définissons $Adj(\sigma) = Adj^+(\sigma) \cup Adj^-(\sigma)$ l'ensemble de toutes les postures adjacentes à σ.

Nous avons maintenant défini l'espace de configuration C et l'ensemble des postures σ. Les configurations qui réalisent une posture σ sont soumises à des contraintes physiques dans C. Ces contraintes sont :

— Équilibre statique.

— Forces de contact dans les cônes de frottement.

— Couples limites.

— Collision.

Enfin, nous pouvons formuler notre problème : compte tenu d'une posture initiale et finale σ_{init} et σ_{final} dans Σ, trouver une séquence de postures $(\sigma_1, \ldots, \sigma_n)$ tel que :

- $\sigma_1 = \sigma_{init}$.
- $\sigma_n = \sigma_{final}$.
- $\sigma_{i+1} \in Adj(\sigma_i) \quad \forall i \in \{1, \ldots, n-1\}$

Pour un système où Σ de grande dimension, on divise Σ en F familles disjointes $\Sigma_1 \ldots \Sigma_F$. Une famille Σ_F (cf. Figure 4.3) est composée des fils qui permet le passage d'une posture à une autre adjacente, $Adj(\sigma) \subset \Sigma_F$. Chaque posture $\sigma \in \Sigma$ est défini par un unique co-paramètre θ. Une posture peut être identifié par une représentation de configuration, i.e. q_1 identifie par σ_1 et q_2 identifie par σ_2.

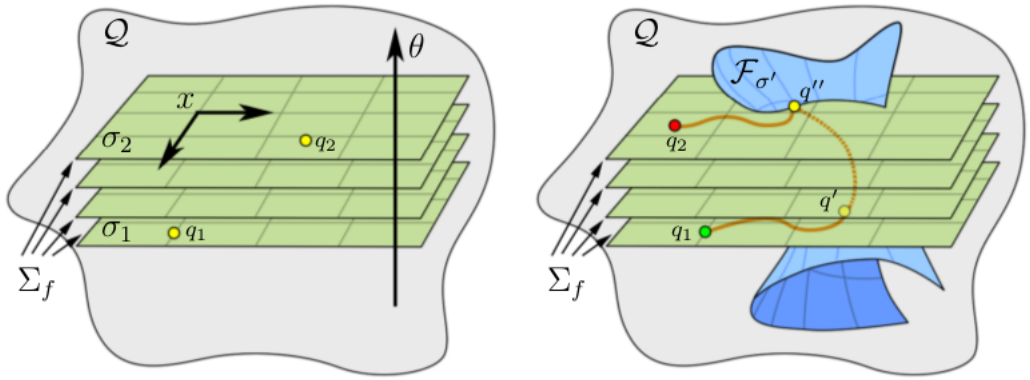


FIGURE 4.3 – figure droite : une famille des postures Σ_F, chaque valeur de θ défini une différente posture. Figure gauche : Se déplaçant dans Σ_F, de q_1 à q_2, nécessite une transition vers une posture différente σ sur q' et revenir sur q''

4.3 Algorithme de recherche

4.3.1 Première méthode

Pour explorer l'ensemble de postures possibles Σ nous avons utilisé un arbre de recherche. En commençant par $\sigma = \sigma_{init}$, nous explorons les postures $\sigma^{'}$ qui sont adjacent à σ, nous cherchons les configurations de transition. Puis nous prenons la **meilleure posture** σ^* et nous réitérons le processus en définissant $\sigma := \sigma^*$ jusqu'à trouver $\sigma^{'}$ qui est assez proche de

σ_{final}.
La posture σ^* est la meilleur si elle possède la meilleur métrique.

Deux structures de donnée sont nécessaires pour écrire cet algorithme : une queue prioritaire Q et un arbre d'exploration récursive T.

- La queue Q est définie par une fonction de coût nommée $Metric$. Cette fonction est attribuée à chaque paire de posture σ et de sa configuration q pour calculer le coût $Metric(\sigma, q)$. Celle-ci permet d'évaluer une posture si elle est la meilleure parmi les fils de la famille Σ_F. Le principe est de calculer la distance entre la position courante de l'objet mobile (à manipulé) et sa position finale désirée ainsi que la la valeur des couples du robot manipulateur. Le but est de minimiser le critère M renvoyer par cette fonction.
- L'arbre T est récursif parce qu'il fait l'appel à lui même. Il consiste essentiellement en
  - **nœuds** qui sont des postures avec la réalisation de la configuration,
  - **arêtes** qui sont dessinées entre des nœuds et encodent des positions adjacentes entre lesquelles existe une configuration de transition.

L'algorithme 5 donne un aperçu du processus de recherche.

Algorithm 5 SearchTree

Data : σ
Result : σ_{final}
$children \leftarrow Adj(\sigma)$
$n_{children}$ size of the vector $children$
v is a vector of visited posture σ
$\sigma^* \leftarrow \varnothing$
$Q \leftarrow \{\varnothing\}$
if $\sigma \notin v$ **then**
    $v \leftarrow \sigma$
end if
for $i \in \{1, \ldots, n_{children}\}$ **do**
    $\sigma' \leftarrow children[i]$
    if ($n_{\sigma'} > n_{\sigma_{min}}$ & $\sigma' \notin v$) **then**
        SolveOnePosture(σ')
        $M \leftarrow metric(\sigma', q)$
    end if
    $Q.push(\sigma', M)$
end for
while $Q \neq \varnothing$ **do**
    $\sigma^* \leftarrow Q.top()$
    $Q.pop()$
    SearchTree(σ^*)
    if ($\sigma^* \neq \varnothing$) **then**
        return σ^*
    end if
end while

Nous avons utilisé certains aspects techniques tels que l'évitement de la boucle infinie. Cet aspect permet de vérifier que lorsqu'une nouvelle posture σ est ajoutée, elle n'appartient pas déjà à T (en cas de marche d'un humanoïde, c'est vrai que les contacts se répètent mais ces

contacts ne donnent pas la même posture parce que les positions de contacts changent). Cela est traduit par $\sigma \notin v$, dans ce cas nous ajoutons σ à v. En plus, nous avons crée un vecteur $children \in \Sigma_F$ de taille $n_{children}$ qui contient les fils de σ.

La fonction "**push**" permet d'ajouter la posture σ' dans la queue Q dans sa position convenable selon la valeur du critère M. Cela est assuré par la structure basique de la queue prioritaire qui permet de classer ses attributs dans l'ordre décroissant. Dans notre cas, nous avons utilisé le classement croissant pour prendre à chaque itération la posture qui possède le critère minimal M.

La fonction "**top**" nous donne le premier élément de Q (la meilleur posture σ^*)

La fonction "**pop**" supprime le premier élément de Q.

La fonction "**SolveOnePosture**" permet de générer une posture.

Le variable $eval$ contient la somme retournée par tous les critères.

Cependant, l'inconvénient de l'algorithme 5 se résume dans le fait que la décision de meilleur choix est prise sans explorer tout l'arbre (cf. Figure 4.4). En d'autres termes, dans chaque niveau de l'arbre l'algorithme choisi le noeud ayant la métrique la petite et il la considère comme la meilleur noeud. L'algorithme donc explore le niveau 1 et choisit un seul noeud et voit ces fils, il passe au niveau 2 et choisir un fils (noeud) et ainsi de suite jusqu'à atteindre le but. Le problème c'est que l'algorithme n'explore plus les fils des autres noeuds d'un même niveau alors qu'il peut trouver un autre chemin meilleur si il fait la somme des valeurs cumulées de la métrique entre la posture initiale et la posture finale.

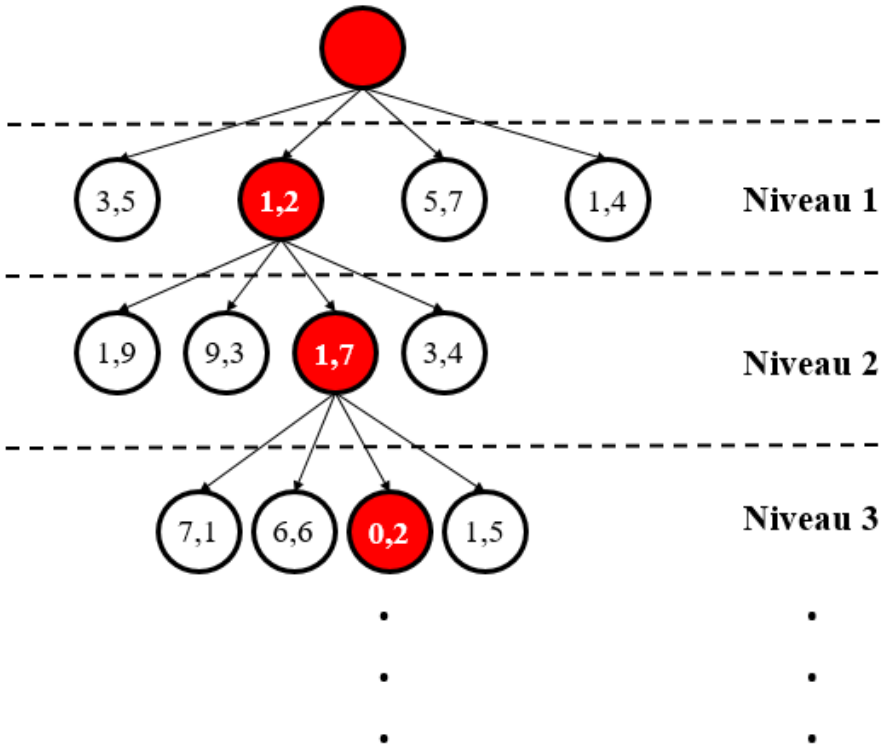


FIGURE 4.4 – Arbre de recherche avec des valeurs aléatoires sans aucune relation avec nos valeurs réelles.

Si, l'algorithme arrive au niveau n de l'arbre et il ne trouve pas une solution (Pas de contact faisables). Dans ce cas, il revient au niveau juste avant (n-1) et re-choisir un autre contact (cf. Figure 4.5).

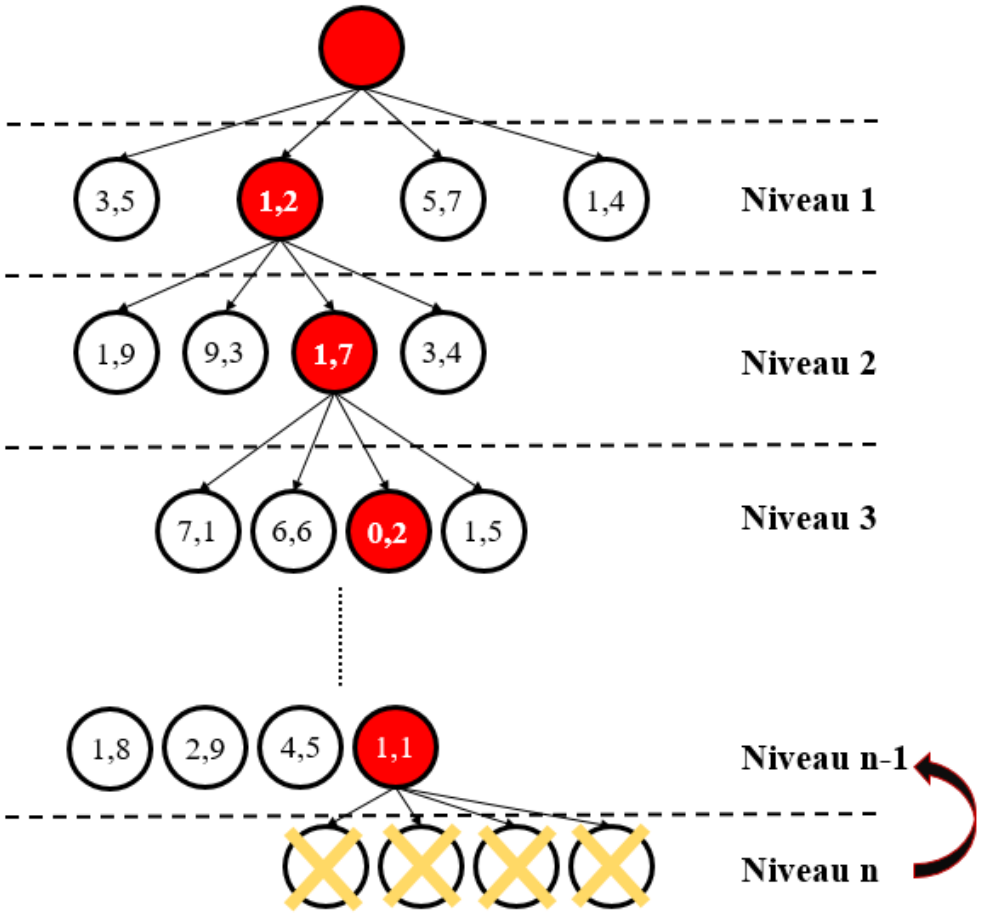


FIGURE 4.5 – Le cas où il n'y a pas des contacts faisables au niveau n de l'arbre.

4.3.2 Deuxième méthode

Dans le cas où la minimisation de la métrique est importante, nous proposons un autre algorithme qui permet d'explorer la totalité de l'arbre et enfin choisir le chemin qui a la **métrique cumulée** est minimale.

Le principe est de parcourir tout les noeuds de l'arbre et à chaque fois sommer la valeur de métrique du noeud courant avec les noeuds qui le précède (cf. Figure 4.7).
Prenant l'exemple dans les figures 4.4 et 4.7, nous avons trouvé dans la figure 4.4 que la noeud qui possède une valeur égale à 1.2 dans le premier niveau est la meilleur. Or, ce n'est pas le cas si on fait la somme, la figure 4.7 montre que la meilleure noeud dans le niveau 1 est la noeud qui comme valeur 1.4 et non pas qui a 1.2.

Notre proposition est traduit par l'algorithme 6 :

Algorithm 6 SearchTree2

Data : σ_{init}
Result : σ_{final}
$children \leftarrow Adj(\sigma)$
$n_{children}$ size of the vector $children$
v is a vector of visited posture σ
$\sigma^* \leftarrow \varnothing$
if $\sigma \notin v$ **then**
    $v \leftarrow \sigma$
end if
for $i \in \{1, \ldots, n_{children}\}$ **do**
    $\sigma' \leftarrow children[i]$
    if ($n_{\sigma'} > n_{\sigma_{min}}$ & $\sigma' \notin v$) **then**
        SolveOnePosture(σ')
        $M \leftarrow Metric(\sigma) + Metric(\sigma')$
    end if
    SearchTreeA(σ')
    if ($\sigma^* = \varnothing$ || $M < Metric(\sigma^*)$) **then**
        $\sigma^* \leftarrow \sigma'$
    end if
end for
return σ^*

Résultat : Nous avons testé cet l'algorithme sur le scénario 1 et il donne le même résultat de l'algorithme 5. Mais, malheureusement, lorsque nous avons essayé avec des tâches plus compliquées et de grande dimension, nous avons trouvés un problème de mémoire RAM. Après des heures d'exécution, le code se bloque à cause d'un erreur de bus. L'avantage de cet algorithme est la sûreté d'avoir un chemin minimal mais l'inconvénient se présente dans le temps d'exécution qui est très lent et le besoin d'une grande mémoire.

Nous proposons, dans ce cas, de tester l'algorithme avec une RAM de taille plus grande si l'optimalité de chemin est très importante. Sinon, si la tache souhaité contient un mouvement régulier et cyclique, la méthode de mouvement avant appuis sera plus rapide et plus efficace.

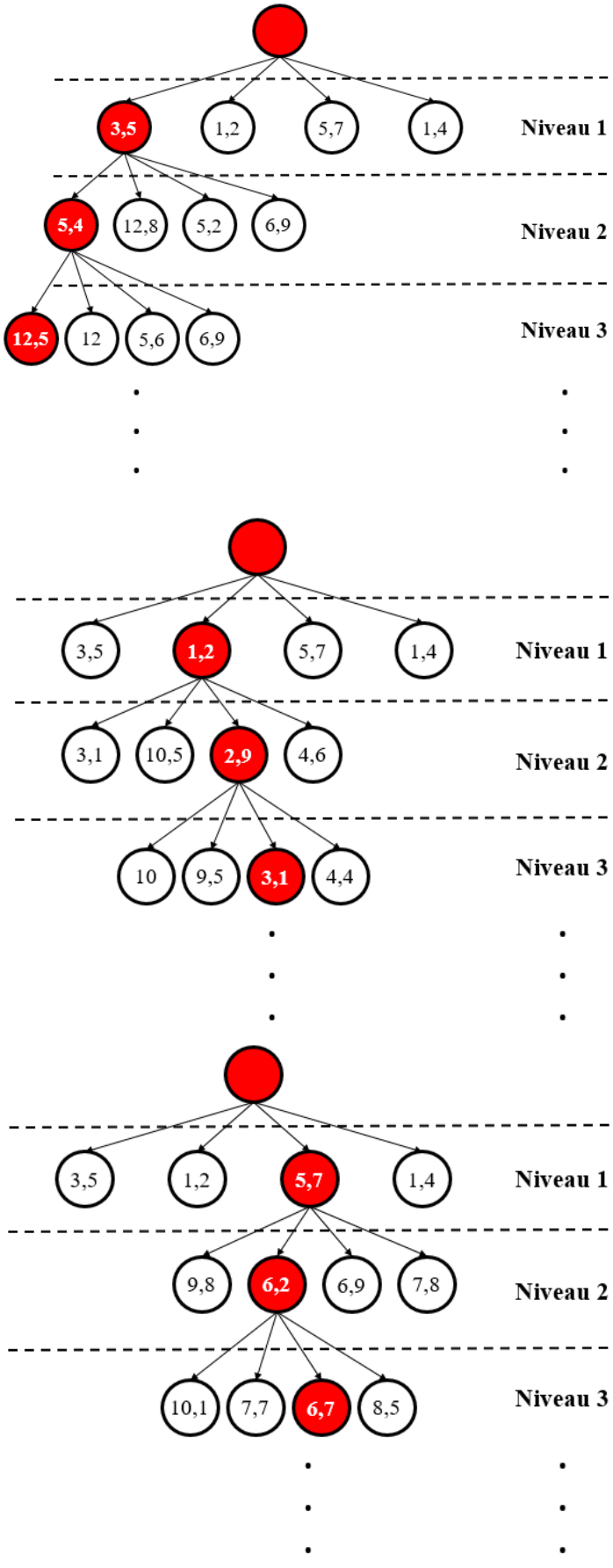
Dans la section suivante nous demandons au système des tâches particulières pour vérifier le bon fonctionnement de l'algorithme proposé.

4.4 Résultats

4.4.1 Premier scénario

Tout d'abord, nous avons essayé de tester l'algorithme de recherche sur un système S composé de deux robots, un Plateforme (composé de deux robots Kuka) et un bâton (robot sans articulation), S = {Plateforme, bâton}.
Le but est de faire passer le bâton du bras gauche au bras droit du même Plateforme.

Remarque 1 *Un chemin de σ à σ', pour $\sigma' \in Adj^-(\sigma)$ serait appelé un chemin de transit, et un chemin de σ à σ', pour $\sigma' \in Adj^+(\sigma)$ serait appelé un chemin de transfert.*


3,5
1,2
5,7
1,4
Niveau 1
5,4
12,8
5,2
6,9
Niveau 2
12,5
12
5,6
6,9
Niveau 3
3,5
1,2
5,7
1,4
Niveau 1
3,1
10,5
2,9
4,6
Niveau 2
10
9,5
3,1
4,4
Niveau 3
3,5
1,2
5,7
1,4
Niveau 1
9,8
6,2
6,9
7,8
Niveau 2
10,1
7,7
6,7
8,5
Niveau 3

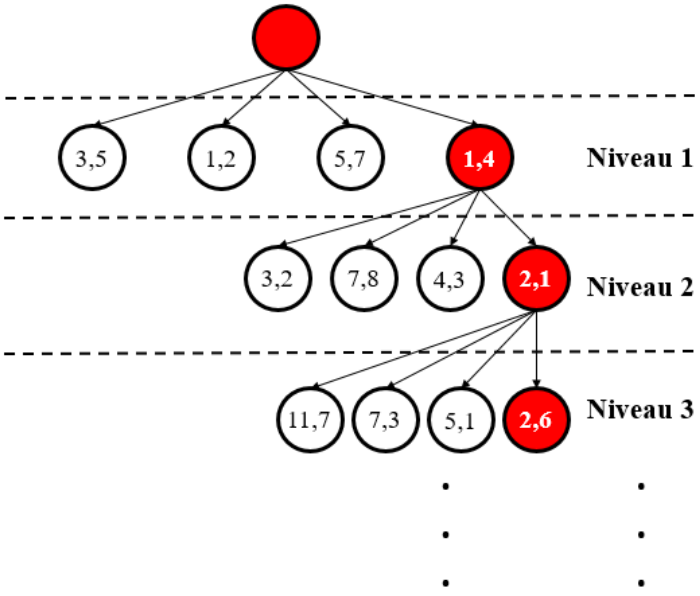


FIGURE 4.7 – Exemple d'exploration d'un arbre de recherche avec des valeurs aléatoires.

Interprétation du résultat : La configuration initiale q_{init} est créée par un contact C1 entre le bras gauche de Plateforme et le bâton. L'algorithme doit trouver la séquence optimale pour atteindre son but final. L'arbre de la Figure 4.8 montre le déroulement de cette séquence.

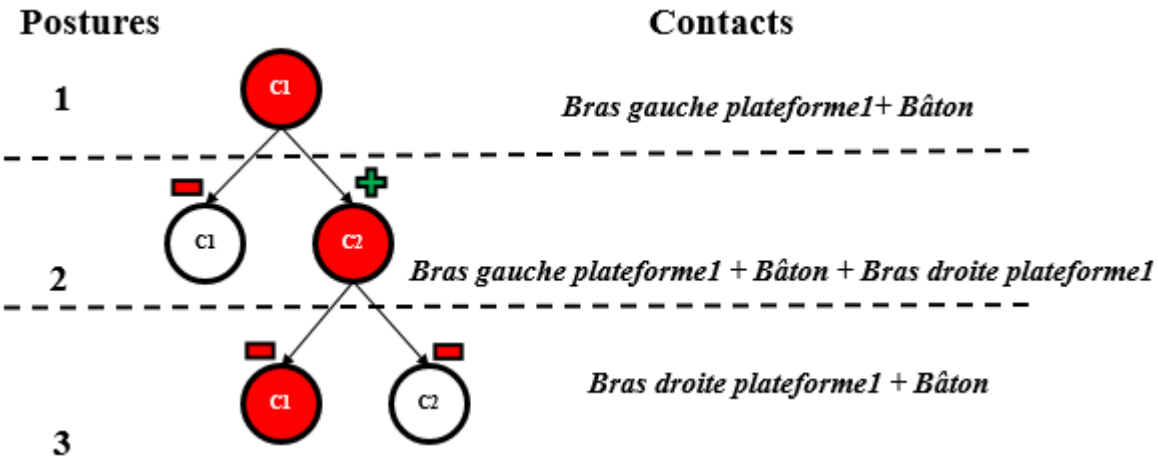
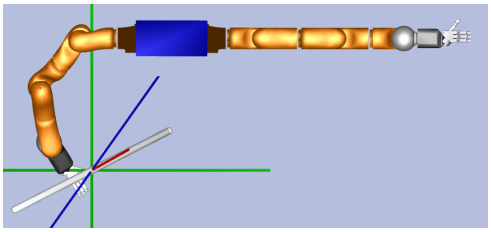


FIGURE 4.8 – Arbre d'exploration composé de deux contacts.

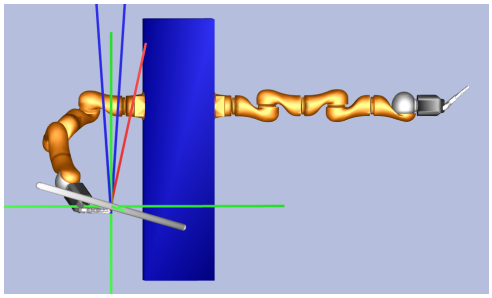
La cercle en rouge désigne que ce contact pris en compte alors que la cercle en blanc désigne que ce contact n'est pas pris. Nous mettons un signe (+) devant un contact si nous voulons l'ajouter sinon un signe (-) si nous voulons le supprimer.

Dans ce cas, en commençant par la première posture, nous avons 2 choix : soit ajouter (+) le contact C2, soit supprimer (-) le contact C1. La suppression de C1 n'est pas faisable parce qu'il nous faut au moins un contact donc l'ajout de C2. Ensuite, à partir de la deuxième posture, soit supprimer C1 soit supprimer C2. Après la suppression de C2 est nous aurons une posture déjà visitée et la suppression de C1 nous donne une posture identique à la posture finale donc nous supprimons C1.

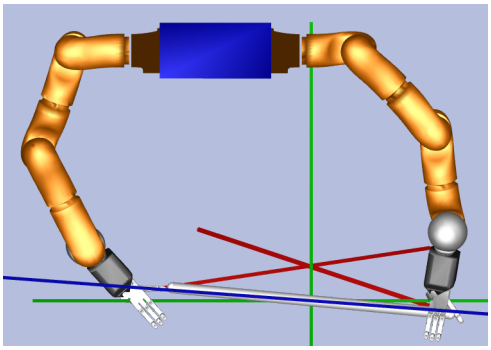
Les résultats de la visualisation donne la figure 4.9 :



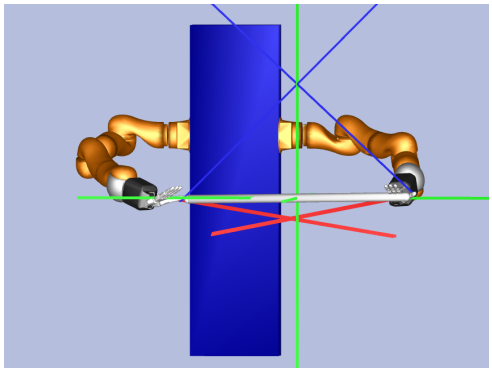
(a) Contact avec le bras gauche (vue de dessus)



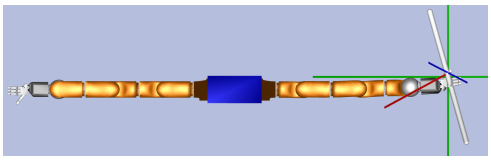
(b) Contact avec le bras gauche (vue arrière)



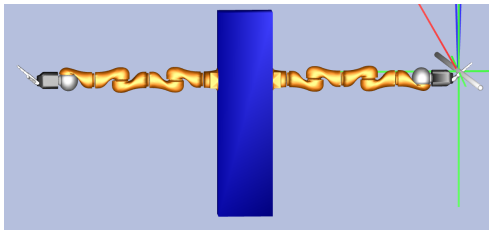
(c) Contact avec les deux bras (vue de dessus)



(d) Contact avec les deux bras (vue arrière)



(e) Contact avec le bras droite (vue de dessus)



(f) Contact avec le bras droite (vue arrière)

FIGURE 4.9 – Une séquence de 3 postures en utilisant 2 contacts C1 et C2 pour faire passer un bâton du bras gauche au bras droit du même plateforme.

4.4.2 Deuxième scénario

Le système S utilisé dans ce scénario est composé de 3 robots, S = {plateforme1, Plateforme2, bâton}. Le but est de faire passer le bâton du bras gauche de Plateforme1 au bras droite de Plateforme2.

La posture initiale σ_{init} contient un contact C1 entre le bras gauche de Plateforme1 et le bâton et nous désirons que la posture finale σ_{final} contiendra un contact entre le bras droite de Plateforme2 avec le bâton. Nous partons de σ_{init}, l'algorithme doit créer un arbre qui contient tout les choix possibles $Adj(\sigma)$ que nous avons en ajoutant ou supprimant un contact. Ensuite, il faut choisir la posture ayant la meilleur métrique et qui n'étant pas visitée avant.

Interprétation du résultat : Plateforme1 commence par un seul contact C1. L'algorithme doit trouver le chemin optimal qui ramène le bâton à C4. En commençant par C1, le robot

a 4 choix, soit d'ajouter le contact C2 ou C3 ou C4 directement ou supprimer C1 (ici on ne peut pas faire une surpression de contact et cela est assuré par la contrainte balance). Dans ce cas, l'algorithme va calculer la métrique qui satisfait le rapprochement de C4 et le confort des couples. Il a trouvé que l'ajout de C2 est le meilleur choix, le bâton se rapproche et la valeur des couples ne dépasse pas le maximum. Ensuite, le robot aura 4 choix aussi, soit ajouter C3 ou C4, soit supprimer C2 ou C1. De même le meilleur choix est de supprimer C1 selon M. Ainsi de suite jusqu'à arriver à C4 de σ_{final} (cf. Figure 4.10).

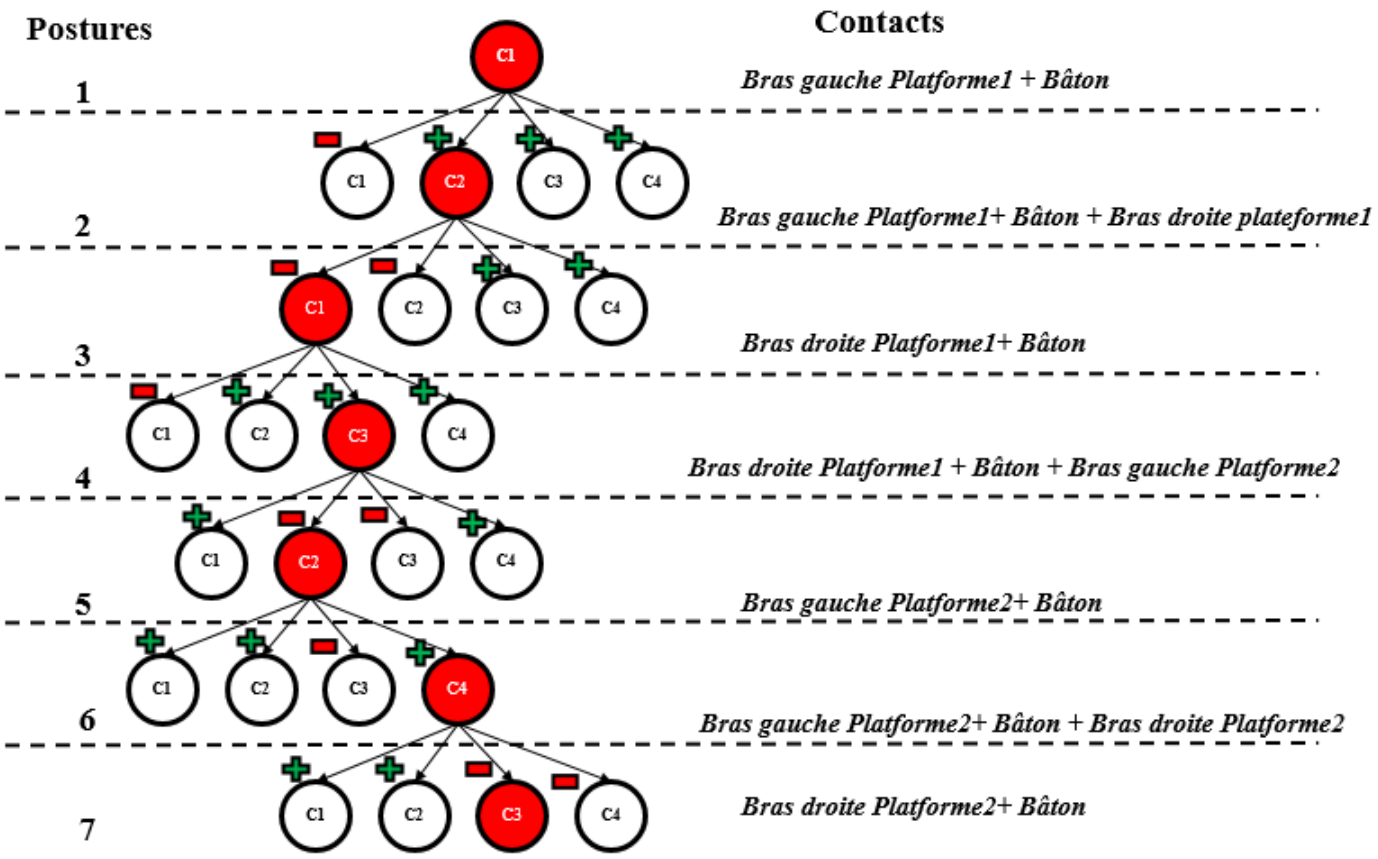
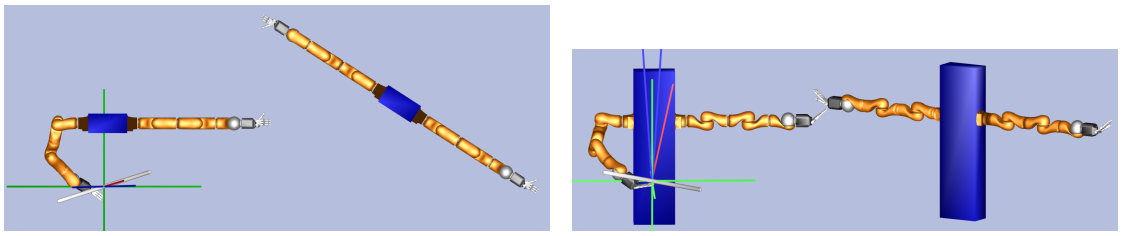
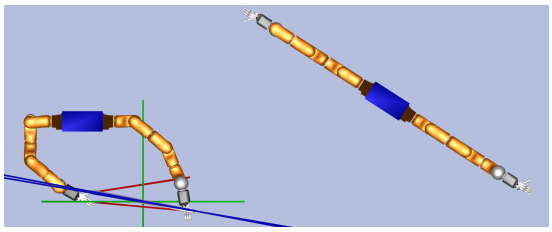


FIGURE 4.10 – Trajet de transfert et de transit pour la séquence de posture.

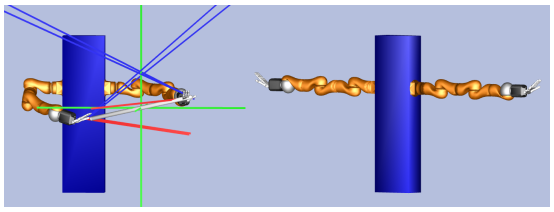
Le résultat de la visualisation est donné par la figure 4.12



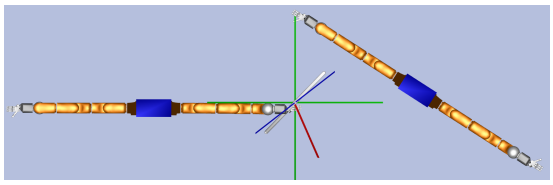
(a) Contact avec le bras gauche de Plateforme1 (b) Contact avec le bras gauche de Plateforme1



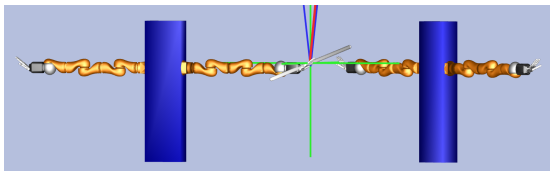
(a) Contact avec le bras gauche et le bras droite de Plateforme1



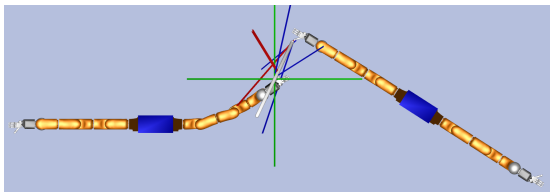
(b) Contact avec le bras gauche et le bras droite de Plateforme1



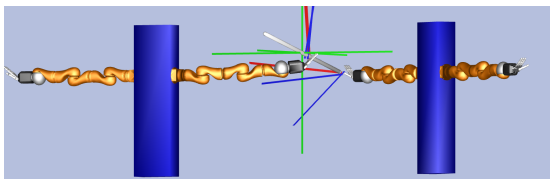
(c) Contact avec le bras droite de Plateforme1



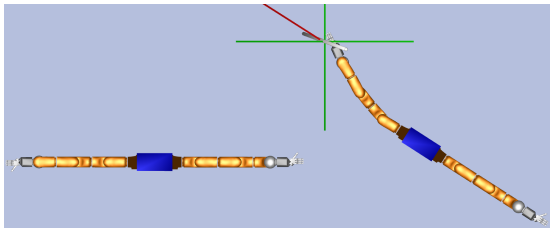
(d) Contact avec le bras droite de Plateforme1



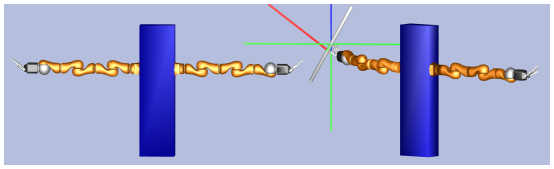
(e) Contact avec bras droite de Plateforme1 et bras gauche de Plateforme2



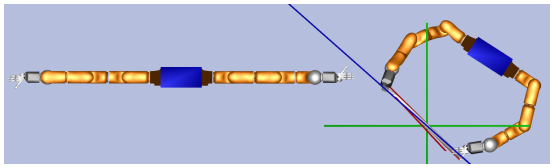
(f) Contact avec bras droite de Plateforme1 et bras gauche de Plateforme2



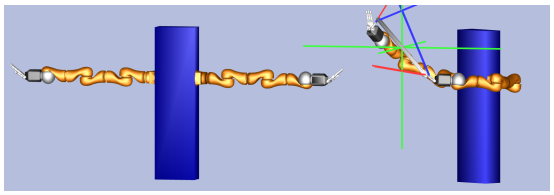
(g) Contact avec bras gauche du Plateforme2



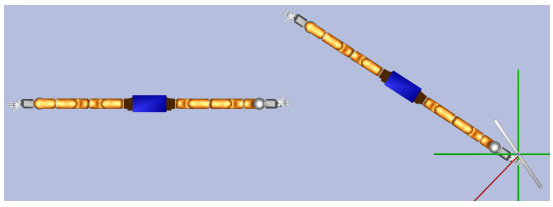
(h) Contact avec bras gauche du Plateforme2



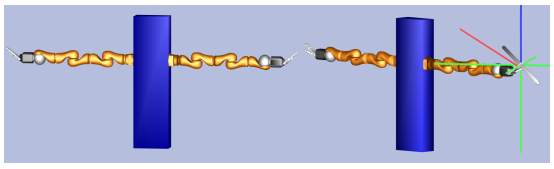
(i) Contact avec bras gauche et bras droite du Plateforme2



(j) Contact avec bras gauche et bras droite du Plateforme2



(k) Contact avec le bras droite du Plateforme2



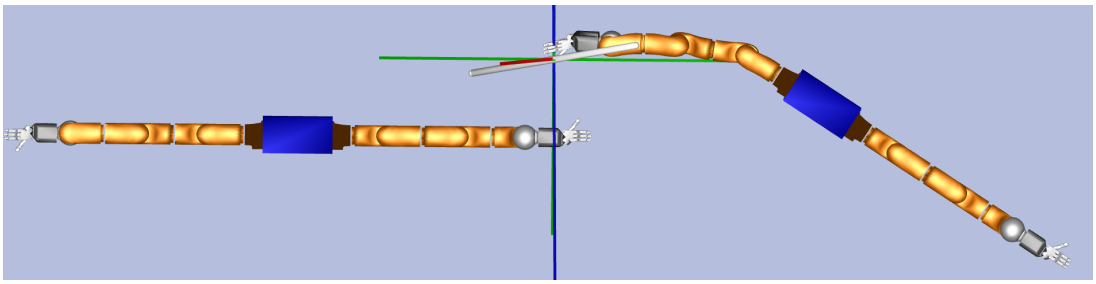
(l) Contact avec le bras droite du Plateforme2

FIGURE 4.12 – Une séquence de 7 postures en utilisant 4 contacts C1, C2, C3 et C4 pour faire passer un bâton du bras gauche du Plateforme1 au bras droit du Plateforme2. Les figures à droite sont en vue de dessus et celles à droite sont en vue d'arrière.

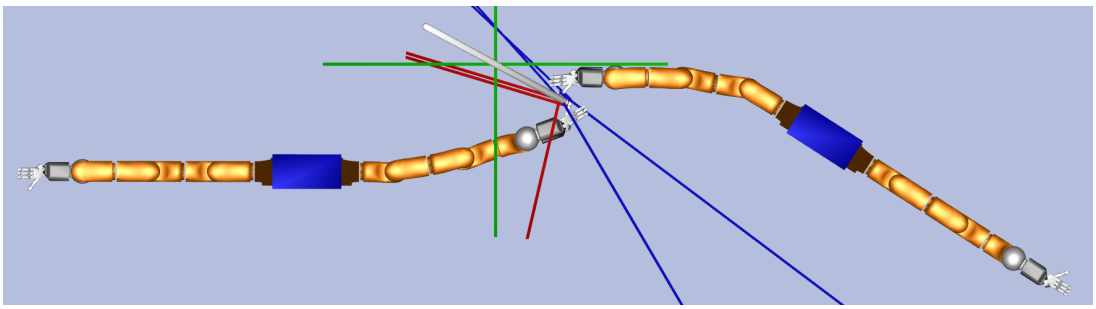
4.4.3 Troisième scénario

Dans ce scénario, nous avons inversé la position de deux robots (Plateforme1 et Plateforme2) et maintenir la même tâche demandée (σ_{init} contient un contact C1 entre le bras gauche de Plateforme1 et le bâton et σ_{final} contiendra un contact entre le bras droite de Plateforme2 avec le bâton) pour vérifier que les robots vont suivre le chemin minimal pour arriver au but.

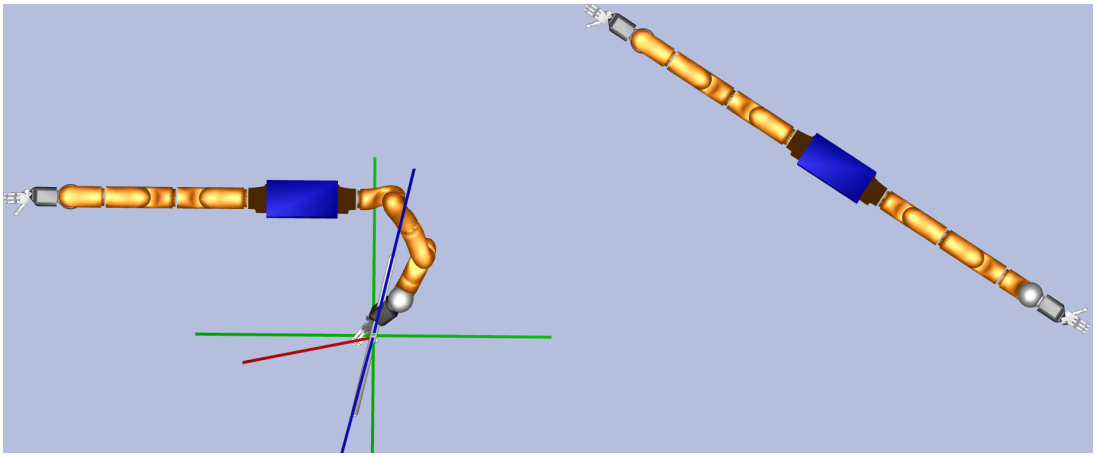
Le résultat est donné par le figure 4.13 :



(a) Contact avec le bras gauche du Plateforme1



(b) Contact avec les deux bras du Plateforme1 et Plateforme2



(c) Contact avec le bras droite du Plateforme2

FIGURE 4.13 – Trajet de transfert et de transit pour la séquence de posture.

Interprétation du résultat : Effectivement, l'algorithme a trouvé que le meilleur chemin est de passer directement de C1 à C4 (cf. Figure 4.14).

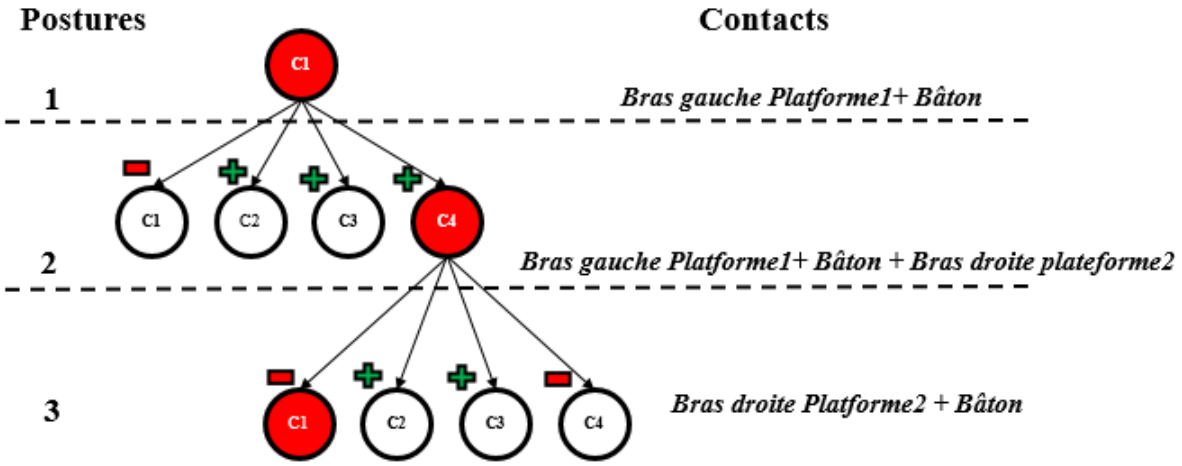
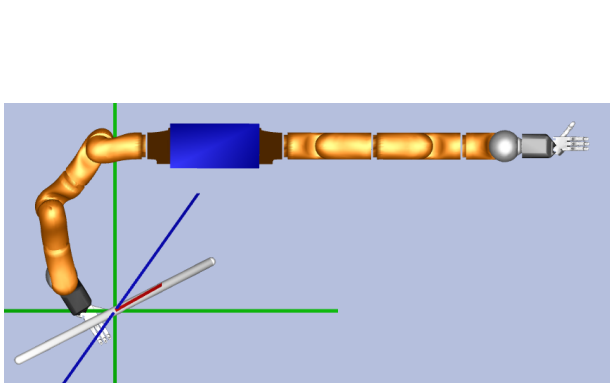


FIGURE 4.14 – Trajet de transfert et de transit pour la séquence de posture de scénario 3.

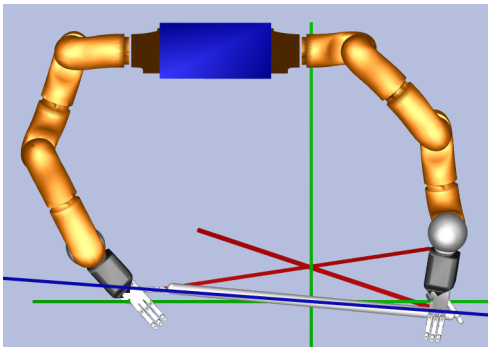
4.5 Évaluation

Dans la section précédente, nous avons trouvé des bon résultats avec l'arbre de recherche proposé. L'avantage de cet arbre est la rapidité puisqu'il n'explore pas la totalité de ces noeuds. En plus, cette méthode donne des résultats même avec des problèmes de grande dimension, c'est à dire avec des tâches composées de plusieurs contacts et de plusieurs robots.

Malheureusement, le manque du temps nous a empêché de continuer ce travail. Nous avons souhaité ajouter la contrainte mémoire à notre projet. Cette contrainte permet de garder la même position (emplacement) de contact d'une posture courante à une posture suivante qui utilise le même contact de la posture précédente. Prenant par exemple le scénario 1 (cf. Figure 4.15), le Plateforme a commencé par un contact C1 avec le bâton et ensuite il a ajouté un deuxième contact C2 ; dans ce cas la contrainte mémoire doit assurer que le contact C1 sera gardé dans la même position par rapport au bâton.



(a) Le contact est au milieu de bâton



(b) Le contact est aux deux extrémités du bâton

FIGURE 4.15 – Changement des positions du contact d'une posture courante à une posture suivante.

La première posture est correcte, c'est logique de créer le contact au milieu du bâton pour assurer l'équilibre de ce dernier. Par contre, en passant à la deuxième posture qui utilise les mêmes contacts de la posture précédente, l'emplacement de contact doit rester le même.

Autrement dit, la deuxième posture qui est composé de deux contacts C1 et C2, normalement C1 doit rester au milieu du bâton. Or ce n'est pas les cas, le contact C1 change sa position du milieu vers l'extrémité du bâton.

4.6 Conclusion

Nous avons écrit un algorithme de planification d'une séquence de posture multiples pour que plusieurs robots aient recours à des contacts pour effectuer des tâches de manipulation. L'autonomie des robots est améliorée pour planifier une séquence acyclique. Cette autonomie est encore accrue en ne spécifiant pas les emplacements de contact candidats pré-discrétisés sur l'environnement, dont la continuité est totalement explorée par le planificateur. Avec l'autonomie, l'autre concept de conduite clé dans ce chapitre était la généralité. Notre planificateur n'a pas été ciblé pour un modèle spécifique de robot ou de système de robots. Le planificateur a réussi à faire face à un ensemble de problèmes pris à partir de différents scénarios.

Chapitre 5

Conclusion générale

Sommaire

5.1 Conclusions

Dans ce travail de master, nous avons implémenté un algorithme de planification d'appuis pour des robots manipulateurs en leur fournissant un outil qui leur permet de planifier les mouvements de manipulation en changeant leurs configurations de contact.

Nous avons pour cela étudié la structure de l'espace de configuration engendré par la notion de contact et mis en évidence la nécessité de développer des outils pour se contraindre à être sur les sous-variétés qui le composent.
Nous avons recodé un générateur de posture qui remplit ce rôle en transformant des données de haut niveau, comme la position de contacts, en un problème d'optimisation. Chaque posture générée par ce générateur délivre une configuration du système de robots sous équilibre statique, dans les limites de ses articulations et de couple, en évitant une collision et en atteignant un ensemble spécifié de contacts qui sont non horizontaux, non-coplanaires et, en tant que tels, assez généraux pour considérer la variété souhaitée de problèmes de planification de robot.

Grâce à ces outils, nous avons implémenté un algorithme de planification d'appuis. Il se base sur deux niveaux principaux. L'un, à la base, est le générateur de posture. Le second construit de manière itérative, en étant guidé par une métrique, un arbre dont les nœuds sont des ensembles de contacts. L'arbre croit vers le but, et la solution est une séquence d'ensembles de contacts, chacun différant du précédent par un contact, et une séquence de postures associées, séquences telles que l'on peut trouver un chemin quasi-statique entre les différentes postures qui respecte l'un après l'autre les ensembles de contacts.

Enfin, grâce à ce stage, j'ai bien appris la programmation orienté objet en c++, j'ai appris les notions de classe tel que l'héritage, le polymorphisme (classe virtuelle, classe virtuelle pure ...), etc. Au début de stage, je n'avais pas ces notions, même je n'avais pas les techniques de compilation et d'exécution sur le terminal de ubuntu. J'ai passé une période pour apprendre ces techniques et ces notions.

5.2 Perspectives

Malheureusement, le manque du temps nous a empêché de continuer ce travail. Par conséquent, nous proposons d'étendre ce travail pour ajouter une contrainte mémoire et essentiellement lui adapter à des environnement plus généraux comportant des objets modélisés de manière peu précise et/ou déformables. Cela sera effectué par la modification du contrainte contact dans le chapitre 3. Il faut étudier le comportement physique d'un contact avec un objet déformable ou/et de poids mal identifié et étudier les forces appliquées sur l'objet. En plus, nous proposons d'adapter ce travail à un robot à base mobile.

Bibliographie

[1] R. Alami, J. Laumond, and T. Siméon, Two Manipulation Planning Algorithms, Proceedings of the Workshop on Algorithmic Foundations of Robotics, 1995, pp. 109–125.

[2] Gustavo Arechavaleta-Servin. An optimality principle governing human walking. Phd thesis, Institut National des Sciences appliquées, Toulouse, 112p., 2007. Doctoract.

[3] Y. Ayaz, K. Munawar, M. Bilal Malik, A. Konno M. Uchiyama. Human-Like Approach to Footstep Planning Among Obstacles for Humanoid Robots. In Intelligent Robots and systemes, 2006 IEEE/RSJ International Conference on, pages 5490-5495, Beijing, October 2006.

[4] T. Bretl. – Multi-Step Motion Planning : Application to Free-Climbing Robots. – PhD. Thesis, Stanford University, 2005.

[5] K. Bouyarmane, A. Escande, F. Lamiraux, and A. Kheddar, Potential Field Guide for Multicontact Humanoid Motion Planning, Proceedings of the IEEE International Conference on Robotics and Automation, 2009.

[6] J. Chestnutt, J. Kuffner, K. Nishiwaki, S. Kagami. –Planning biped navigation strategies in complex environments.– 2003.

[7] O. Cohen, S. Druon, S. Lengagne, A.Mendelsohn, R. Malach, A. Kheddar, D. Friedman, -fMRI based robotic embodiment : a pilot study, IEEE/RAS-EMBS Int. Conf. on Biomedical Robotics and Biomechatronics (BioRob), 2012

[8] M. G. Choi, J. Lee, S. Y. Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Transactions on Graphics, 22(2) :182–203, 2003.

[9] John J. Craig, Introduction to Robotics : Mechanics and Control (3rd Edition), Prentice Hall, 2004.

[10] J. Laumond, Robot Motion Planning and Control, Springer, 1998.

[11] S. Kajita and B. Espiau, Springer Handbook of Robotics, ch. Legged Robots, Springer, 2008.

[12] S. Lengagne and J. Vaillant and E. Yoshida, -Generation of Whole-body Optimal Dynamic Multi-Contact Motions, The International Journal of Robotics Research, 17 pages, 2013

[13] Steven M. Lavalle. Planning algorithms. Cambridge University Press, 2006

[14] B.Mettler. An extremal fields approach for the analysis of human planning and control performance. In Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pages 2152-2158, Pasadena, CA, May 2008.

[15] J. Latombe. – Robot motion planning. – Kluwer Academic Publishers, Boston-Dordrecht-London, 1991.

[16] Steven M. LaValle. – Rapidly-exploring random trees : A new tool for path planning. – Rapport de recherche, 1998.

[17] L. E. Kavraki, P. Svetska, J. C. Latombe, M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation, 12(4) :566–580, June 1996.

[18] S. Miossec, K. Yokoi, A. Kheddar. – Development of a software for motion optimization of robots - application to the kick motion of the hrp2 robot. – 2006.

[19] S. Miossec, S.Lengagne K.Yoki A.kheddar and. Motion optimization of robotic systems and validation of HRP-2 robot. In RSJ National Conference, 2008.

[20] W. Suleiman, E. Yoshida., J-P. Laumond. A. A. Monin. Optimizing Humanoid Motions Using Recursive Dynamics and Lie Groups. In information and Communication Technologies : From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on, pages 1-6, Damascus, April 2008.

[21] J. J. Jr Kuffner, K. Nishiwaki., S. Kagami., M. Inaba H. Inoue. Footstep planning among obstacles for biped robots. In Intelligent Robots and Systems, 2001.

[22] A. Escande, Planification de Points d'Appui pour la Génération de Mouvements Acycliques : Application aux Humanoides, Ph.D. thesis, Université d'Evry-Val d'Essone, December 2008.

[23] S. Lengagne, Planification et re-planification de mouvements sûrs pour les robots humanoïdes, Ph.D. thesis, Université Montpellier 2, Octobre 2009.

[24] K. Bouyarmane, On Autonomous Humanoid Robots : Contact Planning for Locomotion and Manipulation, Ph.D. thesis, Université Montpellier 2, November 2011.

[25] K. Hauser, T. Bretl, J. Latombe. – Non-gaited humanoid locomotion planning. – IEEE/RSJ International Conference on Humanoid Robots, pp. 7–12, December 5-7 2005.

[26] K. Hauser, T. Bretl, K. Harada, J. Latombe. –Using motion primitives in probabilistic sample-based planning for humanoid robots. – Workshop on the Algorithmic Foundations of Robotics, 2006.

[27] A. Escande, A. Kheddar, and S. Miossec, Planning Support Contact-Points for Humanoid Robots and Experiments on HRP-2, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006.

[28] J. Cortes, T. Siméon, and J. Laumond, A Random Loop Generator for Planning the Motions of Closed Kinematic Chains Using PRM Methods, Proceedings of the IEEE International Conference on Robotics and Automation, 2002.

[29] T. Siméon, J. Cortès, J. Laumond, A. Sahbani. -Manipulation planning with probabilistic roadmaps. The International Journal of Robotics Research, 23(7-8) :729–746, July-August 2004.

[30] M. Stilman. – Task constrained motion planning in robot joint space. –IEEE/RSJ International Conference on Robots and Intelligent Systems, 2007.

[31] T.-Y. Li. – Motion planning for humanoid walking in a layered environment. – IEEE International Conference on Robotics and Automation, pp. 3421–3427, 2003.

[32] U. Muico, Y. Lee, J. Popovic, Z. Popovic, Contact-aware Nonlinear Control of Dynamic Characters, ACM Transactions on Graphics (SIG-GRAPH) 28

[33] F. Kanehiro, T. Yoshimi, S. Kajita, M. Morisawa, K. Fujiwara, K. Harada, K. Kaneko, H. Hirukawa, F. Tomita, Whole Body Locomotion Planning of Humanoid Robots based on a

3D Grid Map, in : IEEE International Conference on Robotics and Automation, 1072–1078, 2005.

[34] A. Escande, A. Kheddar, S. Miosse, Planning contact points for humanoid robots, Preprint submitted to Robotic and Autonomous Systems, January 21, 2013

[35] Z. Qiu, A. Escande, A. Micaelli, and T. Robert, Human motions analysis and simulation based on a general criterion of stability, 2011

[36] Introduction to Ipopt : A tutorial for downloading, installing, and using Ipopt. Revision number of this document : Revision : 2020 June 30, 2011

[37] S. Kagami, K. Nichiwaki, T.Kitagawa, T.Sugihara, M.Inaba and H.Inoue, "A fast generation method of a dynamically stable humanoid robot trajectory with enhancedzmp constraint", in Proc. 1st IEEE-RAS Int. Conf. on Humanoid Robots, September 7-8 2000.

[38] A. Goswami and V. Kallem, "Rate of change of angular momentum and balance maintenance of biped robots, IEEE Int. Conf. on Robotics and Automation, New Orleans, LA, April 2004.

[39] D. Mansour, A. Micaelli and P. Lemerle, "A Computational Approach for Push Recovery in case of Multiple Noncoplanar Contacts", in Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, September 25-30 2011, San Francisco, California.

[40] K.Harada, S. Kajita, K. Kaneko H. Hirukawa. ZMP analysis for arm/leg coordination. Tin Intelligent Robots and Sytems, 2003. (IROS 2003). Procreedings. 2003 IEEE/RSJ International Conference on, volume 1, pages 75-81, Octobre 2003.