Planification de points d'appui pour la génération de mouvements acycliques : application aux humanoïdes

Adrien Escande

THESE

présentée à

L'UNIVERSITE D'EVRY–VAL D'ESSONNE

par

Adrien ESCANDE

pour obtenir le diplôme de

Doctorat de l'Université d'Évry - Val d'Essonne

Spécialité : Robotique

Planification de points d'appui pour la génération de mouvements acycliques : application aux humanoïdes

---

Soutenue le 12 Décembre 2008

JURY

M. Claude Laurgeau	Professeur aux Mines de Paris	Président
M. Étienne Dombre	Directeur de Recherche au CNRS	Rapporteur
M. Bernard Espiau	Directeur de Recherche à l'INRIA	Rapporteur
M. Jean-Paul Laumond	Directeur de Recherche au CNRS	Rapporteur
M. Claude Andriot	Chercheur au CEA	Examinateur
M. Sylvain Miossec	Maître de conférences, Université d'Orléans	Examinateur
M. Eiichi Yoshida	Senior Researcher, AIST, Japon	Examinateur
M. Abderrahmane Kheddar	Directeur de Recherche au CNRS	Directeur de thèse

Remerciements

L'histoire a commencé dans un café face à la station Denfert-Rochereau, un matin de Décembre. J'y rencontrais le co-directeur d'un laboratoire franco-japonais de robotique humanoïde au Japon, en vu d'un stage sur la planification d'appuis. Très vite la question est venue :
"Etes vous intéressé par une thèse ? "
Et moi de répondre non, sans aucune hésitation.

Quatre ans plus tard, presque jour pour jour, et quelques centaines de mètres plus loin, je soutenais ma thèse sur le même sujet. Et durant les presque trois ans qu'elle aura durée, de l'autre coté de l'hémisphère, jamais je n'aurai regretté d'avoir changé d'avis. Il y a bien sûr eu quelques moments de doute, de manque de motivation ou de fatigue mais ils se noient dans cette incroyable expérience, parmi les longues heures captivantes à chercher, l'excitation à creuser une nouvelle idée séduisante, pour ensuite bien souvent mettre le doigt sur le *petit* détail qui l'invalide, les instants de satisfaction quand enfin un résultat tombe, et puis tous ces échanges enrichissants, passionnants, professionnels ou amicaux, avec les personnes qu'il m'a été donné de rencontrer aussi bien au cours de mes recherches que dans le semi-dépaysement de la vie japonaise.

Et ils sont nombreux ceux que j'ai côtoyés au cours de cette thèse et qui ont participé, chacun à leur manière, à en faire une réussite, et trois ans mémorables. Ils sont nombreux donc ceux à qui je dois des remerciements, trop nombreux pour que je puisse écrire ici tout ce que je leur dois sans risquer de doubler la taille de ce manuscrit. J'espère néanmoins n'oublier personne dans ces remerciements concentrés.

Tout d'abord un merci inconditionnellement reconnaissant à Abderrahmane Kheddar, mon directeur de thèse, pour m'avoir proposé ce sujet et poussé à rester après mon stage, pour avoir tout mis en œuvre, avec succès, afin de m'assurer les meilleures conditions de travail possibles, pour m'avoir incité à toujours me dépasser, et m'avoir encadré et conseillé sans compter ses heures. Je mesure très bien la chance que j'ai eue.

Un grand merci aux membres de mon jury pour leur temps et leurs commentaires, en particulier aux trois rapporteurs Etienne Dombre, Bernard Espiau et Jean-Paul Laumond, pour la lecture attentive de ce manuscrit et les retours qu'ils m'ont fait dessus. Je dois aussi des remerciements supplémentaires à Jean-Paul pour les discussions que nous avons pu avoir lors de nos rencontres aux cours de ces trois ans, à l'Ecole des Mines, à Tsukuba ou lors de conférences. Un merci particulier à Eiichi Yoshida pour qui le français n'est pas la langue

maternelle, et à Claude Laurgeau pour sa présidence de jury remarquable et remarquée, et pour la gentillesse dont il a fait part depuis mes premiers pas dans la robotique, en cycle ingénieur, jusqu'à cette soutenance.

Un immense merci à Sylvain Miossec pour toutes les discussions passionnantes, pointues ou pointilleuses, enflammées, pugnaces parfois, que nous avons pu avoir, discussions improvisées de cinq minutes qui débutaient la journée pour finalement s'interrompre au déjeuner, ou réunions de travail. Merci pour ses conseils et son suivi de mes travaux, pour la connaissance qu'il m'a apporté en optimisation. Merci aussi pour les tores des STP-BVs.

Cette thèse est profondément attaché au JRL-Japon et à ceux qui le composent ou en ont fait partie au cours des années. Ses membres ont été des collègues, mais sont pour beaucoup aussi des amis, amitiés renforcées par le contexte si particulier de l'expatriation. Les journées avec eux se terminaient souvent bien après la sortie du laboratoire. En cela, il n'est pas toujours possible de séparer les contributions professionnelles et personnelles de chacun, et il serait bien vain de chercher une quelconque signification à l'ordre des remerciements.
Merci à Jean-Rémy, voisin au bureau comme à la ville, pour son aide précieuse face à tous les problèmes qui peuvent surgir dans un pays dont on ne maîtrise que très peu la langue et les coutumes. Interagir avec l'administration de l'AIST, réserver un ryokan, s'abonner à internet, trouver un appartement ou acheter un piano..., il y en avait bien assez pour que j'use et abuse aussi de l'aide de mes deux élèves de français, Tokiko Uga, notre charmante secrétaire, si efficace et toujours souriante, et Hitoshi Arisumi, le plus italien des japonais. A eux aussi un grand merci, ainsi que mes félicitations pour leurs efforts à apprendre ma langue.
Merci à Olivier Stasse pour son aide sur mille et un sujets, des questions techniques aux plus théoriques en passant par les points les plus pratiques, en programmation comme sur le robot. Par son travail au jour le jour, il apporte tellement à ce labo.
Merci à Pierre-Brice Wieber pour ses conseils et enseignements en optimisation, l'idée de la génération de code à partir de HuMAnS et surtout le concept original de la génération de posture. Je regrette qu'il n'ait pas pu revenir plus tôt au JRL.
Merci à Nicolas Mansard pour tant de choses, des discussions et toujours bons conseils scientifiques aux heures de sport éreintantes, des petits-déjeuners - coupe du monde de rugby à la répétition de ma soutenance de thèse.
Le JRL est un lieu de passage intense. Stagiaires, thésards ou post-docs, ce renouvellement continu contribue à la fraicheur du laboratoire et sa vitalité scientifique et humaine. Parmi ceux qui viennent, certains s'y arrêtent plus longtemps que d'autres et je leur dois des remerciements pour le temps passé ensemble, les échanges et les bons moments. Il en est ainsi de Paul, maître du C++ et du karaoke, qu'on ne saurait mentionner sans sa charmante Amélie (ou est-ce le contraire ?), François, Anthony, Toréa, Nosan. Il me faut aussi remercier les personnes que j'ai co-encadrées sur des sujets touchants de plus ou moins près à ma thèse, et qui m'ont chacun beaucoup apporté : Florian, Kevin, Sovan, Amélie encore une fois, Sylvain, Karim et Mehdi.
Thank you to Neo Ee Sian, and to HRG and GRX members, for their help and advices concerning the experiments on HRP-2. I took a lot of interest discussing with Shuuji Kajita, Kenji Kaneko and Hajime Saito. Lastly, JRL wouldn't be what it is without the work of its japanese co-head, Kazuhito Yokoi. With Abder, he built a laboratory where working is

always pleasant. Thank you to him.

Même au plus profond de la thèse il existe un monde à l'extérieur du laboratoire, peuplé de gens à qui je dois beaucoup.
Thomas en tout premier lieu, ami fidèle qui m'a accompagné dans mes premiers pas au Japon et grâce à qui partir dans ce lointain pays ne semblait pas si insurmontable. Merci pour tous ces week-ends tokyoïtes et ces visites du Japon.
I owe also many thanks for the people of Tsukubafriends or affiliated, for all the parties and activities we did : Cécile, Milica, Atsuko, Sandrine, Cédric et Yumi, Yoshi, Stéphane, Yuko, Reiko, Rieko, Nicolas et Christine, Hisako, Tomoko.
Eri-chan, arigatô.
I don't forget either my two volley-ball teams for all the good times spent during trainings, tournaments, restaurants or karaoke. Special thanks for Mizutani-san and Aoki-san for the organization.

Enfin, mes plus grands remerciements à ma famille et mes amis de longues dates en France qui m'ont soutenu en dépit de la distance et accueilli à bras ouverts à chacun de mes retours. Je les ai toujours retrouvés avec beaucoup de plaisir.
Merci à Guillaume, Audrey, Bérénice, Christophe, Kristèle, Ronan. Merci à Nicolas, Rémi, Pierre et Eliette. Merci à mes grands-parents. Et un immense merci à mes parents pour la relecture de ce manuscrit et les vingt-six ans, huit mois et six jours moins quelques heures qui ont conduit à ma soutenance.

Table des matières

Introduction

Tokyo, Salle de Concert du Metropolitan Art Space. Devant deux mille personnes, un robot à l'éclat argenté plaque les derniers accords de la Symphonie pour Orgue n°5 de Charles M. Widor. L'exécution a été magistrale. Les derniers échos de la musique sont vite remplacés par les applaudissements des spectateurs. Le robot se lève et se tourne vers son public pour le saluer. Derrière lui, son assistant replie la partition, puis, avant de s'éloigner, presse le bouton "historic". Sans bruit, les trois parties de l'orgue pivotent alors. Le buffet blanc de l'orgue classique français disparaît tandis que se dessinent graduellement les contours plus stricts d'un orgue baroque allemand. Devant, l'exécutant salue une dernière fois puis quitte la salle. La première partie est terminée.

Cette scène, pour aussi futuriste qu'elle puisse paraître, est quasiment réalisable, et le seul élément qui s'y oppose ne tient ni à la présence d'un robot organiste, ni à celle d'un double orgue pivotant - deux prouesses techniques qui ont vu le jour dans les années 80 - mais à un verbe : le robot *se lève*.
Cette action, si machinale pour un être humain, est loin d'être aussi simple qu'elle n'y paraît. Revenons en arrière et regardons de plus près : notre robot est assis sur un banc un peu moins large que la console de l'orgue (voir Fig 1.1). Il ne peut pas véritablement poser ses pieds car il lui faut éviter de s'appuyer sur le pédalier. Seul l'avant de celui-ci lui laisse une petite corniche, largement occupée par des boutons et pédales. Les genoux sont sous les claviers.

Pour s'extraire de la console, le robot écarte d'abord la main droite de son corps, puis appuie celle-ci, ainsi que la gauche, sur le banc. Il se soulève alors, légèrement groupé pour garder son équilibre, le tronc un peu penché en avant, les pieds sous le banc, et déplace son corps vers le côté. Il fait attention à ne pas se cogner les genoux contre la console, ni à heurter une pédale du pied. En équilibre sur son postérieur, il ramène alors sa main gauche à lui et recommence le mouvement total une seconde pour arriver au bord du banc. Une main sur la console, l'autre sur le banc, il se maintient alors tout en soulevant une "cuisse" pour poser un premier pied à terre. L'autre pied suit et le voilà debout. Fin du ralenti.

Derrière *se lever*, il y a donc pour le robot toute une séquence d'appuis et de mouvements visant à atteindre un but (être debout à côté du banc), tout en maintenant à tout instant l'équilibre et en évitant d'entrer en collision avec les objets alentours. Trouver une telle séquence, c'est-à-dire *planifier des appuis*, dans le cas d'un problème générique est le but de cette thèse.
La motivation de ce travail est de fournir un planificateur de mouvements qui puisse fonctionner pour des environnements délicats du fait de leur géométrie (encombrés ou restreints,

FIG. *1.1 – Les deux faces de l'orgue du Metropolitan Art Space de Tokyo (classique/symphonique Français à gauche, Renaissance Hollandaise/baroque Allemand à droite), et une console d'orgue typique (milieu).*

avec un sol très irrégulier...) où la nécessité d'une démarche fortement acyclique ne permet pas l'utilisation de planificateurs tournés vers des mouvements spécifiques. Comme le montre l'exemple, il devra prendre en compte les notions de postures et d'appuis en réponse aux contraintes géométriques et physiques telles que l'évitement de collision, le maintien de l'équilibre, etc.

Cette thèse prend pour champ d'application les humanoïdes, avatar de synthèse ou robot, et plus particulièrement le robot HRP-2 sur lequel nous expérimentons les résultats de ce travail (entre autres sur un scénario similaire à l'exemple cité mais dans l'environnent bien plus sobre qu'offre un laboratoire de recherche). Elle s'étend cependant à tout système poly-articulé se déplaçant dans un espace physique à deux ou trois dimensions. Elle se propose d'étendre les capacités motrices de ces systèmes afin de les exploiter réellement dans les environnements pour lesquelles ils ont été prévus.

Spécificité de la planification d'appuis

La planification de points d'appui est une classe spécifique de la *planification de mouvement* dont le problème emblématique est celui dit du "déménageur de piano" : comment amener un objet d'une position à une autre de l'espace, sans qu'il n'y ait de collision avec des obstacles. L'objet peut être un simple solide comme un corps poly-articulé, dans un espace à deux ou trois dimensions. On peut considérer plus généralement une combinaison de tels solides ou corps. On pourra utiliser de manière générique le terme de *robot* pour désigner le système dont on souhaite trouver les mouvements. Les différentes déclinaisons de ce problème tiennent principalement aux contraintes que doivent respecter les mouvements du robot. Par exemple, une voiture ne peut pas se déplacer latéralement, un objet inerte ne peut bouger que lorsqu'il est manipulé, etc.

De nombreuses notions ont été développées pour l'étude d'un tel problème, et en particulier celle d'*espace de configuration*. Tout point de cet espace (variété), ou configuration, spécifie complètement un état du robot, et tout état correspond à un point de cet espace. Par exemple, une boîte bougeant sur un plan peut être représentée par un triplet (x, y, θ) correspondant à sa position sur le plan et son orientation. Dans l'espace ce serait six paramètres $(x, y, z, \psi, \theta, \varphi)$. Deux boîtes dans le plan auraient également six paramètres $(x_1, y_1, \theta_1, x_2, y_2, \theta_2)$. Ces paramètres sont les degrés de liberté du "robot".

Dans cet espace, on peut représenter la plupart des concepts utiles à la planification. Ainsi un obstacle y sera représenté par le volume qui correspond à tous les points représentant des états où le robot est en collision avec cet obstacle, la stabilité pourra être décrite comme un sous-ensemble de points, et une contrainte est un sous-ensemble ou une sous-variété de cet espace.

Le problème de planification d'appuis a deux spécificités. Tout d'abord, on y considère un robot qui est sous-actionné, c'est-à-dire qu'il a moins d'actionneurs que de degrés de liberté. Il ne peut alors pas se déplacer comme il veut dans l'espace. En particulier, il ne peut par exemple pas voler. Il est restreint à bouger en fonction des interactions qu'il a avec son environnement : ses appuis.
En second lieu, elle appelle à repenser la notion d'obstacle. En effet, comme illustré dans l'exemple, un objet dans l'environnement a le double rôle de soutien potentiel et d'obstacle. Ainsi en est-il de la console, sur laquelle le robot s'appuie d'une main, mais qu'il évite en même temps avec ses jambes.

Ces deux notions se traduisent dans l'espace de configuration par le fait que le robot est contraint à bouger sur la frontière des obstacles ce qui revient à se trouver sur des sous-variétés de l'espace, chacune correspondant à un ensemble d'appuis. A la succession d'appuis recherchée va correspondre une succession de sous-variétés. Les mouvements quant à eux seront des chemins sur ces sous-variétés.

Ces contraintes sur le système amènent des complications notables, en particulier elles transforment le problème de planification général, qui est une recherche dans un espace continu, en un problème hybride qui mêle discret et continu : le choix des types d'appuis (et donc des sous-variétés sur lesquelles évoluer) est en effet discret, le choix de la position des appuis est par contre continu, tout comme l'est la recherche de la trajectoire entre deux ensembles d'appuis. Ce triple niveau de choix est particulièrement difficile à cause de la taille de l'ensemble des choix discrets, qui explose de manière combinatoire. Ces choix sont de plus fortement dépendants les uns des autres.

Contribution et Plan

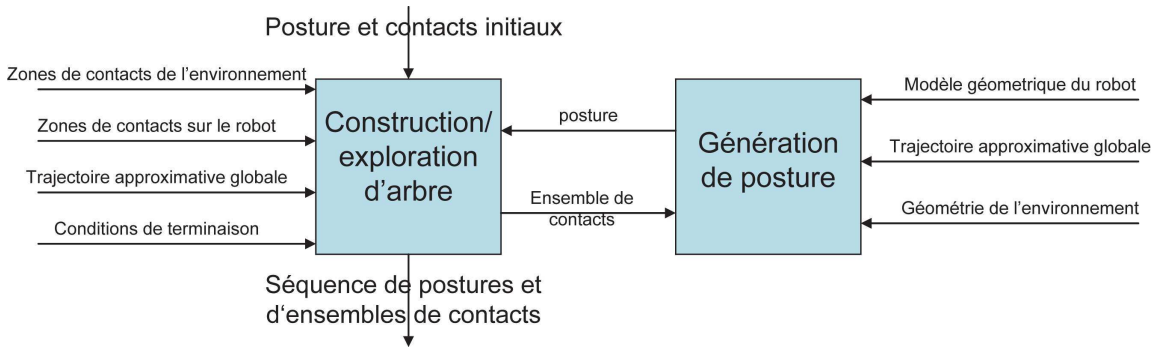


FIG. *1.2 – Structure du planificateur.*

Cette thèse expose la construction d'un planificateur d'appuis (voif figure 1.2). Elle se base sur les composants de celui-ci et s'organise comme suit :
le chapitre 2 illustre le problème de planification de points d'appuis par un exemple qui met

en avant certains points caractéristiques, avant d'introduire plus formellement le problème et les notations et de s'intéresser aux travaux connexes. Viennent ensuite les trois principales contributions de ce travail, expliquées de manière détaillée :

- la description et l'implémentation d'un **générateur de posture** (chapitre 3) qui permet de trouver une configuration du robot respectant une série de contraintes comme l'appartenance à des sous-variétés correspondant aux appuis, les non-collisions ou le maintien de l'équilibre (que nous désignons aussi par le terme *stabilité* du maintien, à ne pas confondre donc avec la notion de stabilité des systèmes, terme plus attitré à l'automatique). Ce générateur est un outil de base pour la suite du travail. Il est aussi très générique et nous montrons rapidement quelques autres exemples de son utilisation,
- l'introduction d'un nouveau volume englobant, le **STP-BV** (Sphere Torus Patch Bounding Volume, chapitre 4) qui permet la régularisation de la fonction distance entre deux objets, ainsi qu'une méthode pour calculer cette distance. Ce travail permet l'écriture des contraintes de non-collision dans le générateur de posture, ainsi que la prise en compte des distances dans des schémas de contrôle en assurant la continuité de la commande. Ce chapitre peut être ignoré en première lecture pour qui ne s'intéresse qu'à la planification même,
- **un algorithme de planification d'appuis**, et les différents modules qui le composent (chapitres 5 et 6). Cet algorithme s'attaque principalement aux choix discrets et continus liés à la prise d'appuis en manipulant des ensembles d'appuis. Il se base sur des heuristiques pour pouvoir être quasiment sûr de trouver une trajectoire continue entre de tels ensembles, mais ne donne pas cette trajectoire.

Enfin, le chapitre 7 montre les résultats de planifications pour plusieurs scénarios, et le passage des plans obtenus vers une implémentation et des expérimentations réelles avec le robot humanoïde HRP-2. Nous discutons par ailleurs des problèmes à la fois techniques et fondamentaux qui restent encore à surmonter dans le cadre d'un travail futur.

Chapitre **2**

Définition du problème

Dans cette thèse, nous cherchons à résoudre le problème suivant :
étant donné

- un système polyarticulé (qu'on appellera robot et qui sera typiquement un robot humanoïde),
- des emplacements de contact possibles sur la surface des corps du robot,
- un environnement avec des surfaces sur lesquelles le robot peut prendre appui,
- une configuration initiale en contact, et une condition de fin sous forme d'un but à atteindre,

trouver une séquence d'ensembles d'appuis (s_i) commençant à la configuration initiale et dont le dernier élément vérifie la condition de fin, et un chemin sans collision et quasi-statique suivant la séquence des appuis.

Nous commençons par donner un exemple simple qui illustre la structure et les difficultés de ce problème (§ 2.1), puis nous définissons les notations utilisées dans le reste de ce travail (§ 2.2) et nous ferons un état de l'art des travaux connexes (§ 2.3).

2.1 Un exemple académique

Considérons un robot dans le plan constitué de deux segments $[OP_1]$ et $[OP_2]$, de longueur respective l_1 et l_2 avec $l_1 > l_2$, tel qu'illustré à la figure 2.1. La position de tout point de ce robot est parfaitement spécifiée à l'aide des quatre paramètres $(x, y, \theta_1, \theta_2)$, où (x, y) est la position du point O et θ_1 et θ_2 sont les angles que font $[OP_1]$ et $[OP_2]$ avec la verticale. L'espace de configuration $\mathcal{C}$ de ce robot est donc une variété à quatre dimensions qui s'identifie à $\mathbb{R}^2 \times SO(1)^2$. Une trajectoire du robot correspond à une courbe dans cet espace.
Étant donné une surface, ce robot peut former un contact avec elle en ses points terminaux P_1 et P_2. Il reste libre alors de tourner autour de ces points. On cherche à étudier les mouvements de ce robot lorsqu'il est restreint à toujours avoir au moins un contact avec son environnement, sans glisser, et on se place de plus dans le cas où ces contacts ne peuvent avoir lieu que sur la droite $y = 0$ (le sol), qui découpe le plan entre un espace libre $(y > 0)$

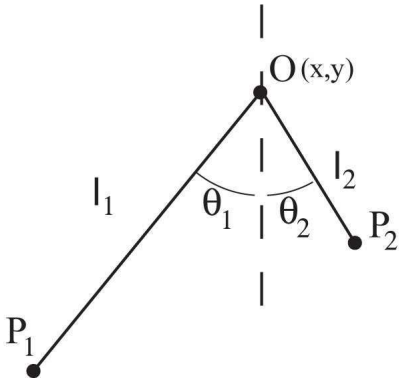


FIG. *2.1 – Un robot à deux corps*

et un obstacle ($y \leq 0$). On verra alors comment la restriction des mouvements influe sur le problème de planification "aller d'une configuration $(x, y, \theta_1, \theta_2)$ (en contact) à une configuration $(x', y', \theta'_1, \theta'_2)$ (en contact)".

2.1.1 Représentation dans $\mathbb{R}^3$

La contrainte "P_i $(i = 1, 2)$ est en contact avec le plan $y = 0$" (sans spécifier où), est une contrainte à une dimension

$$y = l_i \cos(\theta_i) \tag{2.1}$$

Elle définit une sous-variété de dimension trois, $\mathcal{C}^i$.
De même l'ensemble des configurations où P_1 et P_2 sont en contact est une sous-variété de dimension 2, $\mathcal{T} = \mathcal{C}^1 \cap \mathcal{C}^2$. $\mathcal{T}$ est l'espace des configurations de transition entre $\mathcal{C}^1$ et $\mathcal{C}^2$. La forme de ces espaces est constante selon l'axe de x. Elle est montrée par la figure 2.2 pour un hyperplan $x = x_0$.
Un déplacement du robot sur un de ces espaces correspond à un glissement du point (ou des points) de contact du robot.

On appelle $\mathcal{C}^1_{\text{free}}$, $\mathcal{C}^2_{\text{free}}$ et $\mathcal{T}_{\text{free}}$, les sous-ensembles des espaces concernés pour lesquels le robot n'est pas en collision avec l'environnement, c'est-à-dire qu'aucun point du robot autre que celui ou ceux en contact n'a une ordonnée de valeur négative ou nulle. $\mathcal{C}^1_{\text{free}}$ est défini par

$$\begin{cases} P_{1y} & = & 0 \\ y & > & 0 \\ P_{2y} & > & 0 \end{cases} \tag{2.2}$$

or $P_{1y} = y - l_1 \cos(\theta_1)$ et $P_{2y} = y - l_2 \cos(\theta_2)$. La définition se réécrit donc

$$\begin{cases} y = l_1 \cos(\theta_1) \\ \theta_1 \in \,]-\frac{\pi}{2}, \frac{\pi}{2}[\\ |\theta_1| < \cos^{-1}(\frac{l_2}{l_1} \cos(\theta_2)) \end{cases} \tag{2.3}$$

De même $\mathcal{C}^2_{\text{free}}$ est défini par

$$\begin{cases} y = l_2 \cos(\theta_2) \\ \theta_2 \in \,]-\frac{\pi}{2}, \frac{\pi}{2}[\\ |\theta_1| > \cos^{-1}(\frac{l_2}{l_1} \cos(\theta_2)) \end{cases} \tag{2.4}$$

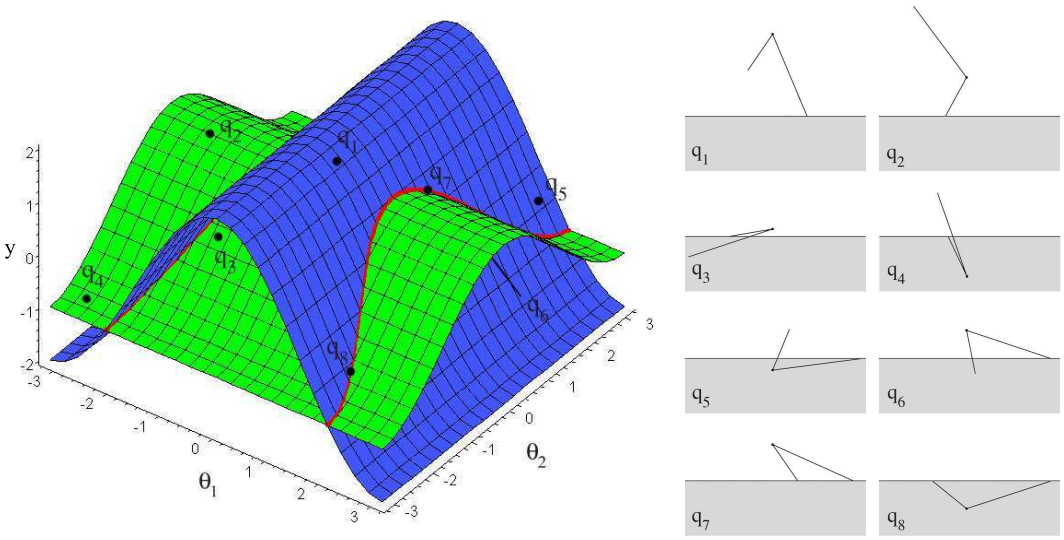


FIG. 2.2 – $\mathcal{C}^1$ *(bleu),* $\mathcal{C}^2$ *(vert) et* $\mathcal{T}$ *(rouge) dans l'espace* (θ_1, θ_2, y) *à* x *constant, et des exemples de configurations.*

Pour $\mathcal{T}_{\text{free}}$, on a

$$\begin{cases} \theta_1 \in \,]-\frac{\pi}{2}, \frac{\pi}{2}\,[\\ \theta_1 = \pm\cos^{-1}(\frac{l_2}{l_1}\cos(\theta_2)) \end{cases} \tag{2.5}$$

On notera que ces trois ensembles sont disjoints (cf Fig. 2.3) et que $\mathcal{T}_{\text{free}}$ est une partie de la frontière des deux autres : $\mathcal{T}_{\text{free}} = \partial\mathcal{C}^1_{\text{free}} \cap \partial\mathcal{C}^2_{\text{free}}$. De même, les $\mathcal{C}^i_{\text{free}}$ sont des frontières de la zone de collision avec le sol.

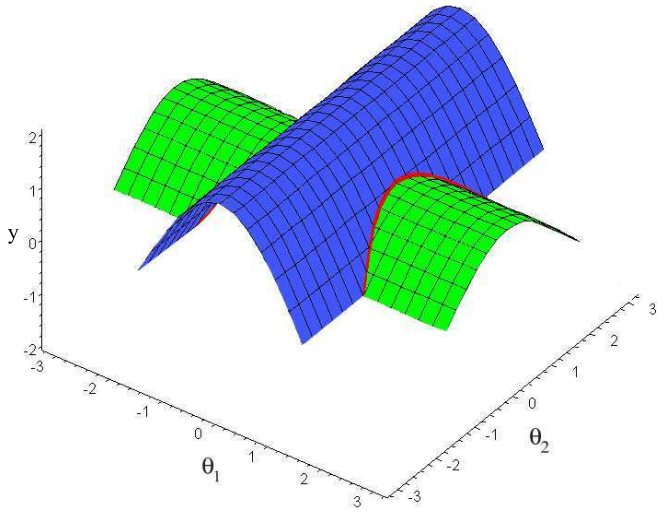


FIG. 2.3 – $\mathcal{C}^1_{free}$ *(bleu),* $\mathcal{C}^2_{free}$ *(vert) et* $\mathcal{T}_{free}$ *(rouge) dans l'espace* (θ_1, θ_2, y) *à* x *constant.*

Considérons la projection

$$p_y(x, y, \theta_1, \theta_2) = (x, \theta_1, \theta_2)$$

Cette projection est injective depuis chacun des ensembles précédents car à un triplet (y, θ_1, θ_2) ne correspond qu'un unique y, soit $y = l_i \cos(\theta_i)$ ($i = 1$ ou 2 selon l'ensemble de départ). Les projections des ensembles restent disjointes, si bien que p_y est injective sur $\mathcal{C}_c = \mathcal{C}^1_{\text{free}} \cup \mathcal{C}^2_{\text{free}} \cup \mathcal{T}_{\text{free}}$, et en particulier p_y est bijective de $\mathcal{C}_c$ sur $p_y(\mathcal{C}_c) = p_y\left(\mathcal{C}^1_{\text{free}}\right) \cup p_y\left(\mathcal{C}^2_{\text{free}}\right) \cup p_y\left(\mathcal{T}_{\text{free}}\right) = \mathbb{R} \times \Theta$ où Θ dénote un ensemble en forme de croix (cf Fig. 2.4)

$$\Theta = \left(\left(\left]-\frac{\pi}{2}, \frac{\pi}{2}\right[\times \left]-\pi, \pi\right]\right) \cup \left(\left]-\pi, \pi\right] \times \left]-\frac{\pi}{2}, \frac{\pi}{2}\right[\right)\right)$$

Sa réciproque est

$$p_y^{-1}(x, \theta_1, \theta_2) = (x, \max\left(l_1 \sin(\theta_1), l_2 \sin(\theta_2)\right) \theta_1, \theta_2)$$

p_y et p_y^{-1} sont continues et donc $\mathcal{C}_c$ est homéomorphe à $\mathbb{R} \times \Theta$ On peut donc étudier et représenter tous les mouvements en contact dans le sous-ensemble $\mathbb{R} \times \Theta$ de $\mathbb{R}^3$, ce qui va en permettre une visualisation.

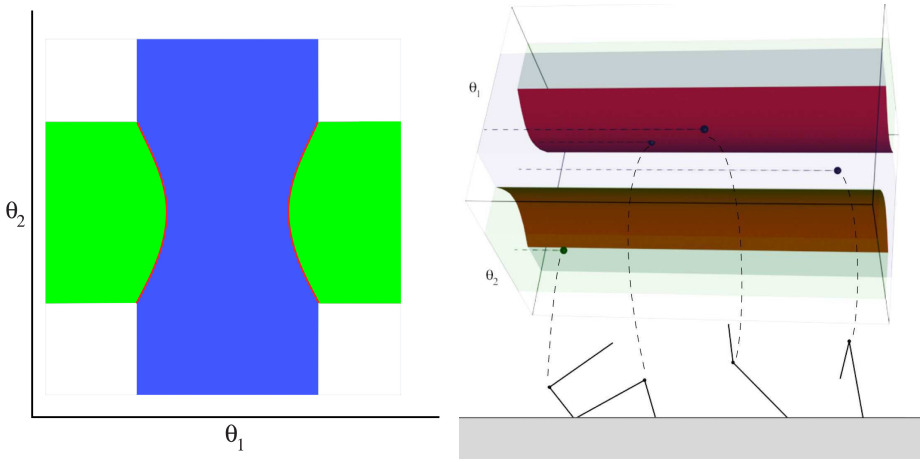


FIG. *2.4 – $\mathcal{C}^1_{free}$ (bleu), $\mathcal{C}^2_{free}$ (vert) et $\mathcal{T}_{free}$ (rouge) se projettent de manière bijective sur le plan (θ_1, θ_2) en 3 ensembles disjoints qui forment une croix $\left(\left(\left]-\frac{\pi}{2}, \frac{\pi}{2}\right[\times \left]-\pi, \pi\right]\right) \cup \left(\left]-\pi, \pi\right] \times \left]-\frac{\pi}{2}, \frac{\pi}{2}\right[\right)\right)$ (à gauche). A chaque point de l'espace produit de cette croix avec la droite des réels, on peut faire correspondre bijectivement une configuration en contact (à droite).*

2.1.2 Sous-espace de contact

Considérons maintenant que le robot est en contact par P_i avec un point fixe donné du sol d'abscisse x_0. Cela correspond à une contrainte bidimensionnelle.

$$\begin{cases} P_{ix} = x_0 \\ P_{iy} = 0 \end{cases} \tag{2.6}$$

soit

$$\begin{cases} x = x_0 - l_i \sin(\theta_i) \\ y = l_i \cos\theta_i \end{cases} \tag{2.7}$$

Celle-ci est une sous-variété de dimension 2 de l'espace de configuration dont nous appelons le sous-ensemble sans collision $\mathcal{C}^i_{x_0}$. La représentation de celui-ci est donnée par la figure 2.5. Sur

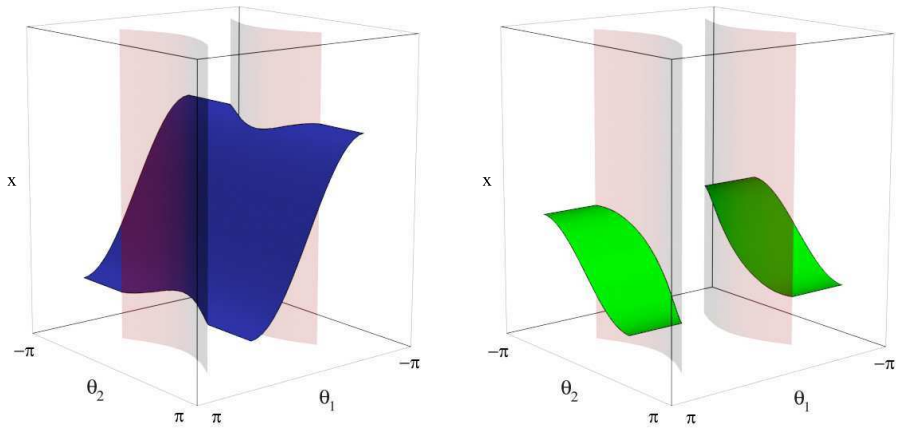


FIG. 2.5 – *Ensemble des configurations sans collision pour un contact fixé de P_1 (à gauche) et P_2 (à droite).*

cette figure est aussi représenté, en rouge transparent l'espace $\mathcal{T}$ des transitions, qui limite les sous-variétés de contact. On notera que bien que sa projection ait deux composantes dans $\mathbb{R} \times \Theta$, une sous-variété correspondant à un contact de P_2 (en vert) est connexe dans l'espace de configuration. Ce découpage est dû à la projection du tore $SO(1)^2$ sur $]-\pi, \pi[^2$. On peut de ce fait passer d'un bord de Θ à son opposé.

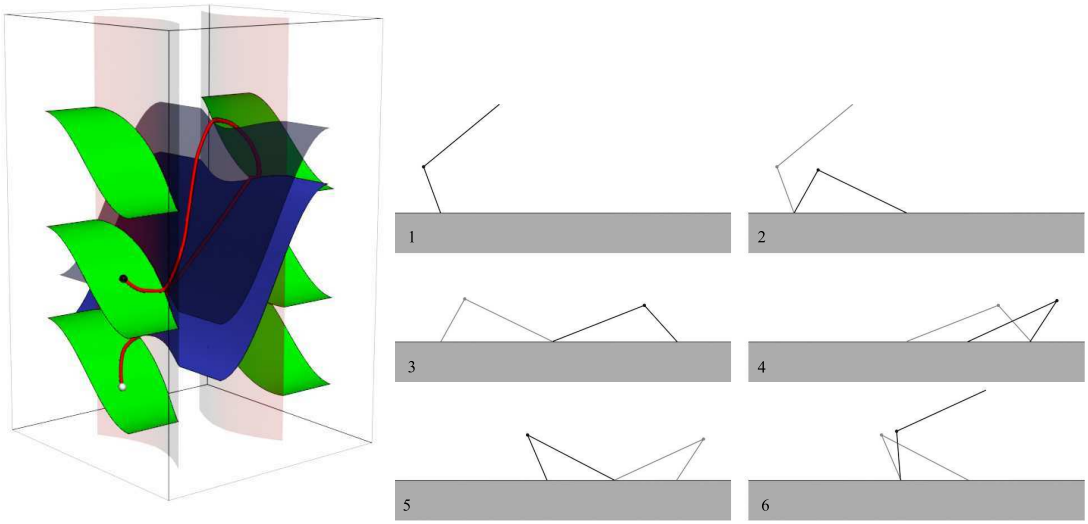


FIG. 2.6 – *Un exemple de marche pour passer d'un point à un autre d'un même espace $\mathcal{C}^i_{free}$. Le robot passe d'une configuration initiale (1, point blanc) à une configuration finale (6, point noir) en passant par quatre points de transition (2 à 5). Une trajectoire possible est illustrée par la ligne rouge.*

Pour le robot, un pas est un chemin dont tous les points sont dans $\mathcal{C}^i_{x_0}$, x_0 fixé, à l'exception des extrémités qui se trouvent dans $\mathcal{T}$. Passer d'un point de $\mathcal{C}^i_{x_0}$ à un point de $\mathcal{C}^i_{x_1}$, $x_0 \neq x_1$ ne peut se faire en restant dans $\mathcal{C}^i_{\text{free}}$: cela correspondrait à un robot qui glisse. Un chemin entre deux tels points demande de planifier plusieurs pas (cf Fig. 2.6).

Une marche consiste en un chemin qui passe périodiquement entre $\mathcal{C}^1$ et $\mathcal{C}^2$, en se restreignant à rester sur la même sous-variété $\mathcal{C}^i_x$ entre deux passages par $\mathcal{T}_{\text{free}}$ (c'est-à-dire à faire

des pas). Une marche peut donc être décrite par une succession de points $(p_1, \cdots, p_l)$ de $\mathcal{T}_{\text{free}}$, telle que deux points consécutifs sont sur la frontière d'un même $\mathcal{C}_x^i$.

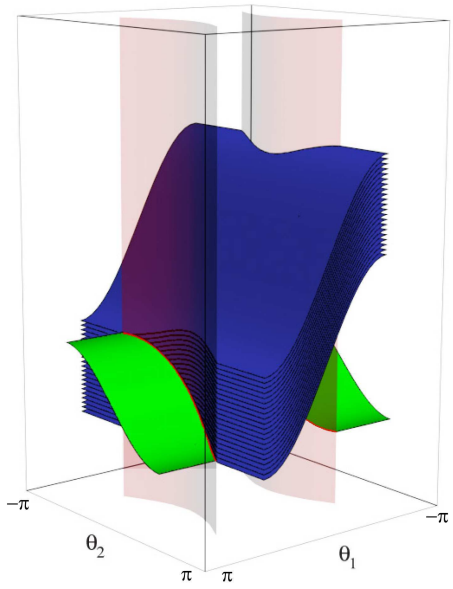


FIG. 2.7 – *Lorsque le robot est sur une sous-variété $\mathcal{C}^2_{x_2}$ (en vert), il a une infinité de possibilités pour un nouveau contact avec P_1 qui correspond à une sous-variété à une dimension, représentée par les courbes rouges. Ces possibilités sont un sous-ensemble d'un feuilletage de $\mathcal{C}^1_{free}$.*

Imaginons que le robot est en contact par P_2 au point d'abscisse x_2. Il a alors un choix continu d'emplacements pour placer P_1 en contact, qui correspond à la longueur du pas : il peut choisir x_1 tel que $l_1 - l_2 < |x_2 - x_1| < l_1 + l_2$ (cf Fig 2.7). Pour un x_1 donné, il n'y a qu'un seul point de $\mathcal{T}$ par lequel passer de $\mathcal{C}^2_{x_2}$ à $\mathcal{C}^1_{x_1}$. L'ensemble des choix possibles engendre une famille de sous-variétés parallèles les unes aux autres (on touche là le fait que $(\mathcal{C}^i_x)_{x\in\mathbb{R}}$ est un *feuilletage* de $\mathcal{C}^i_{\text{free}}$[1].) : chaque sous-variété (ou feuille) correspond à un choix de x_1, et il n'est pas possible de passer directement d'une feuille à une autre.
Selon le choix qu'il fait, le robot ne pourra pas atteindre la même zone avec P_2 au pas suivant : le choix continu d'un contact à un moment influe sur ce que le robot pourra faire ensuite. On va voir qu'en présence d'obstacles, ce choix devient beaucoup plus sensible.

2.1.3 Ajout d'obstacles

On considère maintenant le scénario décrit par la figure 2.8. L'environnement est peuplé par deux obstacles supplémentaires (en plus du sol) sur lesquels, pour simplifier l'exposé (et garder la projection permettant de visualiser dans $\mathbb{R}^3$), le robot ne peut pas prendre de contact.

Ces obstacles réduisent les possibilités de mouvement du robot, ce qui se traduit par des trous dans les sous-variétés que nous étudions. Ces trous perturbent de deux façons la recherche d'un chemin :

[1] Un feuilletage, d'une variété $\mathcal{V}$ de dimension n est une partition $(\mathcal{V}_\alpha)$ de $\mathcal{V}$ en sous-variétés connexes par arc et disjointes, et tels qu'ils soient homéomorphes à $\mathbb{R}^k$ $(k < n)$ au voisinage de n'importe lequel de leur point [Lawson 74]

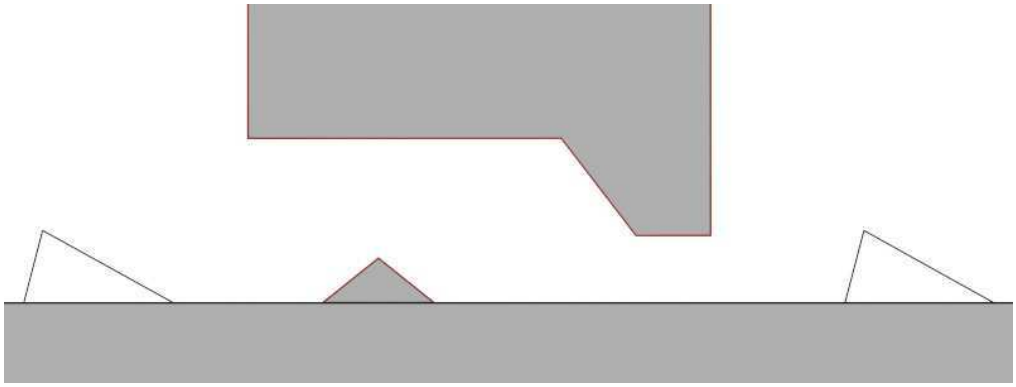


FIG. *2.8 – Scénario de planification avec des obstacles. On cherche à aller de la gauche vers la droite.*

– (i) ils peuvent rendre une variété $\mathcal{C}_x^i$ non connexe. On ne pourra alors pas trouver de chemin sur cette sous-variété entre deux configurations sur deux composants différents,
– (ii) ils peuvent empêcher des changements de contact (trous dans $\mathcal{T}_{\text{free}}$).

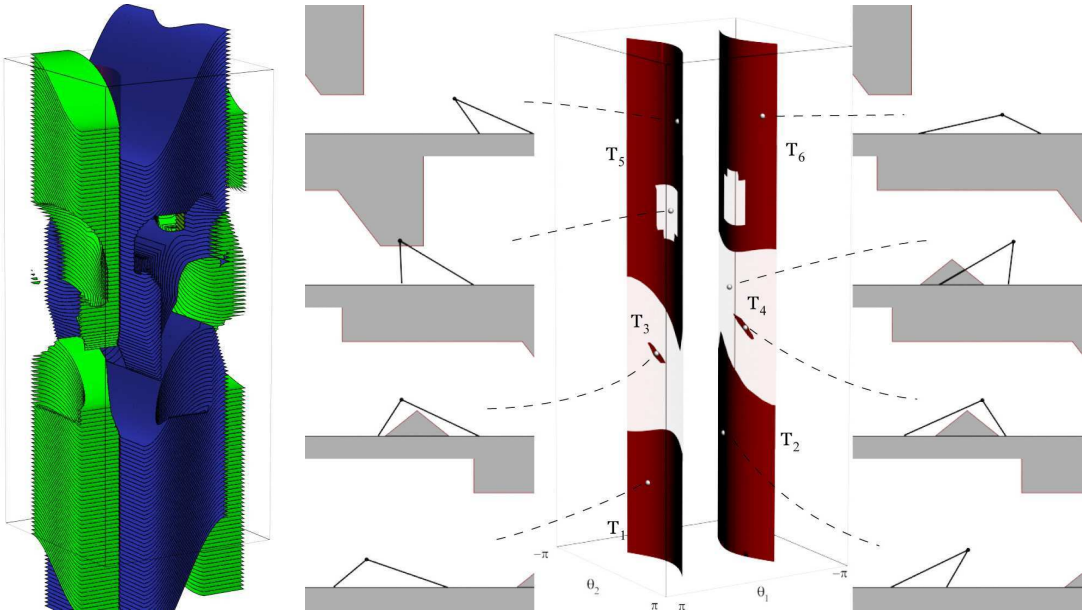


FIG. *2.9 – La projection de l'espace de configuration en contact dans $\mathbb{R}^3$ (à gauche) et l'espace des transitions (à droite) qui, en présence d'obstacles, est découpé en plus de deux composantes connexes.*

La figure 2.9 montre ces trous dans les $\mathcal{C}_x^i$ (à gauche), ainsi que les zones où les transitions ne sont plus possibles. L'espace de transition $\mathcal{T}$ est divisé en six parties connexes. Alors qu'avant on pouvait passer facilement entre les deux composantes connexes, ce n'est maintenant plus le cas. Si le passage entre $\mathcal{T}_1$ et $\mathcal{T}_2$ ou $\mathcal{T}_5$ et $\mathcal{T}_6$ est aisé, en revanche, aller à $\mathcal{T}_3$ ou $\mathcal{T}_4$ requiert de passer par de très petites régions de l'espace. On ne peut même accéder à $\mathcal{T}_4$ que depuis $\mathcal{T}_1$ et à $\mathcal{T}_5$ que depuis $\mathcal{T}_6$. Ainsi pour aller d'une configuration de $\mathcal{T}_5$ à une de $\mathcal{T}_4$ qui aurait été accessible en un seul pas sans les obstacles, le robot doit passer par $\mathcal{T}_6$, $\mathcal{T}_3$, $\mathcal{T}_2$, puis $\mathcal{T}_1$.

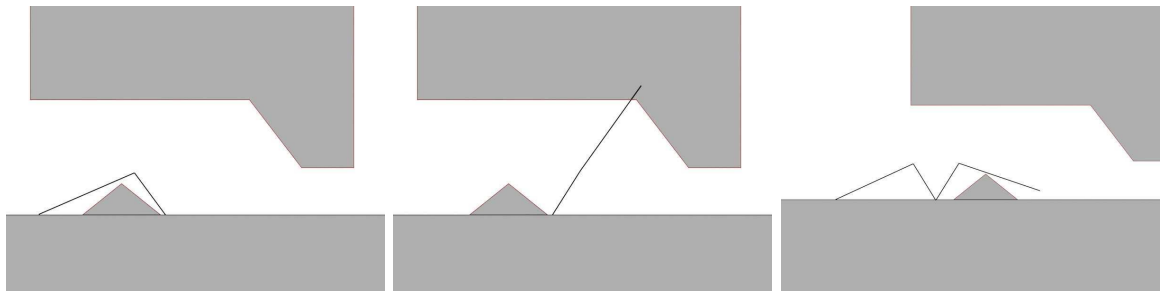


FIG. *2.10 – Deux difficultés de la planification d'appuis : choix du type de contact (gauche et milieu) et choix de la position d'un contact (droite).*

On distingue dans ce scénario deux difficultés qui sont emblématiques de la planification d'appuis, et sont une traduction des perturbations évoquées précédemment. Ces difficultés sont illustrées par la figure 2.10.
Tout d'abord le choix d'un type d'appui peut être crucial. Un mauvais appui, ou un mauvais ensemble d'appuis peut bloquer le robot dans son évolution. Ainsi, passer par une configuration de $\mathcal{T}_4$ (P_1 à droite et P_2 à gauche de l'obstacle triangulaire, voir la figure de gauche), ne permet pas d'avancer plus. En effet, il n'y a pas de passage de $\mathcal{T}_4$ à $\mathcal{T}_5$, car les $\mathcal{C}_x^1$ dont les frontières ont une intersection non vide avec ces ensembles ne sont pas connexes (cas (i)) : comme le montre l'image du milieu, il n'y pas la place nécessaire pour le robot d'être en position complètement dépliée, ce qui est un passage obligé pour aller d'un ensemble à l'autre. C'est donc un **problème de choix discret** : le type de posture (P_1 ou P_2 en premier dans notre cas) a des conséquences importantes sur les possibilités pour le robot d'évoluer.

En second lieu, lorsque apparaissent des petits espaces de transition comme $\mathcal{T}_3$, le choix de l'endroit des contacts devient très important. $\mathcal{T}_3$ ne peut être atteint qu'à partir d'un petit sous-ensemble de configurations de transition de $\mathcal{T}_2$. Choisir un contact trop avant l'obstacle en empêche le franchissement (figure de droite) : l'espace $\mathcal{C}_x^1$ à partir du point de transition a une intersection vide de sa frontière avec $\mathcal{T}_3$ (point (ii)).
C'est ici un **problème de choix continu** d'un contact acceptable dont les conséquences peuvent se répercuter fortement dans la suite.

Malgré les difficultés, on peut quand même trouver un plan pour ce scénario. Il est illustré par la figure 2.11. L'image de gauche montre une trajectoire solution, et les sous-espaces par lesquels elle passe. A droite sont illustrés les points de transition (en haut) et la trajectoire du robot dans l'espace 2D (en bas).

2.1.4 Similitudes et différences avec la planification d'appuis

Un certain nombre d'idées vues dans ce scénario se retrouvent dans la planification d'appuis. Tout d'abord le déplacement sur des sous-variétés. La donnée de contacts se fait par le biais d'égalités qui définissent de tels sous-espaces. On retrouve le fait que ces sous-espaces sont des frontières de la représentation des obstacles dans l'espace de configuration. De manière générale, on a les mêmes structures (espace de transition, feuilletage, etc.).

Ensuite, la stabilité, absente ici, est similaire à un obstacle dans la mesure où elle interdit une région de l'espace de configuration. La seule différence est que la frontière de cette région ne peut être un sous-espace de contact.

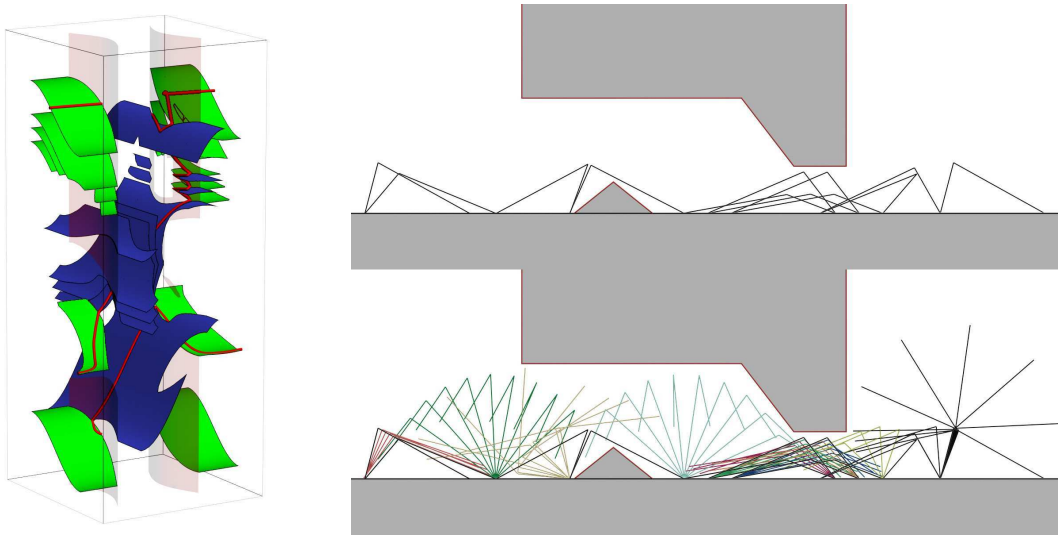


FIG. *2.11 – Le plan final.*

La grande différence vient du nombre d'espaces considérés. Dans cet exemple, il n'y a que deux emplacements de contact possibles (P_1 et P_2) et une seule surface d'appui, générant trois sous-espaces. Ce nombre d'espaces explose de manière combinatoire avec le nombre d'emplacements de contact et de surfaces d'appui. Le choix discret, très réduit ici, devient alors un problème majeur.

Enfin, il y a une différence dans les différentes dimensions d'espace : ici, le chemin ne passe que par des espaces de dimensions 2 et 0 : les surfaces $\mathcal{C}_x^1$ et les points de transition. Sur un robot tel qu'un humanoïde, il y a la possibilité de prendre plus de deux appuis, ce qui donne lieu à plus de deux dimensions différentes de variétés, avec des dimensions plus élevées. En particulier, la transition entre un contact et un autre se fait par un espace de dimension non nulle dans lequel il faut trouver un chemin (qui correspond à transférer le poids du robot d'un appui à l'autre), ou à partir duquel on peut rajouter un nouveau contact, définissant une nouvelle sous-variété de dimension inférieure.

2.2 Définitions

L'état d'un robot $\mathcal{R}$ à n articulations évoluant dans un espace de travail $\mathcal{W}$ à 3 dimensions peut être à tout moment complètement représenté par $n+6$ variables $\mathbf{q}$ correspondant aux n valeurs des angles des articulations (degrés de liberté internes), et aux 6 paramètres de position et orientation d'un de ses corps relativement à un repère de $\mathcal{W}$. Le robot est donc classiquement représenté par un point dans l'espace

$$\mathcal{C} = \times_n^1 [q_i^-, q_i^+] \times \mathbb{R}^3 \times SO(3) \subset \mathbb{R}^{n+3} \times SO(3) \tag{2.8}$$

où q_i^- et q_i^+ sont les limites inférieure et supérieure de la $i^{\text{ème}}$ articulation. Cet espace est appelé *espace de configuration*. Pour une configuration donnée $\mathbf{q}$, $\mathcal{R}(\mathbf{q})$ est la représentation du robot dans l'espace de travail.
Un obstacle O dans l'espace de travail devient un volume O^C dans $\mathcal{C}$ défini comme $O^C =$

$\{\mathbf{q} \in \mathcal{C} | \mathcal{R}(\mathbf{q}) \cap O \neq \emptyset\}$, soit l'ensemble des configurations pour lesquelles le robot et l'obstacle sont en collision dans l'espace de travail. Par suite on peut construire $\mathcal{C}_{\text{obs}} = \bigcup O_i^c$, l'ensemble des configurations où le robot est en collision avec un obstacle.
Si r_i désigne le ième corps du robot, et $r_i(\mathbf{q})$ son image dans $\mathcal{W}$ à la configuration $\mathbf{q}$, on définit aussi l'ensemble des auto-collisions du robot $\mathcal{C}_{\text{auto}} = \{\mathbf{q} | \exists i \neq j, r_i(\mathbf{q}) \cap r_j(\mathbf{q}) \neq \emptyset\}$.
On construit alors le sous-ensemble de $\mathcal{C}$ libre de toute collision $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \{\mathcal{C}_{\text{obs}} \cup \mathcal{C}_{\text{auto}}\}$. Les algorithmes de planification classiques recherchent un chemin dans ce dernier ensemble.

Le robot est cependant 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 $\mathcal{C}_{\text{free}}$, mais doit faire des incursions dans $\partial\mathcal{C}_{\text{obs}}$, la frontière de $\mathcal{C}_{\text{obs}}$. Hors de $\partial\mathcal{C}_{\text{obs}}$, le robot n'est soumis qu'à la gravité, et se trouve dans une phase balistique. Cela est par exemple le cas lors d'un saut, ou de la phase de vol d'une course. Pendant cette phase, le mouvement du robot dans $\mathcal{C}_{\text{free}}$ est contraint à un sous-ensemble de chemins, en particulier le trajet du centre de gravité est parfaitement déterminé par les conditions à la sortie de $\partial\mathcal{C}_{\text{obs}}$.
On ne peut cependant entièrement représenter ce type de trajectoire dans $\mathcal{C}$ qui est un espace purement géométrique. Il faudrait pour cela y ajouter l'espace $\dot{\mathcal{C}}$ des vitesses, et le temps, ce qui accroît fortement la dimension de l'espace de recherche. Tout au long de ce travail, on se restreindra au cas où 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 $\partial\mathcal{C}_{\text{obs}}$. Plus précisément, il doit se trouver dans $\mathcal{C}_c = \partial\mathcal{C}_{\text{obs}} \setminus \mathcal{C}_{\text{auto}}$, l'espace des configurations de contact sans auto-collision. Cet espace est une sous-variété de $\mathcal{C}$ de dimension $n+5$. Il représente l'ensemble des configurations pour lesquelles le robot a au moins un point en contact avec l'environnement, sans avoir aucune collision (stricte) avec celui-ci et sans être en auto-collision (au sens large : il n'y a pas de contact entre les corps du robot). On peut écrire

$$\mathcal{C}_c = \left\{ \mathbf{q} \in \mathcal{C} | \exists i, \mathcal{R}(\mathbf{q}) \cap O_i \neq \emptyset, \mathcal{R}(\mathbf{q}) \cap \overset{\circ}{O}_i = \emptyset \ et \ \forall j \neq k, r_j(\mathbf{q}) \cap r_k(\mathbf{q}) = \emptyset \right\} \tag{2.9}$$

où $\overset{\circ}{X}$ désigne l'intérieur de X.
Parmi tous les contacts possibles entre le robot et l'environnement certains ont un intérêt particulier, alors qu'il est souhaitable d'en éviter d'autres. En effet, il sera par exemple plus stable de prendre appui avec une surface du robot qu'avec un unique sommet. A l'opposé, certaines parties fragiles ne devraient pas avoir à supporter d'efforts. On prédéfinit donc sur le robot un certain nombre de points de contact, points qui seront mis en correspondance avec un point de l'environnement. On préférera des points sur des surfaces planes du robot ou, mieux encore, des surfaces déformables, susceptibles d'offrir des contacts plan-plan stables. Outre les raisons ci-dessus, le choix de points particuliers est de plus une façon de simplifier le problème, mais a aussi une justification supplémentaire dans le cadre de contacts plan-plan. En effet dans le cas où une surface plane $S^{\mathcal{R}}$ du robot est en contact avec une surface plane S^E de l'environnement, il y a une infinité de paires de points $(P^{\mathcal{R}}, P^E)$ en contact ($P^{\mathcal{R}} \in S^{\mathcal{R}}$ et $P^E \in S^E$). La donnée de n'importe laquelle de ces paires est suffisante pour définir le contact. On peut donc se fixer sans perte de généralité un point P_0^R sur $S^{\mathcal{R}}$ qui sera apparié avec un point de S^E étendue en conséquence. Soit $\mathcal{P}_{\mathcal{R}}$ l'ensemble de ces points. Les coordonnées d'un point P_i^j sont données dans le repère du corps r_j du robot.

2.3 Travaux connexes

2.3.1 Planification dans $\mathcal{C}_{\text{free}}$ et sur un sous-espace.

La planification d'appuis est un cas particulier de la planification de mouvements pour laquelle existe une abondante littérature. [Latombe 91] constitue un ouvrage de référence qui donne un état de l'art des travaux effectués jusqu'à 1990. Nous en extrayons un algorithme dont nous nous inspirons dans ce travail (cf chapitre 5). [LaValle 06] reprend le travail des années suivantes qui ont notamment vu l'avènement des méthodes probabilistes au rang desquelles la famille des PRM (*probabilistic roadmaps*) [Kavraki 96] et celle des RRT (*Rapidly exploring Random Trees*) [LaValle 98] forment les deux classes d'algorithmes les plus utilisées et adaptées.

Un algorithme PRM construit un graphe dans $\mathcal{C}_{\text{free}}$ dont les sommets sont des configurations tirées aléatoirement, et les arêtes des chemins de $\mathcal{C}_{\text{free}}$. Ce graphe permet d'appréhender la structure de $\mathcal{C}_{\text{free}}$ en trouvant ses différentes composantes connexes. Un chemin entre la configuration initiale et celle finale du problème consiste en un chemin sans collision depuis la configuration initiale jusqu'à un sommet $\mathbf{q}_1$ du graphe, un chemin sans collision d'un sommet $\mathbf{q}_2$ à la configuration finale, et entre les deux un chemin dans le graphe, de $\mathbf{q}_1$ à $\mathbf{q}_2$. Les chemins entre deux points sont trouvés par un planificateur local simple (qui dans le cas le plus simple se borne à chercher si la ligne droite entre ces deux points est bien dans $\mathcal{C}_{\text{free}}$).

Un algorithme RRT construit un arbre dont la racine est la configuration initiale, les nœuds des configurations et les arêtes des chemins de $\mathcal{C}_{\text{free}}$, jusqu'à arriver à la configuration finale du problème. Pour ce faire, il tire à chaque itération une configuration aléatoire $\mathbf{q}$ et à partir du nœud de l'arbre le plus proche $\mathbf{q}_0$ construit une nouvelle branche en direction de $\mathbf{q}$, c'est-à-dire un nœud $\mathbf{q}'$ sur le segment $\mathbf{q}_0\mathbf{q}$ et une arête de $\mathbf{q}_0$ à $\mathbf{q}'$.

Un pas, au sens de notre problème de planification, est un déplacement sur une sous-variété, qui correspond à des contraintes dites d'*égalité holonomique*. Trouver un pas revient à résoudre un problème de planification où le chemin est contraint à un sous-ensemble de l'espace de configuration. Lorsque la sous-variété est facilement paramétrable, on peut la considérer comme un nouvel espace de configuration et utiliser les techniques précédentes. Par exemple un robot avec un seul corps en contact peut être considéré comme un robot dont ce corps est la base fixe, dont l'espace de configuration devient celui des n angles internes, au signe près si nécessaire. Les sous-variétés sont cependant souvent trop complexes pour cela et il faut donc les regarder comme des variétés plongées dans $\mathcal{C}$. 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. Or celle-ci étant de dimension inférieure à celle de $\mathcal{C}$, elle est de mesure nulle dans $\mathcal{C}$. 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), [Cortés 02] 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 on définit ensuite la posi-

tion par cinématique inverse. Les valeurs des articulations hors des boucles sont déterminées de manière complètement aléatoire.

Si en revanche les contraintes sont quelconques, [Stilman 07] propose une adaptation plus générale des RRT dont le principe est de projeter un point nouvellement construit $\mathbf{q}'$ soit directement sur la sous-variété, soit sur son espace linéarisé tangent en $\mathbf{q}_0$.

2.3.2 Locomotion et manipulation

La planification d'appuis s'inscrit dans le cadre de la locomotion de robot multi-membres pour laquelle 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 le contact. La classe de ce cadre la plus étudiée et utilisée est la marche d'un humanoïde qui est une instance particulière du problème que nous nous posons. Ce problème est aussi similaire à la manipulation d'objet : les contacts y deviennent des préhensions, entre lesquelles le robot suit un chemin. On peut voir 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, faites de sous-variétés dont les frontières s'intersectent. Dans les deux cas, trouver un chemin met en jeu à la fois des déplacements et des choix de contacts/prises. Les algorithmes de planification résolvant ces problèmes se partagent en deux groupes selon leur façon d'approcher le problème : planifier d'abord une trajectoire générale du robot, puis trouver les appuis qui lui correspondent, ou bien commencer par le choix des appuis puis trouver les trajectoires entre deux ensembles de contacts successifs. Notre travail tombe dans la deuxième catégorie. Cependant, sur une idée présentée au chapitre 6 (l'utilisation d'une trajectoire guide), nous empruntons à la première.

2.3.3 Mouvement précédant les appuis

Lorsque le placement précis des appuis n'est pas nécessaire, on peut laisser ce choix de côté pour se concentrer sur le chemin global du robot. Une fois ce chemin obtenu, on génère alors les points d'appuis nécessaires. Généralement ce type de planification à deux niveaux est très rapide car le calcul du chemin se fait en petite dimension, et le placement des contacts est donnée par des heuristiques simples, basées quand cela est possible sur des mouvements cycliques.

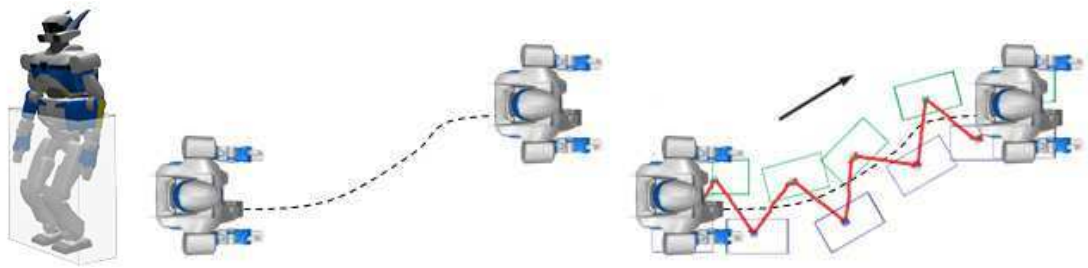


FIG. *2.12 – Exemple de mouvement précédent les appuis. La partie inférieure du robot est approximée par une boîte (à gauche), pour trouver un chemin global (milieu) pour lequel on génère alors une séquence d'appuis (à droite).*

Ce type d'approche est souvent utilisée pour la marche (humanoïde dans les exemples que nous connaissons, mais l'extension à d'autres robots comme les quadrupèdes est aisée) sur sol plat ou suffisamment régulier. L'idée est de prendre un volume (un cylindre ou une boîte) englobant le robot [Kuffner 98] ou sa partie inférieure [Pettré 03] et de planifier un chemin horizontal (l'altitude est fixée par rapport au sol) sans collision puis de lui rajouter une notion de temps. Un contrôleur, basé sur des mouvements cycliques pré-enregistrés sert alors à faire suivre la trajectoire obtenue en générant les mouvements de tout le robot. Dans [Pettré 03], les degrés de liberté du robot sont découpés en deux parties, une de locomotion, qui concerne les jambes, l'autre réactive, qui sert dans un deuxième temps à éviter d'éventuelles collisions que le suivi de la trajectoire du cylindre aurait occasionnées (décomposition fonctionnelle). La même démarche est utilisée dans [Yoshida 05] (cf Fig. 2.12) en utilisant un générateur de pas dynamique le long de la trajectoire. Les degrés de liberté non utilisés par la marche sont affectés à des missions de manipulation et sont planifiés en même temps que le chemin global. L'utilisation d'une marche dynamique peut amener à ne pas suivre parfaitement ce chemin par la suite. [Yoshida 06] intègre un mécanisme de déformation itérative a posteriori de ce chemin pour corriger les problèmes de collisions que l'erreur de suivi génère.

En manipulation, le pendant de cette approche est de planifier le chemin de l'objet manipulé seul puis d'en déduire une séquence de prises et relâchements de l'objet par les robots manipulateurs [Koga 94]. Il faut cependant s'assurer lors de la génération du chemin qu'il existe toujours une possibilité de maintenir l'objet en tout point, ce que fait [Koga 94]. Ce type de vérification n'est pas nécessaire pour les exemples de travaux que nous donnons avant car, par de simples considérations géométriques, on s'assure de la possibilité de mettre les pieds au sol.

L'approche s'étend à des sols plus complexes [Li 03] en 2.5D. Le choix des contacts devient alors plus compliqué et il n'existe parfois pas de solution pour suivre la trajectoire. La complexité du sol peut aussi tenir à la présence de peu d'endroits sur lesquels prendre appui. Dans ce cas il peut devenir difficile de trouver un chemin qui pourra être suivi par les contacts. [Boissonnat 00] propose dans le cas d'un robot araignée simplifié une méthode pour calculer explicitement l'espace des configurations stables dans lequel le chemin peut alors être trouvé.

FIG. *2.13 – Planification de mouvements pour un humain avec pré-définition d'emplacements de contact dans l'environnement.*

Enfin l'approche du mouvement avant le choix des contacts a été utilisé dans un cadre similaire à notre travail [Kalisiak 01] (cf Fig. 2.13) : un humanoïde peut se déplacer en utilisant ses mains et ses pieds. Pour cela des positions de contact sont prédéfinies qui correspondent

soit aux mains, soit aux pieds, soit aux deux, et le robot se déplace sous l'effet d'un champ de potentiel qui agit sur son centre de masse (utilisation de l'algorithme RPP, voir [Latombe 91]. Le choix des contacts se fait en parallèle de la planification du chemin du centre de masse selon des conditions encodées dans une machine d'états finie qui permet plusieurs modes de locomotion (marcher, ramper, grimper, etc). La position des membres en contact est définie par cinématique inverse.

2.3.4 Appuis précédant le mouvement

Lorsque le choix des contacts revêt de l'importance, par exemple lorsque l'environnement est trop complexe pour s'assurer de pouvoir suivre un chemin, il faut commencer par choisir les contacts dont on déduit ensuite un chemin.

Cette méthode est généralement plus lente que celle précédente mais elle est aussi plus générale dans la mesure où les simplifications qui étaient faites dans la première méthode (boîte englobante, mouvements cycliques, etc.) sont autant de contraintes sur le mouvement qu'il n'est pas toujours possible de suivre, et ce de deux manières :

- les contraintes sur la génération du chemin global peuvent empêcher de trouver celui-ci. Par exemple avec l'utilisation du cylindre, la présence d'obstacles sur le sol peut bloquer le planificateur à cause des collisions, quand bien même le robot n'aurait aucun problème à les enjamber,
- le chemin fait passer par des endroits dont le planificateur n'avait pas idée de la difficulté et dans lesquels des appuis stables ne sont pas possibles, par exemple.

Les problèmes de planification d'appuis/manipulation se divisent en deux catégories qui se distinguent par la structure de leur espace de configuration :

- l'espace est une collection de sous-variétés. C'est le cas lorsque les emplacements de contacts de l'environnement/prises sont discrets. Chaque variété correspond alors à un ensemble d'appariements entre un contact de l'environnement et un emplacement de contact du robot.
- l'espace est une collection de feuilletages, comme lorsque le choix de la position des contacts est continu, ce qui est notre cas.

2.3.4.1 Contacts discrets

FIG. *2.14 – Le robot LEMUR escaladant un mur d'escalade (à gauche) et un résultat de planification d'appuis pour le robot HRP-2 avec utilisation d'une librairie de trajectoire.*

De par la structure de l'espace, il s'impose assez naturellement d'utiliser un graphe d'adjacence entre les sous-variétés (ou leurs composantes connexes) : deux variétés sont adjacentes si il existe un chemin pour passer de l'une à l'autre, ce qui se réduit à trouver une configuration qui leur est commune. Dans cette catégorie, on trouve les travaux de Bretl [Bretl 05] (cf Fig. 2.14) sur la planification pour robot grimpeur. La planification se fait en deux étapes : tout d'abord la construction d'un graphe d'adjacence de manière incrémentale à partir de la variété contenant la configuration initiale jusqu'à arriver à la variété de la configuration finale. Pour trouver les voisins d'une variété $\mathcal{V}$, l'algorithme regarde les variétés $\mathcal{V}_i$ qui correspondent à un contact en plus ou en moins et cherche une configuration $\mathbf{q}_i$ dans $\mathcal{V} \cap \mathcal{V}_i$. Si cette configuration existe, les variétés sont adjacentes. Une fois que le graphe contient la variété finale, un chemin de ce graphe est proposé. Il faut alors vérifier la possibilité de passer entre les $\mathbf{q}_i$ de ce chemin. Cela se fait par le biais d'un algorithme PRM utilisé sur les sous-variétés correspondants à chaque paire de $\mathbf{q}_i$ consécutifs. En cas d'échec la construction du graphe est reprise de façon à proposer un autre chemin.

La force de cet algorithme consiste en la réduction de dimension obtenue grâce au graphe, qui permet une bonne recherche de l'espace en évitant d'avoir à faire les appels coûteux aux PRM à chaque fois qu'on y ajoute un nœud. Cela est possible car la probabilité de trouver un chemin entre deux $\mathbf{q}_i$ est très forte. En d'autres termes, les variétés sont majoritairement connexes.

Par la suite ce travail a été adapté pour les robots humanoïdes [Hauser 05] en permettant à ceux-ci de prendre contact grâce à des emplacements prédéfinis, et en améliorant la façon de tirer les configurations dans l'utilisation de la PRM. Les positions possibles des contacts restent données à l'avance. [Hauser 06] rajoute l'utilisation de trajectoires prédéfinies qui sont adaptées aux contacts utilisés pour avoir des chemins plus naturels d'un point de vue visuel (cf Fig. 2.14). Ces trajectoires peuvent aussi aider à choisir les points de contact.

2.3.4.2 Contacts continus

La discrétisation des emplacements de contact, quand elle n'est pas dictée par la nature même du problème mais par un souci de simplification, pose quelques problèmes. Tout d'abord se pose la question de comment discrétiser. Ensuite la discrétisation peut faire manquer des solutions.

Lorsque le choix des contacts est continu, on est en présence de feuilletages, ce qui signifie que dans un espace donné, le robot ne peut bouger que dans certaines directions. Il existe cependant un résultat démontré dans [Alami 95] qui dit que pour un feuilletage $\mathcal{F}$ qui est l'intersection de deux autres $\mathcal{F}_1$ et $\mathcal{F}_2$, si il existe un chemin quelconque (c'est-à-dire sans tenir compte des directions imposées par les feuilles de $\mathcal{F}$) entre deux configurations de feuilles de $\mathcal{F}$, alors il existe une séquence finie de chemins sur des feuilles de $\mathcal{F}_1$ et $\mathcal{F}_2$ qui permet d'aller d'une configuration à l'autre[2].
Cette *propriété de réduction* peut être très utile car elle permet de considérer une composante connexe d'un feuilletage comme un seul et même 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 compression de l'espace. Cette propriété sert dans [Siméon 04] à la construction d'un

[2]La démonstration dans le cadre original demande que le robot ne soit pas en contact avec les obstacles. Dans le cadre d'un robot humanoïde, il faut rajouter la notion de stabilité à celle d'obstacle ce qui rend une possible extension non triviale

graphe de manipulation. On notera qu'identifier les composantes connexes des feuilletages (par utilisation de PRM) a cependant un coût qui peut devenir grand quand celui-ci n'a pas une paramétrisation simple. Or dans notre cas, il y aurait beaucoup de feuilletages à étudier, rendant cette méthode impraticable. Elle reste cependant une méthode élégante et puissante lorsque le nombre d'espaces est petit et il serait intéressant d'étudier son extension aux robots humanoïdes. Cependant, la propriété de réduction ne dispense pas de trouver par la suite la séquence dont on connaît l'existence.

Pour trouver une séquence de pas il existe plusieurs méthodes dans le cadre de la marche (sur sol non plat). [Chestnutt 03] propose d'utiliser un algorithme A*, étant donné un ensemble de placement de pieds possible. [Chestnutt 07] adapte en ligne cet ensemble. Cette méthode est suffisamment rapide pour planifier en temps réel dans un environnement changeant. [Choi 03] utilise une 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, de façon à lisser le résultat.

Tous ces travaux ne sont cependant pas facilement adaptables au cadre de cette thèse car ils restent dans un nombre restreint de sous-espaces. Le nombre des nôtres au contraire explose avec le nombre d'objets sur lesquels prendre contact et le nombre d'emplacements de contact possibles sur le robot : pour n emplacements et p objets, le nombre d'espaces possibles pour un 1 à k contacts est

$$\sum_{i=1}^{k} C_n^k \cdot p^k \tag{2.10}$$

Pour $n = 12$, $p = 3$ et $k = 4$, comme c'est le cas dans les planifications de la fin de cette thèse, ce nombre est 46 665 ! Certains de ces espaces sont cependant vides.

Le cadre de cette thèse peut être vu comme une jonction entre [Siméon 04] avec lequel nous partageons le type de structure d'espace et [Bretl 05] pour le grand nombre d'espaces potentiellement à considérer. Notre façon de résoudre le problème est plus proche du second travail avec lequel nous partageons des idées dans la construction de l'arbre. Les grandes différences sont dans la continuité des emplacements de contact qui impose de trouver une façon efficace d'en choisir certains, et la façon d'orienter la construction du graphe. Un travail qui s'attaque aux mêmes problèmes (avec cependant un plus petit nombre d'espaces) dans un cadre plus proche de la manipulation est [Hauser 07] où le choix des points de changement de mode, équivalent au choix des points de contact, est dirigé par une approche inspirée des RRT.

2.4 Conclusion

Nous avons, dans ce chapitre, mis en évidence les difficultés du problème de planification auquel s'attaque ce travail. Ces difficultés se trouvent dans l'interdépendance des choix discrets et continus qui se posent tout au long de la planification, et qui, bien que locaux, peuvent influencer des choix à venir lointains. Nous avons passé en revue l'état de l'art relatif, en locomotion et manipulation et désigné la catégorie dans laquelle tombe notre approche : la

planification “appuis précédant le mouvement”. Nous avons aussi mis en évidence qu'à notre connaissance, aucun travail ne s'attaque à la fois à la combinatoire du choix discret et à la structure d'espace particulière (feuilletée) issue du choix continu.
Maintenant que le problème est posé, nous allons introduire un outil de base du planificateur proposé dans cette thèse : *le générateur de posture*.

Chapitre 3

Génération de posture

La donnée d'un ou plusieurs contacts à respecter définit une sous-variété $\mathcal{C}_0$ de $\mathcal{C}$ dont on ne peut généralement trouver facilement un point tant les équations qui la définissent sont non-linéaires. En particulier, puisque $\mathcal{C}_0$ est de mesure nulle dans $\mathcal{C}$, on ne peut trouver de point dessus par tirage aléatoire de configuration. Différents travaux en planification s'attaquent à se problème de recherche de configuration sur un sous-espace, que ce soit dans le cas de tâches à accomplir [Stilman 07] ou en présence de boucles cinématique fermées [Cortés 03]. L'idée générale est de projeter un point, possiblement déjà proche, sur $\mathcal{C}_0$, en réduisant la distance de ce point à la sous-variété.
Si les contacts définissent un sous-espace sur lequel doit se trouver la configuration, celle-ci est aussi contrainte par un ensemble de conditions sur la géométrie et la physique du robot, telles que les collisions, la stabilité, etc. Nous détaillons ici une méthode pour générer sur un sous-espace une configuration qui respecte de telles contraintes, par le biais de l'écriture et la résolution d'une optimisation sous contraintes. Des travaux similaires ont été réalisés, notamment autour de l'humain virtuel Santos™ [1] ([AbdelMalek 07], ainsi que [Farrell 07] pour un état de l'art sur le sujet). L'utilisation d'une routine d'optimisation pour chercher à satisfaire des contraintes pour un robot humanoïdes, ce que nous présentons ici a aussi été proposé dans le cadre de la marche satique par [Bourgeot 02].

3.1 Définition du problème

On se place dans un contexte général où le robot doit réaliser un certain nombre de tâches et on cherche une configuration répondant au problème. Une tâche est un objectif à atteindre, descriptible par un ensemble d'égalités et d'inégalités, écrite dans un espace lié au robot. Par exemple un contact consiste à imposer, dans l'espace de travail, une distance nulle entre une partie d'un corps du robot et une partie d'un objet de l'environnement. Éviter une collision consiste à garder positive la distance entre un corps et un objet, ou entre deux corps, dans ce

[1] http ://www.digital-humans.org/

même espace. On peut aussi vouloir du robot qu'il regarde un objet, etc. Si pour une tâche T_i, on note g_i sa partie égalité et h_i sa partie inégalité, il faut donc trouver une solution au problème

$$\mathcal{Q} = \begin{cases} q_i^- \leq q_i \leq q_i^+, \quad \forall i \in [|1, n|] & (3.1) \\ \epsilon_{ij} \leq d(r_i(\mathbf{q}), r_j(\mathbf{q})), \quad \forall (i,j) \in \mathcal{I}_{\text{auto}} & (3.2) \\ \epsilon_{ik} \leq d(r_i(\mathbf{q}), O_k), \quad \forall (i,k) \in \mathcal{I}_{\text{coll}} & (3.3) \\ s(\mathbf{q}) \leq 0 & (3.4) \\ g_i(\mathbf{q}) = 0 \quad \forall\ T_i & (3.5) \\ h_i(\mathbf{q}) \leq 0 \quad \forall\ T_i & (3.6) \end{cases}$$

Les inégalités (3.1) sont les bornes sur les valeurs des articulations. On considère ici que toutes les articulations n'ont qu'un degré de liberté (liaison pivot ou glissière). Pour des articulations plus complexes (liaison rotule par exemple), il faudrait écrire les limitations articulaires sous une forme plus générale $b(\mathbf{q}) \leq 0$.
Les équations (3.2) et (3.3) représentent respectivement les contraintes de non auto-collision et non collision. $d(A, B)$ désigne la distance (signée) entre deux objets, et les ϵ_{ij} et ϵ_{ik} sont des distances de sécurité à respecter pour un couple d'objets donné. Habituellement, les ϵ_{ij} d'une part, les ϵ_{ik} de l'autre, sont égaux entre eux et positifs. On se réserve cependant la possibilité de les régler séparément, ainsi que de leur donner des valeurs négatives si la géométrie des objets prend déjà en compte une marge de sécurité. $\mathcal{I}_{\text{auto}}$ est un sous-ensemble de $[|1, n+1|]^2$: il n'est pas nécessaire de considérer toutes les paires de corps du robot car certaines ne peuvent jamais être en collision, et il n'est aussi pas toujours possible de décrire la non collision sous forme d'une distance minimum à ne pas dépasser (cf. 3.4.3). Si on note N_O le nombre d'objets de l'environnement, $\mathcal{I}_{\text{coll}}$ est une partie de $[|1, n+1|] \times [|1, N_O|]$: il faut au moins en retirer les couples (r_i, O_k) qui doivent être en contact.
s est un critère de stabilité écrit sur les seules variables articulaires $\mathbf{q}$, par exemple le critère de projection du centre de gravité dans le polygone de support (cf 3.4.4).

Cette forme du problème ne permet cependant pas de prendre en compte des critères de stabilité avancés ou les limites sur les couples articulaires. Cela demande de considérer comme variables supplémentaires les forces $\mathbf{f}$ appliquées sur le robot et les couples τ. De manière générale, on peut vouloir utiliser d'autres variables *intermédiaires* $\mathbf{v}$, qui servent à écrire le problème bien que leur valeurs finales n'aient pas d'importance. On s'intéresse uniquement à l'existence de valeurs admissibles pour de telles variables vis-à-vis d'un ensemble de contraintes.
Le problème se réécrit

$$\mathcal{Q}' = \begin{cases} q_i^- \leq q_i \leq q_i^+, \quad \forall i \in [|1, n|] & (3.7) \\ \tau_i^- \leq \tau_i \leq \tau_i^+, \quad \forall i \in [|1, n|] & (3.8) \\ \epsilon_{ij} \leq d(r_i(q), r_j(q)), \quad \forall (i,j) \in \mathcal{I}_{\text{auto}} & (3.9) \\ \epsilon_{ik} \leq d(r_i(q), O_k), \quad \forall (i,k) \in \mathcal{I}_{\text{coll}} & (3.10) \\ \tau + J(\mathbf{q})^T \mathbf{f} = \mathbf{g}(\mathbf{q}) & (3.11) \\ s(\mathbf{q}, \mathbf{f}) \leq 0 & (3.12) \\ g_i(\mathbf{q}, \mathbf{f}, \tau, \mathbf{v}) = 0 \quad \forall\ T_i & (3.13) \\ h_i(\mathbf{q}, \mathbf{f}, \tau, \mathbf{v}) \leq 0 \quad \forall\ T_i & (3.14) \end{cases}$$

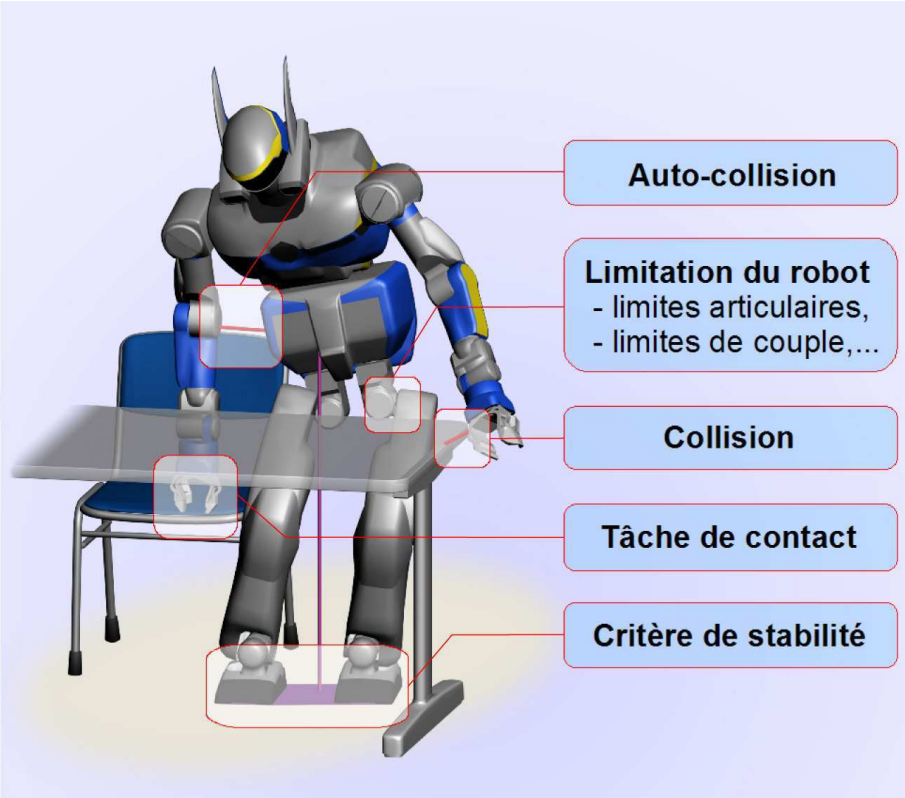


FIG. *3.1 – Les différentes contraintes à respecter par le robot dans le cadre de la génération de posture pour la projection sur un sous-espace de contact : limitations physiques du robot (voir aussi 3.4.5), évitement de collision et auto-collision (3.4.2 et 3.4.3), contacts (3.4.1) et stabilité (3.4.4 et 3.4.5).*

Les équations (3.8) traduisent les limites sur les couples. Couples et forces sont reliés par les équations de la dynamique $M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \tau + J(\mathbf{q})^T\mathbf{f}$ écrites dans le cas statiques (eq. (3.11)). Enfin, s est un critère de stabilité basé sur les forces externes appliquées au robot, comme par exemple détaillé en 3.4.5.

$\mathcal{Q}$ (ou $\mathcal{Q}'$) définit par ses égalités une sous-variété de $\mathcal{C}$ à condition que celles-ci ne soient pas contradictoires (auquel cas, $\mathcal{Q}$ est vide). $\mathcal{Q}$ est un sous-ensemble de cette sous-variété du fait des inégalités. Une fois encore, ce sous-ensemble peut être vide. Tel serait le cas d'un robot qui devrait avoir un unique contact ne pouvant assurer sa stabilité. Si toutefois $\mathcal{Q}$ n'est pas vide, il n'est généralement pas réduit à un point, et en compte alors une infinité indénombrable (cf Fig 3.2). Il peut être parfois souhaitable de se donner un critère pour choisir parmi cet ensemble. On écrit alors le problème d'optimisation suivant :

$$\min obj(\mathbf{q}) \tag{3.15}$$
$$\mathbf{q} \in \mathcal{Q} \tag{3.16}$$

ou plus généralement

$$\min obj(\mathbf{q}, \mathbf{f}, \tau, \mathbf{v}) \tag{3.17}$$
$$(\mathbf{q}, \mathbf{f}, \tau, \mathbf{v}) \in \mathcal{Q}' \tag{3.18}$$

Le choix du critère *obj* est discuté au paragraphe 3.3.

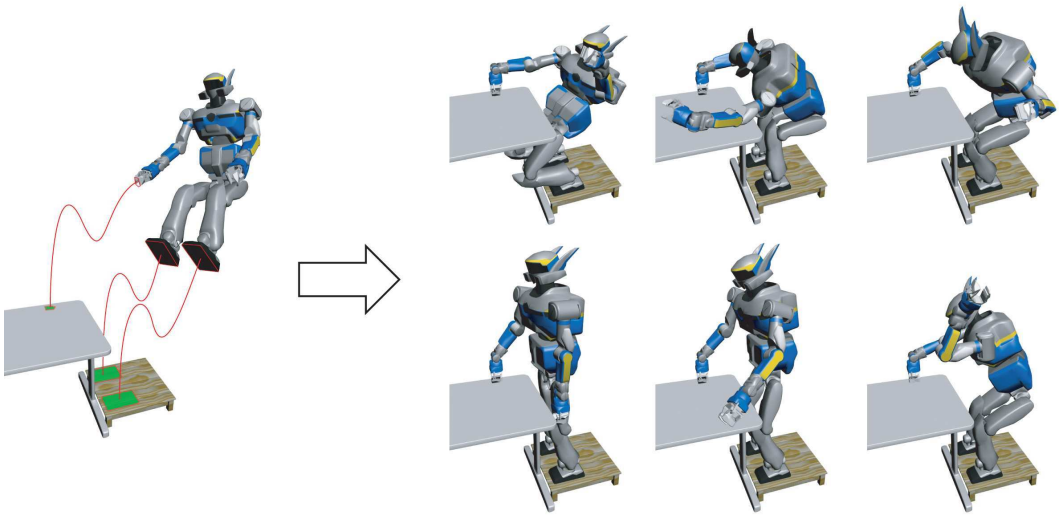


FIG. *3.2 – Pour un même problème $\mathcal{Q}$, ici illustré à gauche pour 3 tâches de contact, si il existe une solution, il en existe généralement une infinité indénombrable. Quelques unes sont données sur la partie droite de la figure.*

De par ses égalités, une tâche définit une variété sur laquelle le robot doit rester. En ce sens on est très proche de la notion de tâche définie par Samson *et al.* [Samson 91]. La différence dans notre cas est que la "régulation" de la tâche est laissée au générateur de posture que nous introduisons ici, ainsi que la présence possible d'inégalités qui restreignent la variété à un de ses sous-ensembles. Il est néanmoins intéressant de garder le parallèle à l'esprit et notamment cela induit une traduction aisée du problème $\mathcal{Q}$ en un problème de commande pour une pile de tâches ([Mansard 06]), afin de jouer de manière robuste aux erreurs de modélisation une posture obtenue sur robot réel.

3.2 Implémentation

3.2.1 Algorithme d'optimisation

Du fait de la présence (possible) de liaisons pivot sur le robot, la géométrie de celui-ci dépend d'angles, ce qui implique qu'au moins une partie des contraintes sont non-linéaires ou même non quadratiques. Parmi les différentes méthodes d'optimisation existantes (cf [Bonnans 03]) pour résoudre un problème avec de telles contraintes, nous avons choisi FSQP ([Lawrence 97])
FSQP est un programme d'optimisation de la famille des SQPs (pour Sequential Quadratic Programming). Il permet la minimisation d'une fonction (ou du maximum d'un ensemble de fonctions) non linéaire sous contraintes possiblement non linéaires :

$$\min \max obj_i(\mathbf{x}) \tag{3.19}$$
$$subj.\ to \quad \begin{cases} x_i^- \leq x_i \leq x_i^+ & \\ h_i(\mathbf{x}) \leq 0 & \text{(inégalités non linéaires)} \\ \langle \mathbf{c_i}, \mathbf{x} \rangle - d_i \leq 0 & \text{(inégalités linéaires)} \\ g_i(\mathbf{x}) = 0 & \text{(égalités non linéaires)} \\ \langle \mathbf{a_i}, \mathbf{x} \rangle - b_i = 0 & \text{(égalités linéaires)} \end{cases}$$

La résolution se fait en deux grandes étapes :

1. recherche d'un point admissible pour toutes les contraintes linéaires et les contraintes d'inégalité non linéaires,
2. satisfaction des contraintes d'égalité non linéaires et optimisation du critère.

La première étape est immédiate si le point fourni en entrée à l'algorithme vérifie déjà les contraintes. Si ce n'est pas le cas, elle se déroule elle même en deux phases : (a) recherche d'un point admissible $\mathbf{x}_0^l$ pour les contraintes linéaires, (b) recherche d'un point admissible $\mathbf{x}_0$ pour toutes les contraintes sauf celle d'égalité non linéaires.
(a) se fait par la résolution d'un simple problème quadratique (QP) (résoudre un LP suffirait, mais en l'absence de critère cela revient au même et FSQP ne comprend pas de routine pour un LP). (b) requiert la résolution d'un problème de minimisation comme 3.19, en prenant les inégalités non linéaires comme fonctions objectifs, et en ignorant les égalités non-linéaires :

$$\min \max h_i(\mathbf{x})$$
$$subj.\ to \quad \begin{cases} x_i^- \leq x_i \leq x_i^+ & \\ \langle \mathbf{c_i}, \mathbf{x} \rangle - d_i \leq 0 & \text{(inégalités linéaires)} \\ \langle \mathbf{a_i}, \mathbf{x} \rangle - b_i = 0 & \text{(égalités linéaires)} \end{cases}$$

La résolution de ce système se fait en partant de $\mathbf{x}_0^l$ et en utilisant le même schéma d'optimisation que celui utilisé pour la seconde étape. La simplification ici est due à la présence de contraintes linéaires uniquement. Si le critère $\max h_i(\mathbf{x})$ de ce problème transformé est inférieur ou égal à 0, alors tous les $h_i(\mathbf{x}_0)$ sont inférieurs ou égaux à 0, et $\mathbf{x}_0$ est donc un point admissible pour les contraintes linéaires et celles d'inégalités non linéaires.
Lors de la deuxième étape les contraintes d'égalité non linéaires sont résolues en même temps que le critère est optimisé. Cela se fait en transformant le critère. Tout d'abord les contraintes d'égalité non linéaires sont éventuellement transformées :

$$g_i' = \begin{cases} g_i \quad \text{si } g_i(\mathbf{x_0}) \leq 0 & (3.20) \\ -g_i \quad \text{si } g_i(\mathbf{x_0}) > 0 & (3.21) \end{cases}$$

puis on définit le nouveau critère $obj' = obj - \sum p_i g_i'$ où les p_i sont des paramètres positifs (multiplicateurs de Lagrange) qui seront itérativement modifiés et obj est le critère initial. Les contraintes d'égalité $g_i'(\mathbf{x}) = 0$ sont alors transformées en contraintes d'inégalité (non

linéaires) $g'_i(\mathbf{x}) \leq 0$ et le problème à optimiser devient

$$\min obj' \quad subj.\ to \quad \begin{cases} x_i^- \leq x_i \leq x_i^+ \\ h_i(\mathbf{x}) \leq 0 \\ g'_i(\mathbf{x}) \leq 0 \\ \langle \mathbf{c_i}, \mathbf{x} \rangle - d_i \leq 0 \\ \langle \mathbf{a_i}, \mathbf{x} \rangle - b_i = 0 \end{cases}$$

qui ne fait plus intervenir que des contraintes linéaires et d'inégalités non linéaires. Ce problème est résolu de manière itérative. A chaque pas, FSQP approxime le problème autour du point courant par un problème quadratique sous contraintes linéaires. La résolution de ce problème donne une direction de recherche pour le problème global, et une recherche de pas est fait le long de cette direction. C'est cette résolution séquentielle de problèmes quadratiques qui donne son nom à l'algorithme. Parallèlement, les poids p_i sont augmentés si nécessaire, pour forcer la solution à se rapprocher des contraintes d'égalité non linéaires, jusqu'à ce $g'(\mathbf{x_k}) \geq -\epsilon_v$ où ϵ_v est la violation des contraintes d'égalité permise par l'utilisateur.

Avoir une idée du fonctionnement de ces deux étapes est important pour la construction des tâches $\mathcal{T}_i$ et leur utilisation, en particulier pour pouvoir détecter rapidement des ensembles de tâches incompatibles. En effet, la première étape étant une version simplifiée du problème, elle est généralement résolue rapidement, mais surtout elle échoue rapidement dans la très grande majorité des cas. Une construction adéquate des tâches permet alors de gagner du temps quand il n'y a pas de solution. Nous tirerons parti de cette remarque lors de la planification de nouveaux contacts.

Les différentes raisons d'échecs de FSQP sont les suivantes :

- lors de la première étape
  1. impossibilité de trouver un point initial $\mathbf{x}_0^l$ satisfaisant les contraintes linéaires. Cet échec est quasi immédiat puisque ne nécessitant que la résolution d'un QP,
  2. impossibilité de trouver un point initial $\mathbf{x}_0$ pour les contraintes linéaires et d'inégalités linéaires. Généralement rapide, sauf problème de stabilité numérique ou quand les contraintes ne sont que très légèrement incompatibles,
- lors de la seconde étape
  1. nombre d'itérations maximum atteint (ce nombre est fixé par l'utilisateur). C'est généralement le cas le plus coûteux en temps. Les contraintes sont difficiles à résoudre, mais l'algorithme arrive cependant à avancer, demandant de nombreuses évaluations des fonctions et de leurs gradients,
  2. impossibilité de trouver une direction de recherche. Outre des problèmes numériques dans la résolution du QP, cela est souvent dû à une incompatibilité entre des contraintes égalités non linéaires et les autres contraintes. Ce cas s'observe par exemple quand deux contacts forcent des corps du robot à être en collision. Si l'incompatibilité est évidente, l'échec est souvent rapide. Dans les autres cas, la durée avant sa manifestation peut être très longue,
  3. impossibilité d'avancer le long de la direction de recherche,
  4. l'algorithme a réussi à avancer le long de la direction de recherche, mais la nouvelle itération est numériquement semblable à la précédente : l'algorithme n'avance pas dans le problème,

5. les paramètres de pénalité p_i ont atteint une valeur maximale (considérée comme l'infini) sans que le problème ne soit résolu : l'algorithme n'arrive pas à satisfaire les contraintes d'égalité non linéaires. Cet échec est parfois coûteux.

De manière générale, on retiendra que l'échec est moins coûteux quand il arrive en première phase. Sauf à avoir un critère mal construit, il est généralement dû à une incompatibilité des contraintes (i.e., $\mathcal{Q}$ est vide dans le cas de notre problème de génération de posture). Il sera donc par exemple judicieux dans certains cas de rajouter des contraintes inégalité autour des contraintes d'égalité non linéaires de la façon suivante : pour une contrainte d'égalité $g_i(\mathbf{x}) = 0$, on construit les contraintes $g_i^+(\mathbf{x}) - m^+ \leq 0$ et $-g_i^-(\mathbf{x}) + m^- \leq 0$. Si celles-ci sont vérifiables, on obtiendra lors de la première étape un point proche de la solution. Dans le cas contraire, on a un échec plus rapide que celui qui interviendrait lors de la seconde étape.

Pour la résolution d'un QP, FSQP a besoin qu'on lui fournisse la valeur de la fonction objectif et des contraintes aux points qu'il demande ainsi que leurs gradients. Il se sert aussi du Hessien, mais le calcule de manière itérative lui-même (méthode BFGS [Broyden 70] [Fletcher 70] [Goldfarb 70] [Shanno 70]). Les preuves de convergence et de rapidité de convergence de l'algorithme sont faites sous l'hypothèse que les fonctions objectif et contraintes sont C^∞. Dans la pratique cependant, des fonctions C^1 suffisent.
Il faut donc veiller à définir des contraintes à l'aide de fonctions f vérifiant les points suivants :

- valant 0 uniquement sur la contrainte,
- gradient continu : $\forall i, \quad \frac{\partial f}{\partial x_i}$ est une fonction continue,
- gradient non nul quand la contrainte l'est. Dans le cas contraire, l'algorithme aura du mal à passer du bon côté d'une contrainte inégalité ou à satisfaire une contrainte égalité, ne sachant pas dans quelle direction avancer au voisinage de la contrainte.

En pratique, la contrainte sur le gradient continu peut être relâchée si on est sûr que les discontinuités arrivent loin des régions intéressantes, c'est-à-dire soit là où la fonction s'annule soit là où l'optimum a lieu, et pourvu qu'elles n'impliquent pas un changement de sens radical du gradient.

3.2.2 Architecture globale du générateur de posture

D'un point de vu informatique, la résolution d'un problème d'optimisation se fait classiquement par le biais d'un *couple optimisateur-simulateur*. L'optimisateur se charge de la partie résolution numérique, quand le simulateur calcule la valeur du critère ou des contraintes pour chaque point choisi par l'optimisateur. Il y a un fort découplage, au sens où l'optimisateur n'a aucune idée du type de problème qu'il résout et le simulateur n'a aucune notion d'optimisation. L'échange entre les deux se réduit à la demande par l'optimisateur de la valeur d'une fonction donnée (le critère, la $i^{ème}$ contrainte, ou un gradient par exemple) en un point, et à la réponse de l'optimisateur à cette requête. Un certain nombre de présupposés sont établis sur ce que chacun doit attendre de l'autre, comme par exemple le niveau de continuité des fonctions ou le bon conditionnement du problème.
Nous conservons cette approche dans la génération de posture (cf Fig. 3.3). Notre optimisateur ici est FSQP, que nous encapsulons dans une couche objet OFSQP. Le simulateur quant à lui se nomme PostureCalculator dans notre architecture et rassemble l'ensemble des contraintes ainsi que le critère, qui sont des fonctions se basant sur des calculs de géométrie directe du robot. Nous détaillons les différents niveaux de la partie simulation.
Robot : c'est une classe en charge des calculs liés à la géométrie directe du robot ou à son

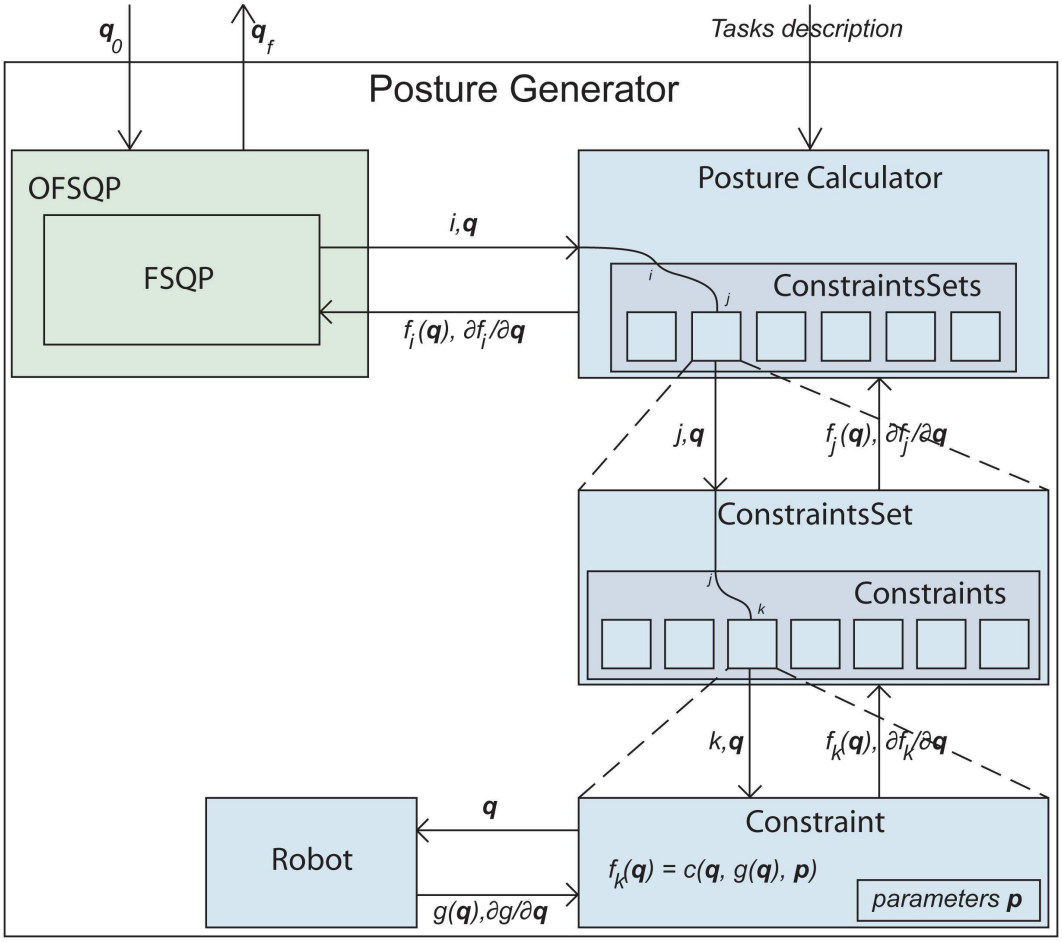


FIG. *3.3 – Architecture du générateur de posture. L'optimisation se fait par un dialogue entre FSQP et le Posture Calculator, dans lequel l'optimisateur (en vert) demande l'évaluation d'une fonction (contrainte ou objectif), ou de son gradient, et reçoit une réponse du simulateur (en bleu). La demande est transmise à la base de la hiérarchie à la contrainte, qui repose sur les calculs de géométrie directe faits par le robot et fournis via la fonction g.*

gradient. On peut par exemple lui demander les coordonnées dans le repère absolu d'un point dont on connaît les coordonnés fixes dans le repère d'un corps, le gradient de ce point, ou encore la matrice de transformation entre deux corps en vu d'un calcul de distance entre eux. Elle est muni d'un système de drapeaux pour éviter de calculer plusieurs fois les mêmes données pour un même $\mathbf{q}$. Ces drapeaux sont remis à 0 lorsque $\mathbf{q}$ change. Le calcul de la géométrie directe et de son gradient est discuté dans l'annexe A et le paragraphe 3.2.3.

Constraint : une contrainte est une fonction de $\mathbb{R}^m$ dans $\mathbb{R}$ où m est le nombre de variables du problème ($n + 6$ dans le cadre du problème $\mathcal{Q}$, plus dans le cadre de $\mathcal{Q}'$) ainsi que le choix entre égalité ou inégalité. Un objet *Constraint* dans notre cas doit pouvoir renvoyer la valeur de la fonction qu'il implémente pour un point $\mathbf{q}$, ainsi que son gradient. La fonction implémentée est choisie parmi une librairie de fonctions et spécifiée par une série de paramètres. Par exemple, la contrainte de non collision entre le corps i et le corps j correspond à une fonction $a \cdot d(i, j, \mathbf{q}) + b$ dont les paramètres sont a, b, i et j.

ConstraintSet : un groupe de contraintes. Typiquement, un contact, ou plus généralement une

tâche, est un *ConstraintSet*, mais c'est aussi le cas du critère de stabilité. Cet ensemble 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 spécifier l'emplacement de contact prédéfini du robot par son identifiant et de donner le repère de l'environnement définissant le contact, et le *ConstraintSet* de contact génère automatiquement les 8 contraintes définissant le contact (cf § 3.4.1).
PostureCalculator : le simulateur. Il regroupe tous les ensembles de contraintes et maintient lui-même à jour celui de la stabilité. Il a aussi la charge de trier les contraintes dans l'ordre demandé par FSQP (i.e. contraintes d'inégalité non-linéaires, d'égalité non-linéaires, d'inégalité linéaires et d'égalité linéaires).

3.2.3 Gradient de la géométrie directe

Le calcul de la géométrie directe se fait de manière très classique. Quelques détails, ainsi que des façons d'accélérer les calculs sont donnés dans l'annexe A. Le calcul de son gradient nécessite une attention particulière pour obtenir des calculs rapides. Soit P un point de coordonnées $(x, y, z)^T$ dans le corps i du robot. Si f est une fonction de P, $\frac{\partial f}{\partial \mathbf{q}}$ fait intervenir, par composition $\frac{\partial \mathbf{P}}{\partial \mathbf{q}}$:

$$\frac{\partial f(\mathbf{P})}{\partial \mathbf{q}} = \frac{\partial f(\mathbf{P})}{\partial \mathbf{P}} \frac{\partial \mathbf{P}}{\partial \mathbf{q}} \tag{3.22}$$

Les coordonnées de $\mathbf{P}$ dans le repère absolu sont des fonctions de $\mathbf{q}$, fixées par la structure cinématique du robot, si bien qu'il est aisé de calculer $\frac{\partial \mathbf{P}}{\partial \mathbf{q}}$. Cependant cette approche n'est pas flexible, or il est nécessaire de pouvoir calculer $\frac{\partial \mathbf{P}}{\partial \mathbf{q}}$ pour un point P quelconque lié au robot. Dans un souci de généricité et de vitesse de calcul, il est nécessaire de distinguer quels sont les calculs qui sont indépendants des coordonnées de P et ceux qui ne le sont pas. On procède comme suit : les coordonnées de P dans le repère absolu sont $M^i(x, y, z, 1)^T = R^i(x, y, z)^T + \mathbf{t}^i + \mathbf{t}^i_0 = x \cdot \mathbf{C}^i_1 + y \cdot \mathbf{C}^i_2 + z \cdot \mathbf{C}^i_3 + \mathbf{t}^i + \mathbf{t}^i_0$ où $\mathbf{C}^i_1$, $\mathbf{C}^i_2$ et $\mathbf{C}^i_3$ sont les colonnes de R^i. On a alors :

$$\frac{\partial \mathbf{P}}{\partial \mathbf{q}} = x \cdot J^i_1 + y \cdot J^i_2 + z \cdot J^i_3 + J^i_4 \tag{3.23}$$

avec

$$J^i_j = \frac{\partial \mathbf{C}^i_j}{\partial \mathbf{q}}, \quad j = 1, 2, 3 \tag{3.24}$$

$$J^i_4 = \frac{\partial \mathbf{t}^i}{\partial \mathbf{q}} \tag{3.25}$$

Les J^i_j sont des matrices de taille $3 \times (n + 6)$ que nous nommons *matrices de pré-gradient*. Elles ne dépendent que de $\mathbf{q}$ et du corps, mais sont totalement indépendantes du point P, ce qui permet d'avoir une forme générique du gradient d'un point. Ces matrices peuvent être pré-calculées de façon à générer un code optimisé (en particulier en prenant en compte les nombreux 0 qui y apparaissent).
FSQP faisant appel à tous les gradients à chaque itération, nous avons opté pour le calcul de toutes ces matrices ensemble, dans une seule et même fonction, ce qui permet de factoriser grandement les calculs.

Ces matrices ne comprennent des éléments non nuls que sur les colonnes dont les indices correspondent à ceux des articulations dans la chaîne cinématique menant au corps i. Elles

ont donc un grand nombre d'éléments nuls et on tire parti de cela dans le calcul de $\frac{\partial \mathbf{P}}{\partial \mathbf{q}}$ en écrivant une fonction spécifique pour chaque corps.

Si au lieu d'un point P, on s'intéresse à un vecteur $\mathbf{v}$ de coordonnées $(v_x, v_y, v_z)^T$ dans le corps i, la démarche est la même, mais ne fait pas intervenir $\mathbf{t}^i$ et $\mathbf{t}_0^i$ et donc J_4^i n'apparaît pas :

$$\frac{\partial \mathbf{v}}{\partial \mathbf{q}} = v_x \cdot J_1^i + v_y \cdot J_2^i + v_z \cdot J_3^i \tag{3.26}$$

3.3 Critères

Le critère d'optimisation utilisé permet de faire le choix d'une configuration dans le sous-espace solution de l'espace de configuration lorsque celui-ci est non vide et non trivial. Ce critère peut être n'importe quelle fonction au moins deux fois continue au choix de l'utilisateur. Néanmoins pour obtenir une bonne convergence, on préférera utiliser des fonctions convexes (au moins dans la partie d'espace utilisée). Si la fonction n'est pas convexe, il y a des risques que l'optimisation finisse dans un minimum local.

Le choix du critère est dépendant de l'utilisation faite du générateur de posture.
Si l'on ne s'en sert que 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 l'application implique de faire adopter les postures trouvées par un robot réel, on pourra préférer maximiser une marge de stabilité. Par exemple on cherchera à minimiser la distance entre la projection du centre de gravité et le centre du plus grand cercle inscrit dans le polygone de sustentation dans les cas de contact simple (cf § 3.4.4) ou celle entre le torseur des forces de gravité appliquées au CdM et le centre de la plus grande hypersphère inscrite dans la région de stabilité présentée § 3.4.5.
Si l'on s'attache à l'apparence du robot, il est classique de rechercher une posture ressemblant à celle d'un humain. De nombreux critères ont été proposés pour cela. On pourra entre autres citer la minimisation des couples qui donne de bons résultats, ou des notions de confort de la posture (voir par exemple [Yang 04]). Plus simple et plus rapide, la distance à une posture de référence $\mathbf{q}^0$ sous la forme $obj = \sum (q_i - q_i^0)^2$ est le critère que nous adopterons en premier lieu dans ce travail, avant de le raffiner sous forme d'une variante de la distance à un chemin dans l'espace de configuration :

$$obj = \min_s \sum_{i=0}^{n+6} (q_i - p_i(s))^2 \tag{3.27}$$

où $s \in [0,1] \rightarrow \mathbf{p}(s) = (p_1(s), \cdots, p_{n+6}(s))^T \in \mathcal{C}$ est le dit chemin.
Enfin on pourra encore citer les cas où l'on veut maximiser l'information donnée par un capteur du robot. Par exemple, on souhaite que le robot prenne une pose qui lui permette d'observer via ses caméras le plus possible d'environnement inconnu ou de surface inconnue d'un objet. Ce dernier cas est détaillé au paragraphe 3.6.2.2.

3.4 Contraintes

3.4.1 Contacts

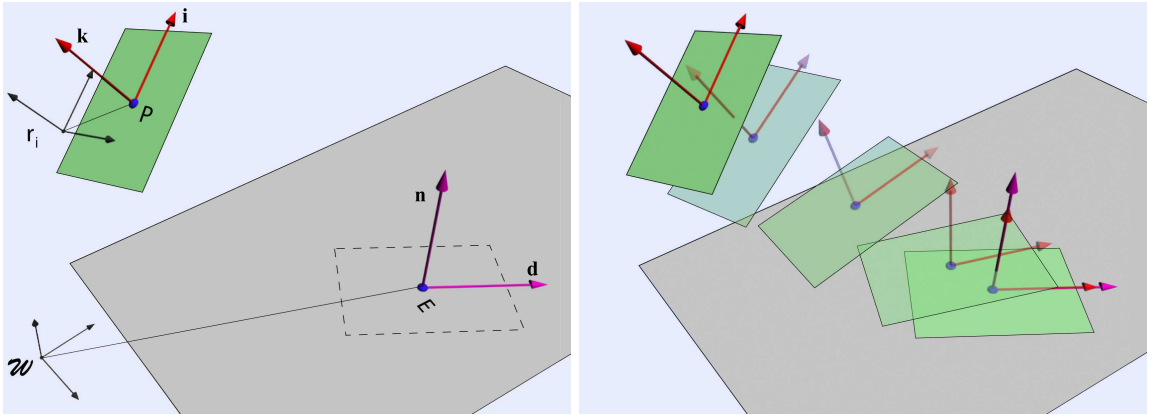


FIG. 3.4 – *Spécification d'un contact par la mise en correspondance d'un repère du robot avec un repère de l'environnement.*

Le type de contact que nous considérons principalement dans ce travail, est le contact que nous nommons **fort** dans ce travail et qui correspond à la donnée complète de la position et l'orientation d'un corps r_i du robot : il ne permet pas de mouvement relatif entre r_i et O_k, l'objet en contact. Cette immobilisation de r_i dans $\mathcal{W}$ se définit par 6 égalités correspondant à la privation des 6 degrés de liberté possible de r_i.
Soit P un point de la surface de r_i de coordonnées $\mathbf{P}$ dans r_i et E un point de la surface de l'environnement de coordonnées absolues $\mathbf{E}$ (cf Fig. 3.4). A supposer que ni P, ni E ne soit sur une partie non différentiable de sa surface de support (comme un coin ou une arrête), on peut définir un vecteur normal en chacun de ces points. Soient $\mathbf{n}_P$ et $\mathbf{n}_E$ les vecteurs normaux à ces deux points, exprimés dans leur repère respectif, $\mathbf{n}_P$ pointant vers l'intérieur de r_i et $\mathbf{n}_E$ vers l'extérieur de l'objet auquel il est attaché. La contrainte tridimensionnelle $\mathbf{P}^{\mathcal{W}} = \mathbf{E}$, où $\mathbf{P}^{\mathcal{W}}$ est le vecteur de coordonnées de P dans le repère absolu, empêche les mouvements de translation de r_i. Contraindre $\mathbf{n}_P$ et $\mathbf{n}_E$ à être colinéaires de même sens retire deux degrés de rotation, ne laissant plus à r_i que la possibilité de pivoter autour des normales. La définition complète du contact fort demande donc aussi d'imposer un angle d'orientation relatif entre les deux objets. Pour cela, on doit définir $\mathbf{d}_P$ et $\mathbf{d}_E$ orthogonaux respectivement à $\mathbf{n}_P$ et $\mathbf{n}_E$, et on fixe l'angle entre ces deux vecteurs.

En pratique, les contacts se font pour des points du robot définis à l'avance. Pour un tel point P, $\mathbf{n}_P$ est parfaitement défini, et on fixe $\mathbf{d}_P$. L'orientation du contact se définit alors par le choix de $\mathbf{d}_E$ perpendiculaire à $\mathbf{n}_E$, en prenant pour contrainte que $\mathbf{d}_P$ et $\mathbf{d}_E$ doivent être colinéaires et de même sens. Si on définit $\mathbf{b}_P = \mathbf{n}_P \wedge \mathbf{d}_P$, et $\mathbf{b}_E = \mathbf{n}_E \wedge \mathbf{d}_E$, un contact fort peut être interprété comme la mise en correspondance des repères $(P, \mathbf{d}_P, \mathbf{b}_P, \mathbf{n}_P)$ et

$(E, \mathbf{d}_E, \mathbf{b}_E, \mathbf{n}_E)$. L'écriture des contraintes se fait comme suit :

$$\begin{aligned} P_x^{\mathcal{W}} - E_x &= 0 && (3.28)\\ P_y^{\mathcal{W}} - E_y &= 0 && (3.29)\\ P_z^{\mathcal{W}} - E_z &= 0 && (3.30)\\ \mathbf{b}_P^{\mathcal{W}}.\mathbf{d}_E &= 0 && (3.31)\\ \mathbf{n}_P^{\mathcal{W}}.\mathbf{d}_E &= 0 && (3.32)\\ \mathbf{n}_P^{\mathcal{W}}.\mathbf{b}_E &= 0 && (3.33)\\ -\mathbf{d}_P^{\mathcal{W}}.\mathbf{d}_E &\leq 0 && (3.34)\\ -\mathbf{b}_P^{\mathcal{W}}.\mathbf{b}_E &\leq 0 && (3.35) \end{aligned}$$

Les égalités avec produits scalaires forcent les colinéarités évoquées ci-dessus quand les deux inégalités expriment le "de même sens". $\mathbf{P}^{\mathcal{W}}$ étant relié à $\mathbf{P}$ par la transformation $M^i(q)$, tout comme $\mathbf{d}_P^{\mathcal{W}}$, $\mathbf{b}_P^{\mathcal{W}}$ et $\mathbf{n}_P^{\mathcal{W}}$ le sont à $\mathbf{d}_P$, $\mathbf{b}_P$ et $\mathbf{n}_P$ respectivement, toutes ces contraintes sont non-linéaires.

Il existe plusieurs variations autour de ce contact fort qui correspondent à divers niveau de relaxation des contraintes sur r_i (voir aussi [Khalil 02], chap. 6, pour une description détaillé des diverses tâches de contact) .

Tout d'abord, le contact **léger** pour lequel on laisse la liberté à r_i de se mouvoir le long de la surface de contact, et de tourner autour de la normale de contact : P est seulement contraint a se trouver à une distance nulle de la surface, et les normales doivent être colinéaires et de même sens. Si la surface de l'environnement est plane, on peut lui associer pour normale sortante $\mathbf{n}_E$ et la tâche de contact devient :

$$\begin{aligned} \left(\mathbf{P}^{\mathcal{W}} - \mathbf{E}\right).\mathbf{n}_E &= 0 && (3.36)\\ \mathbf{d}_P^{\mathcal{W}}.\mathbf{n}_E &= 0 && (3.37)\\ \mathbf{b}_P^{\mathcal{W}}.\mathbf{n}_E &= 0 && (3.38)\\ -\mathbf{n}_P^{\mathcal{W}}.\mathbf{n}_E &\leq 0 && (3.39) \end{aligned}$$

Il reste à ce contact deux degrés de liberté en translation et un en rotation.

Si une des surfaces de contact est déformable, on peut vouloir être moins strict sur les angles, c'est-à-dire laisser une marge sur les colinéarités de vecteurs. On remplace alors certaines des contraintes de perpendicularité $\mathbf{u}.\mathbf{v} = 0$ par des contraintes de quasi-colinéarité $\mathbf{u}.\mathbf{v}' \geq a$ où a est une constante dépendante de la norme des vecteurs et de l'angle maximal autorisé entre les deux vecteurs.

Enfin, pour tirer parti des remarques sur la rapidité d'échec de FSQP dans la phase de résolution des contraintes d'inégalité non linéaires, on est aussi amené à utiliser une tâche de **pré-contact** qui vérifie la possibilité d'atteindre la surface de l'environnement :

$$\begin{aligned} \left(\mathbf{P}^{\mathcal{W}} - \mathbf{E}\right).\mathbf{n}_E &\leq d && (3.40)\\ \mathbf{d}_P^{\mathcal{W}}.\mathbf{n}_E &= 0 && (3.41)\\ \mathbf{b}_P^{\mathcal{W}}.\mathbf{n}_E &= 0 && (3.42)\\ -\mathbf{n}_P^{\mathcal{W}}.\mathbf{n}_E &\leq 0 && (3.43) \end{aligned}$$

d étant une distance au plan de contact sous laquelle on veut savoir si on peut amener le point P du robot.

3.4.2 Contraintes de collision

Pour deux objets quelconques O_1 et O_2, qu'ils soient corps du robot ou parties de l'environnement, on souhaite pouvoir écrire une contrainte g, telle que

$$\begin{cases} g(\mathbf{q}) < 0 & \text{si } O_1(\mathbf{q}) \cap O_2(\mathbf{q}) = \emptyset \quad (3.44) \\ g(\mathbf{q}) = 0 & \text{si } O_1(\mathbf{q}) \cap O_2(\mathbf{q}) = \partial O_1(\mathbf{q}) \cap \partial O_2(\mathbf{q}) \quad (3.45) \\ g(\mathbf{q}) > 0 & \text{sinon} \quad (3.46) \end{cases}$$

et vérifiant les propriétés nécessaires, exposées à la fin du paragraphe 3.2.1. Au signe près, une fonction de distance ou pseudo-distance **signée** remplit ces critères, à l'exception notable de la continuité du gradient. Nous exposons dans le chapitre 4 une façon de modifier légèrement les objets qui permet d'utiliser la distance euclidienne d classique comme base pour les contraintes de collision. Cette distance est signée, dans le sens où 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 :

$$\epsilon - d(O_1, O_2) \leq 0 \quad (3.47)$$

où ϵ est une marge de sécurité.

Le calcul du gradient d'une telle contrainte mérite quelques explications (voir aussi [Lefebvre 05]) :
on note $P_1(\mathbf{q})$ et $P_2(\mathbf{q})$ les points témoins de la distance, c'est-à-dire les points de $O_1(\mathbf{q})$ et $O_2(\mathbf{q})$ respectivement pour lesquels la distance est réalisée : $d(O_1(\mathbf{q}), O_2(\mathbf{q})) = \|P_1(\mathbf{q})P_2(\mathbf{q})\|$. On note plus simplement $d(\mathbf{q})$ cette distance. Si le gradient de la distance est continu, alors cette paire de points est unique pour un $\mathbf{q}$ donné. Tel est le cas ici. La difficulté de calcul du gradient de la distance vient de ce que P_1 et P_2 bougent dans O_1 et O_2 lorsque $\mathbf{q}$ varie. On ne peut donc appliquer directement la méthode exposée en 3.2.3 pour le calcul du gradient de ces points, qui entre par composition dans le calcul du gradient de la distance.
A $\mathbf{q}$ 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 :

$$\min_{\mathbf{P}_1, \mathbf{P}_2} \left\| M^1(\mathbf{q})\mathbf{P}_1 - M^2(\mathbf{q})\mathbf{P}_2 \right\| \quad (3.48)$$

dont la condition d'optimalité s'écrit en différentiant le critère

$$\left(M^1(\mathbf{q})\mathbf{P}_1 - M^2(\mathbf{q})\mathbf{P}_2\right)^T \left(M^1(\mathbf{q})\frac{\partial \mathbf{P}_1}{\partial \mathbf{X}} - M^2(\mathbf{q})\frac{\partial \mathbf{P}_2}{\partial \mathbf{X}}\right) = 0 \quad (3.49)$$

soit

$$\mathbf{n}_d^T \left(M^1(\mathbf{q})\frac{\partial \mathbf{P}_1}{\partial \mathbf{X}} - M^2(\mathbf{q})\frac{\partial \mathbf{P}_2}{\partial \mathbf{X}}\right) = 0 \quad (3.50)$$

avec $\mathbf{X} = (\mathbf{P}_1, \mathbf{P}_2)$ et $\mathbf{n}_d$ est le vecteur distance (i.e. formé des deux points témoins) normé et exprimé dans le repère absolu.
Cette condition reste identique dans le cas d'une pénétration, où la distance s'écrit alors sous la forme d'un problème de maximisation. Elle traduit le fait que les mouvements des points témoins relativement aux corps se font perpendiculairement au vecteur distance. Cela permet

une simplification dans le calcul du gradient de la distance :

$$\frac{\partial d}{\partial q_i} = \frac{\sqrt{\|M^1(\mathbf{q})\mathbf{P}_1 - M^2(\mathbf{q})\mathbf{P}_2\|^2}}{\partial q_i} \tag{3.51}$$

$$= \frac{\left(M^1(\mathbf{q})\mathbf{P}_1 - M^2(\mathbf{q})\mathbf{P}_2\right)^T}{\sqrt{\|M^1(\mathbf{q})\mathbf{P}_1 - M^2(\mathbf{q})\mathbf{P}_2\|^2}} \cdot \frac{\partial}{\partial q_i}\left(M^1(\mathbf{q})\mathbf{P}_1 - M^2(\mathbf{q})\mathbf{P}_2\right) \tag{3.52}$$

$$= \mathbf{n}_d^T \left(\frac{\partial M^1(\mathbf{q})}{\partial q_i}\mathbf{P}_1 + M^1(\mathbf{q})\frac{\partial \mathbf{P}_1}{\partial q_i} - \frac{\partial M^2(\mathbf{q})}{\partial q_i}\mathbf{P}_2 - M^2(\mathbf{q})\frac{\partial \mathbf{P}_2}{\partial q_i}\right) \tag{3.53}$$

or

$$\mathbf{n}_d^T \left(M^1\frac{\partial \mathbf{P}_1}{\partial q_i} - M^2\frac{\partial \mathbf{P}_2}{\partial q_i}\right) = \mathbf{n}_d^T \left(M^1\frac{\partial \mathbf{P}_1}{\partial \mathbf{X}} - M^2\frac{\partial \mathbf{P}_2}{\mathbf{X}}\right)\frac{\partial \mathbf{X}}{\partial q_i} = 0 \tag{3.54}$$

d'où

$$\frac{\partial d}{\partial q_i} = \mathbf{n}_d^T \left(\frac{\partial M^1(\mathbf{q})}{\partial q_i}\mathbf{P}_1 - \frac{\partial M^2(\mathbf{q})}{\partial q_i}\mathbf{P}_2\right) \tag{3.55}$$

En considérant P_1^d et P_2^d, les points fixes de O_1 et O_2 qui à $\mathbf{q}$ sont confondus avec P_1 et P_2, et en notant $\mathbf{P}_1^d$ et $\mathbf{P}_2^d$ leurs coordonnées, on peut alors écrire

$$\frac{\partial d}{\partial \mathbf{q}} = \mathbf{n}_d^T \left(\frac{\partial \mathbf{P}_1^d}{\partial \mathbf{q}} - \frac{\partial \mathbf{P}_2^d}{\partial \mathbf{q}}\right) \tag{3.56}$$

qui se calcule grâce au mécanisme exposé en 3.2.3

3.4.3 Contraintes d'auto-collision analytiques

Entre deux corps consécutifs du robot, la contrainte de non collision s'exprime au travers de la limite articulaire, ou si ce n'est pas le cas, cette limite est ajustée pour empêcher une collision. Elle ne peut s'écrire comme une distance positive entre les deux corps, car ceux-ci sont à tout moment imbriqués l'un dans l'autre. Dans le cas d'une liaison composée, qui comprend plusieurs liaisons rotules d'axes concourants et met en jeu plus de deux corps, comme c'est le cas pour une épaule d'humanoïde par exemple, la non-collision entre deux corps non consécutifs ne peut pas non plus s'exprimer comme une contrainte sur la distance, pas plus que se restreindre aux limites articulaires classiques.
Si q_i,...,q_{i+k} sont les valeurs articulaires d'une telle liaison composée (généralement $k = 1$ ou $k = 2$), l'espace produit $Q_{ik} = \left[q_i^-, q_i^+\right] \times ... \times \left[q_{i+k}^-, q_{i+k}^+\right]$ se découpe entre les configurations en collision Q_{ik}^c et celles sans collision Q_{ik}^f pour les corps en jeu dans la liaison composée. Ce découpage se déduit géométriquement dans les cas simples, empiriquement sinon. Dans le cas où un simple test d'appartenance à l'un de ces deux ensembles est nécessaire, l'utilisation d'une look-up table suffit à déterminer s'il y a collision ou non [Okada 06]. Cela ne peut cependant pas être utilisé dans le cadre d'une optimisation. Il faut alors chercher une fonction au moins C^1, $h\left(q_i, ..., q_{i+k}\right)$, strictement positive uniquement sur l'intérieur de Q_{ik}^c et strictement négative uniquement sur celui de Q_{ik}^f. Une telle fonction peut-être difficile à obtenir et il faudra parfois se contenter d'approximations. Celles-ci devront être conservatives.

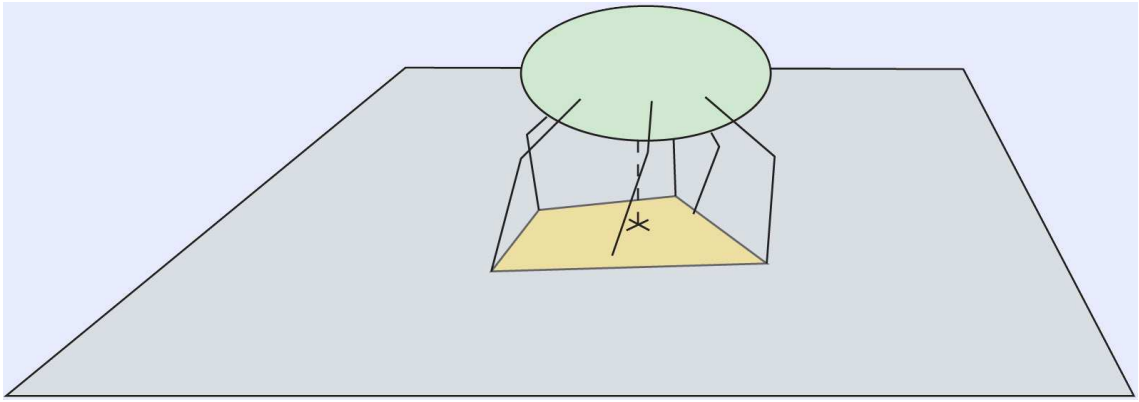


FIG. *3.5 – Lorsque tous les contacts se font sur un même plan horizontal, on a le critère de stabilité suivant : le robot est stable si et seulement si son centre de masse se projette dans l'enveloppe convexe des points de contact.*

3.4.4 Contrainte de stabilité par projection du CdM

Pour être statiquement stable, il faut que le robot vérifie les équations d'Euler-Newton :

$$M\mathbf{g} + \sum \mathbf{f}_i = 0 \tag{3.57}$$

$$M\mathbf{G}(\mathbf{q}) \wedge \mathbf{g} + \sum \mathbf{C}_i \wedge \mathbf{f}_i = 0 \tag{3.58}$$

où $\mathbf{G}(\mathbf{q})$ est la position du centre de gravité du robot, M sa masse, $\mathbf{g}$ le vecteur gravité et les $\mathbf{f}_i$ sont les forces de contact appliquées au robot aux points $\mathbf{C}_i$
En substituant $M\mathbf{g}$ dans 3.58 en utilisant la relation 3.57, on obtient

$$\mathbf{G}(\mathbf{q}) \wedge \sum \mathbf{f}_i = \sum \mathbf{C_i} \wedge \mathbf{f}_i \tag{3.59}$$

On considère que tous les points $\mathbf{C}_i$ sont sur un même plan horizontal $z = h$ et on appelle $\mathbf{G}^h$ le projeté de $\mathbf{G}$ sur ce plan : $\mathbf{G}^h = (x, y, h)^T$. Les deux premières lignes de l'équation précédente se récrivent alors

$$\sum \begin{pmatrix} yf_{i_z} - hf_{i_y} \\ hf_{i_x} - xf_{i_z} \end{pmatrix} = \sum \begin{pmatrix} C_{i_y} f_{i_z} - hf_{i_y} \\ hf_{i_x} - C_{i_x} f_{i_z} \end{pmatrix} \tag{3.60}$$

Les hf_{i_x} et hf_{i_y} se simplifient et en changeant l'ordre des lignes, il apparaît

$$\left(\sum f_{i_z}\right) \mathbf{G}(\mathbf{q})_i^p = \sum f_{i_z} \mathbf{C}_i^p \tag{3.61}$$

où $\mathbf{G}(\mathbf{q})^p$ et les $\mathbf{C}^p$ sont les coordonnées à 2 dimensions des points $\mathbf{G}(\mathbf{q})^h$ et $\mathbf{C}$ dans le plan $z = h$.
Or les contacts étant unilatéraux entre le robot et son environnement, on a la condition $f_{i_z} \geq 0$. En récrivant l'égalité ci-dessus

$$\mathbf{G}(\mathbf{q})^p = \frac{\sum f_{i_z} \mathbf{C}_i^p}{\sum f_{i_z}} \tag{3.62}$$

qui fait apparaître $\mathbf{G}(\mathbf{q})^p$ comme un barycentre à coefficients positifs des $\mathbf{C}_i^p$. On a alors la propriété suivante : un robot dont tous les contacts se font sur le même plan horizontal

est statiquement stable si et seulement si son centre de gravité se projette dans l'enveloppe convexe de ses points de contact. Cette enveloppe s'appelle classiquement le *polygone de sustentation* (cf Fig. 3.5).

Si les contacts se trouvent sur des plans horizontaux différents, ce critère est conservatif : intuitivement, le robot peut se servir des frottements au niveau des contacts pour maintenir la projection de son centre de gravité légèrement en dehors de l'enveloppe convexe des points de contact. On se servira par la suite de ce critère dans tout cas où les contacts se font sur des plans horizontaux.

La détermination des contraintes de stabilité sur la base de ce critère se fait comme suit : à chaque contact prédéfini sur le robot, on associe une enveloppe convexe de la surface de contact par la donnée de ses points dans le repère du corps concerné r_i. Les contacts désirés étant fixés dans l'environnement, on sait où seront positionnés ces points. Pour chaque contact, on peut alors calculer l'intersection de la surface plane qu'ils forment avec celle (supposée polygonale) de l'environnement sur laquelle le contact a lieu. On obtient un polygone dont on garde les sommets C_i. $C_0, \cdots, C_m$ étant l'ensemble de ces points pour la totalité des contacts désirés, on les projette sur un plan horizontal, et seuls sont gardés les $C_0^p, \cdots, C_k^p$ qui font partie de l'enveloppe convexe de cette projection.
On suppose ces points rangés dans l'ordre trigonométrique, et on définit $\mathbf{n}_i = \overrightarrow{C_i^p C_{i+1}^p} \wedge (0,0,1)^T$ pour $i = 0 \cdots k$ avec $C_{k+1} \equiv C_0$. Les contraintes de stabilité s'écrivent alors :

$$\overrightarrow{G^p(\mathbf{q})C_i^p}.\mathbf{n}_i \leq 0 \quad \forall\, i \in [|0,k|] \tag{3.63}$$

Parce qu'il ne fait pas intervenir les efforts internes au robot, ce critère considère ce dernier comme un simple solide ayant de multiples contacts avec l'environnement. En particulier, cela implique que le robot est toujours capable de délivrer des couples suffisants pour rester "solide".

3.4.5 Contrainte de stabilité : existence de forces admissibles

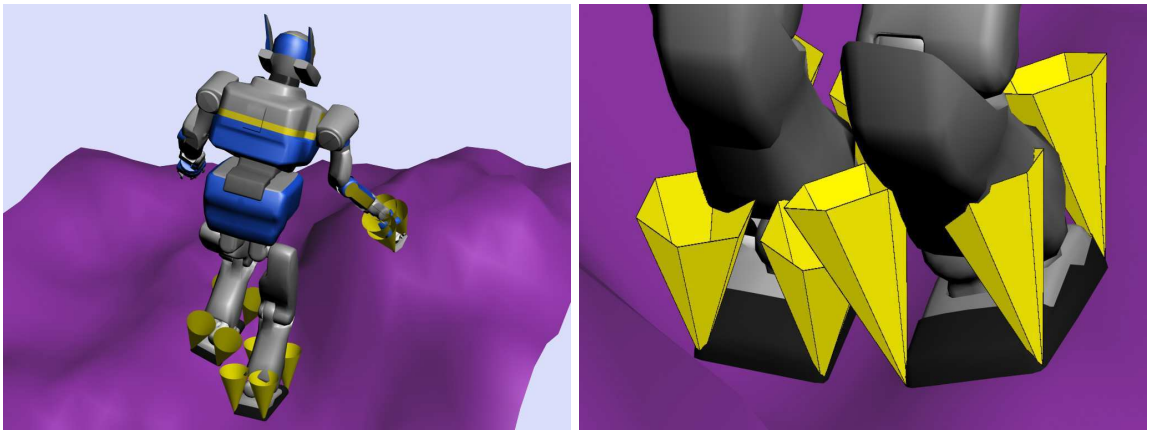


FIG. *3.6 – Quand les contacts ne se situent pas sur un même plan parallèle (comme sur l'image de gauche), il faut prendre un critère plus général que la projection du centre de gravité dans le polygone de sustentation. On peut alors passer par l'existence de forces restant dans les cônes de frottement. Ceux-ci sont parfois discrétisés pour faciliter les calculs (droite).*

Le critère de projection du centre de gravité dans le polygone de sustentation est restrictif quant à son utilisation (cf Fig. 3.6). De manière très générale, la stabilité statique d'un robot répond aux équations suivantes [Wieber 02] :

$$\begin{pmatrix} \mathbf{g}_1(\mathbf{q}) \\ \mathbf{g}_2(\mathbf{q}) \end{pmatrix} = \begin{pmatrix} \tau \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} J_{c_1}^T(\mathbf{q}) \\ J_{c_2}^T \end{pmatrix} \mathbf{f} \tag{3.64}$$

$$\tau \in \mathcal{T} = \times[\tau_i^-, \tau_i^+] \tag{3.65}$$

$$\mathbf{f} \in \mathcal{F} = \times\mathscr{C}_j \tag{3.66}$$

L'équation 3.64 se découpe en deux blocs de n et 6 lignes qui correspondent à la dynamique interne et externe du robot respectivement. $\mathbf{g} = (\mathbf{g}_1^T, \mathbf{g}_2^T)^T$ est le vecteur contenant les effets de la gravité sur le robot, τ est le vecteur des couples articulaires, $\mathbf{f} = \left(\mathbf{f}_1^T, \cdots, \mathbf{f}_k^T\right)^T$ regroupe les forces aux points de contact $C_1, \cdots, C_k$, qui sont définis et fixés dans l'environnement par la donnée des contacts désirés, et $J_c = (J_{c_1}, J_{c_2})$ est la matrice jacobienne des contacts. A une éventuelle multiplication par une matrice près, les 6 dernières lignes sont les équations d'Euler-Newton (3.57 et 3.58).
$\mathcal{T}$ est l'espace des couples admissibles, défini comme l'espace produit des limites sur chaque couple et $\mathcal{F}$ est l'espace des forces admissibles défini comme l'espace produit des cônes de frottements $\mathscr{C}_j$ aux points de contact C_j. Un cône $\mathscr{C}_j$ est défini par les lois de Coulomb : $\mathscr{C}_j = \left\{\mathbf{f}_j \in \mathcal{R}^3 \mid \mu_j f_j^n \geq \left\|\mathbf{f}_j^t\right\|\right\}$, avec f_j^n composante normale de la force et $\mathbf{f}^t$ ses composantes tangentielles, μ_j étant le coefficient de friction pour ce contact.

La question de la stabilité statique du robot devient alors : pour une configuration $\mathbf{q}$ donnée, existe-t-il un jeu de forces $\mathbf{f}$ dans $\mathcal{F}$ et un jeu de couples τ dans $\mathcal{T}$, vérifiant l'équation 3.64 ?
Mathématiquement, cette question peut se ramener à la question de l'appartenance de $\mathbf{g}_2(\mathbf{q})$ à un nouvel ensemble à 6 dimension $\mathcal{G}(\mathbf{q})$. On se passe alors de l'utilisation des variables $\mathbf{f}$ et τ. Cet ensemble se calcule comme suit : des n premières lignes de 3.64 on peut ramener les limites sur τ à des limites sur $\mathbf{f}$ de la forme $A(\mathbf{q})\mathbf{f} + \mathbf{b}(\mathbf{q}) \leq 0$ définissant un ensemble $\mathcal{F}'(\mathbf{q})$ dans l'ensemble des forces, avec

$$A(\mathbf{q}) = \begin{pmatrix} -J_{c_1}^T(\mathbf{q}) \\ J_{c_1}^T(\mathbf{q}) \end{pmatrix} \qquad et \qquad \mathbf{b}(\mathbf{q}) = \begin{pmatrix} \mathbf{g}_1(\mathbf{q}) - \tau^+ \\ \tau^- - \mathbf{g}_1(\mathbf{q}) \end{pmatrix} \tag{3.67}$$

$\mathbf{f}$ doit donc appartenir à $\mathcal{F} \cap \mathcal{F}'(\mathbf{q})$ et vérifier les 6 dernières lignes de 3.64. On construit alors $\mathcal{G}(\mathbf{q}) = J_{c_2}^T\left(\mathcal{F} \cap \mathcal{F}'(\mathbf{q})\right)$. Le problème de cette approche est que le calcul de $\mathcal{G}(\mathbf{q})$ fait intervenir le calcul d'une intersection entre le polytope $\mathcal{F}'(\mathbf{q})$ et un produit de cônes en haute dimension, puis une projection de celle-ci (par $J_{c_2}^T$) (cf Fig. 3.7). Même en discrétisant les cônes, le calcul de l'intersection est trop lourd en pratique, d'autant plus qu'il dépend de $\mathbf{q}$. Pour avoir le critère de stabilité exact, il faudra donc passer par un problème d'optimisation comprenant $\mathbf{f}$ et τ, tel que le problème $\mathcal{Q}'$ exposé en 3.1. Les contraintes de stabilité $s(\mathbf{q}, f(\mathbf{q}))$ s'écrivent alors

$$-f_j^n \leq 0 \tag{3.68}$$

$$\mathbf{f}_j^{t\,2} - \mu_{0_j}^2 f_j^{n\,2} \leq 0 \tag{3.69}$$

$$\mathbf{g}_2(\mathbf{q}) - J_{c_2}^T \mathbf{f} = 0 \tag{3.70}$$

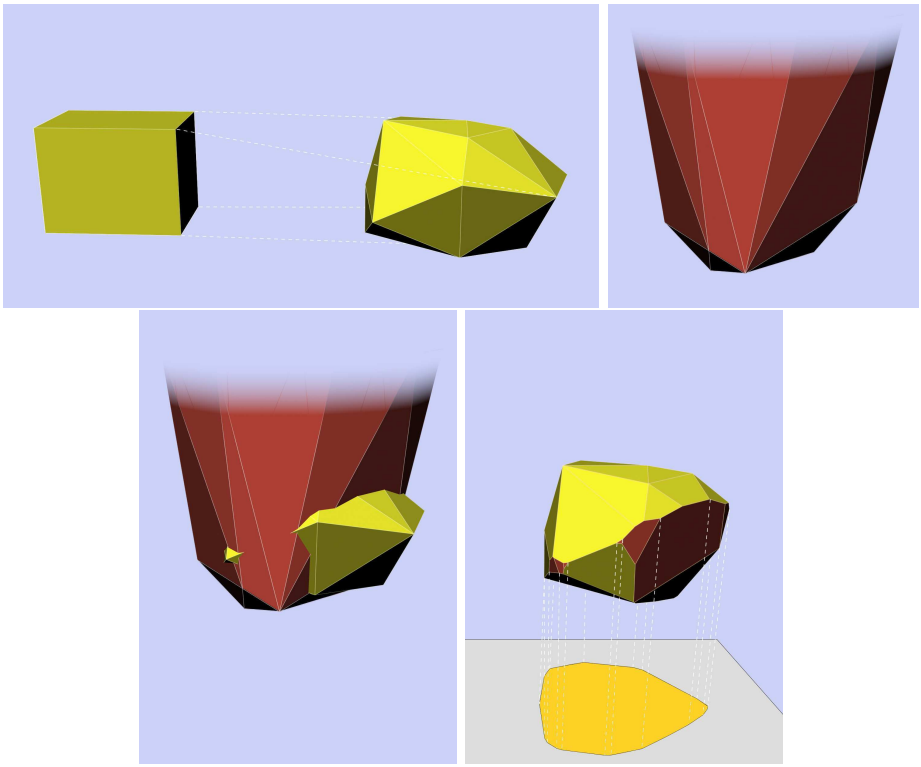


FIG. 3.7 – *L'hyperrectangle à n dimensions des couples admissibles (parallèlépipède rectangle) se projette dans l'espace des forces en un polytope $\mathcal{F}'(\mathbf{q})$ (en jaune). L'intersection de celui-ci avec l'ensemble produit $\mathcal{F}$ des cônes de frottement (en rouge, représenté dans le cas de cônes discrétisés), permet d'obtenir l'ensemble des forces admissibles (polytope rouge et jaune). Cet ensemble est alors projeté par $J_{c_2}^T$ (en orange) dans l'espace des torseurs s'appliquant au centre de gravité du système, et donnant ainsi l'ensemble admissible de ces torseurs $\mathcal{G}(\mathbf{q})$ (dans le cas discrétisé).*

Si toutefois on considère que le robot peut délivrer des couples quelconques (c'est-à-dire qu'il se comporte comme un solide), il n'est alors plus besoin de s'intéresser à l'ensemble $\mathcal{F}'(\mathbf{q})$ et de calculer son intersection avec $\mathcal{F}$. Si de plus on discrétise les cônes $\mathcal{C}_j$ en $\mathcal{C}_j^d$, on obtient par produit d'espaces un ensemble de forces admissibles $\mathcal{F}^d$ qui est un polytope et dont la projection (par $J_{c_2}^T$) se calcule aisément pour obtenir un ensemble de stabilité $\mathcal{G}^d$ qui est lui même un polytope et ne dépend pas de la configuration $\mathbf{q}$ du robot, mais seulement de la position désirée des points de contacts C_j. Ce polytope se représente sous la forme d'une intersection de demi-espaces $\mathbf{a}_k.\mathbf{g}_2(\mathbf{q}) + b \leq 0$ qui définissent les contraintes de stabilité. Si les $\mathcal{C}_j^d$ sont inclus dans les $\mathcal{C}_j$, alors ce critère est conservatif, sous l'hypothèse qu'on peut avoir des couples quelconques.

3.5 Extensions et limitations

3.5.1 Génération de trajectoire

Moyennant quelques aménagements, le schéma de génération de posture qui a été présenté précédemment peut permettre la génération de trajectoire, c'est-à-dire trouver une fonction $t \rightarrow \mathbf{q}(t)$ vérifiant des tâches éventuellement fonctions du temps t. Par exemple on peut

vouloir trouver une trajectoire où un robot humanoïde fait un pas, déplaçant son pied d'une position à une autre. La position du pied n'est alors contrainte qu'au début et à la fin du mouvement. Cependant les contraintes telles que la collision ou la stabilité, ainsi que celles des autres contacts, doivent être vérifiées à chaque instant. Pour une trajectoire sur l'intervalle de temps $[t_0, t_f]$, on a le problème modifié suivant :

$$\mathcal{Q}_t = \begin{cases} t \to \mathbf{q}(t) \text{ est une fonction } C^2 & (3.71) \\ q_i^- \leq q_i(t) \leq q_i^+, \quad \forall i \in [|1,n|] \quad \forall t & (3.72) \\ \epsilon_{ij} \leq d(r_i(\mathbf{q}(t)), r_j(\mathbf{q}(t))), \quad \forall (i,j) \in \mathcal{I}_{\text{auto}} \quad \forall t & (3.73) \\ \epsilon_{ik} \leq d(r_i(\mathbf{q}(t)), O_k), \quad \forall (i,k) \in \mathcal{I}_{\text{coll}} \quad \forall t & (3.74) \\ s(\mathbf{q}(t), t) \leq 0 \quad \forall t & (3.75) \\ g_i(\mathbf{q}(t), t) = 0 \quad \forall\, T_i \quad \forall t & (3.76) \\ h_i(q(t), t) \leq 0 \quad \forall\, T_i \quad \forall t & (3.77) \\ \mathbf{q}(t_0) = q_0 & (3.78) \\ \mathbf{q}(t_f) = q_f & (3.79) \end{cases}$$

La première ligne assure la continuité de l'accélération, les deux dernières contraintes fixent les conditions initiales et finales.
Afin de pouvoir trouver une solution il faut simplifier le programme à la recherche de fonctions paramétrables, comme des polynômes ou des splines [deBoor 78]. Pour une classe de fonctions donnée, on cherche donc des valeurs des paramètres $\mathbf{p}$ telles que $\mathbf{q}(\mathbf{p}, t)$ vérifie l'ensemble de contraintes précédent.

Il existe des méthodes pour vérifier des contraintes sur un intervalle continu [Lengagne 07]. FSQP cependant ne permet pas cela. Dans le cadre d'une simple extension de notre programme de génération de posture original, nous adoptons un approche souvent choisie [Miossec 06] qui consiste à vérifier les contraintes en un certain nombre de points dans le temps. Si ce nombre de points est suffisant, alors les contraintes seront vraisemblablement vérifiées sur tout l'intervalle. On en vient donc à résoudre le problème suivant.

$$\mathcal{P}_t = \begin{cases} t \to \mathbf{q}(\mathbf{p}, t) \text{ est une fonction } C^2 & (3.80) \\ q_i^- \leq q_i(\mathbf{p}, t_l^i) \leq q_i^+, \quad \forall i \in [|1,n|] \quad \forall l \in \left[\left|0, l^i\right|\right] & (3.81) \\ \epsilon_{ij} \leq d(r_i(\mathbf{p}, t_l^i), r_j(\mathbf{p}, t_l^i)), \quad \forall (i,j) \in \mathcal{I}_{\text{auto}} \quad \forall l \in \left[\left|0, l^{ij}\right|\right] & (3.82) \\ \epsilon_{ik} \leq d(r_i(\mathbf{q}(\mathbf{p}, t_l^i)), O_k), \quad \forall (i,k) \in \mathcal{I}_{\text{coll}} \quad \forall l \in \left[\left|0, k^{ik}\right|\right] & (3.83) \\ s(\mathbf{q}(\mathbf{p}, t_l^i), t_l^i) \leq 0 \quad \forall l \in \left[\left|0, l^i\right|\right] & (3.84) \\ g_i(\mathbf{q}(\mathbf{p}, t_l^i), t_l^i) = 0 \quad \forall\, T_i \quad \forall l \in \left[\left|0, l^i\right|\right] & (3.85) \\ h_i(q(\mathbf{p}, t_l^i), t_l^i) \leq 0 \quad \forall\, T_i \quad \forall l \in \left[\left|0, l^i\right|\right] & (3.86) \\ \mathbf{q}(t_0) = q_0 & (3.87) \\ \mathbf{q}(t_f) = q_f & (3.88) \end{cases}$$

Le critère d'optimisation peut s'écrire comme l'intégrale discrète en fonction du temps du critère utilisé pour une génération de posture simple. Cependant, il est aussi fait recours à des critères mettant en jeu la dynamique du robot ou des dérivées de variables par rapport

au temps [Miossec 06] [Harada 06].
Si la discrétisation est assez fine, le nombre de contraintes obtenues devient vite très grand : pour une discrétisation en k pas de temps, $\mathcal{P}_t$ contient environ $k+1$ fois plus de contraintes que $\mathcal{Q}$. FSQP sait néanmoins tirer parti de la structure d'un tel problème (dit de programmation semi-infini) pour minimiser le nombre de fonctions à évaluer. Cependant, le nombre de variables est lui aussi en forte augmentation, car multiplié par le nombre de paramètres utilisés pour décrire les fonctions du temps. Le temps de résolution du système croît donc fortement : pour un humanoïde, on passe d'une résolution de l'ordre du dixième de seconde pour une génération classique à un calcul de plusieurs minutes pour une trajectoire simple. On touche là une limite due au choix de FSQP comme algorithme d'optimisation : si on choisit des fonctions qui ne dépendent localement que d'un petit nombre de leurs paramètres, comme c'est le cas de B-splines, le gradient des contraintes fait apparaître un grand nombre de zéros. La résolution du problème d'optimisation passe alors par la manipulation de matrices creuses que FSQP ne sait pas gérer de manière spécifique, entraînant des calculs plus longs. Le problème de génération de trajectoire gagnerait donc à être résolu par un autre algorithme d'optimisation. IPOPT [Wächter 06] par exemple peut faire ses calculs sur des matrices creuses. Nous n'avons pas eu le temps de mener des essais avec cet optimisateur.

Il faut noter qu'avec les critères de stabilité proposés ci-avant, il n'y a pas de notion de dynamique. Si on garde ces critères, on se restreint à chercher un chemin quasi-statique. Le paramètre de temps t ne doit alors être vu que comme une variable d'avancement le long de ce chemin. En particulier on pourra en toute généralité prendre t dans $[0,1]$. L'obtention d'une véritable trajectoire se fait ensuite par l'ajout d'un profil de vitesse le long de ce chemin, de façon à respecter l'hypothèse quasi-statique.

3.5.2 Génération de postures multiples

A mi-chemin entre la génération de posture simple et la génération de trajectoire, la génération de postures multiples consiste à résoudre en même temps, pour un même robot, plusieurs problèmes de génération ayant un couplage entre eux : $\mathcal{Q}_1$,...$\mathcal{Q}_k$, étant des problèmes tels que $\mathcal{Q}$ défini en 3.1, on cherche $\mathbf{q}_1$,...,$\mathbf{q}_k$ tel que

$$\mathcal{Q}_{\text{mult}} = \begin{cases} \mathbf{q}_i \in \mathcal{Q}_i \quad \forall i \in [|1,k|] & (3.89) \\ g_i(\mathbf{q}_1, \cdots, \mathbf{q}_k) = 0 \quad \forall\, T_i \in \mathcal{T}_{\text{coupl}} & (3.90) \\ h_i(\mathbf{q}_1, \cdots, \mathbf{q}_k) \leq 0 \quad \forall\, T_i \in \mathcal{T}_{\text{coupl}} & (3.91) \end{cases}$$

où $\mathcal{T}_{\text{coupl}}$ est un ensemble de tâches impliquant plusieurs postures du robot. On peut ainsi par exemple chercher l'existence de pas entre deux postures n'ayant pas de contact en commun comme illustré sur la figure. 3.8. Pour cela, les contacts intermédiaires des pieds sont définis comme contacts légers et les tâches de couplage forcent la continuité de placement des pieds entre deux postures consecutives. Cet exemple de génération multiple se fait dans un cas très simple où d'autres algorithmes de placement de pieds ([Kuffner 03] [Chestnutt 05] [Chestnutt 07]) iraient beaucoup plus vite, cependant cette approche est très générale et permet de prendre en compte toute sorte de contraintes, en particulier des contraintes de non-collision compliquées. Cela dit, il ne vise qu'à illustrer une extension possible de notre génération de posture. Nous ne nous sommes pas proprement penchés sur une application de la génération multiple et a fortiori sur une comparaison avec d'autres méthodes de résolution pour ces applications.

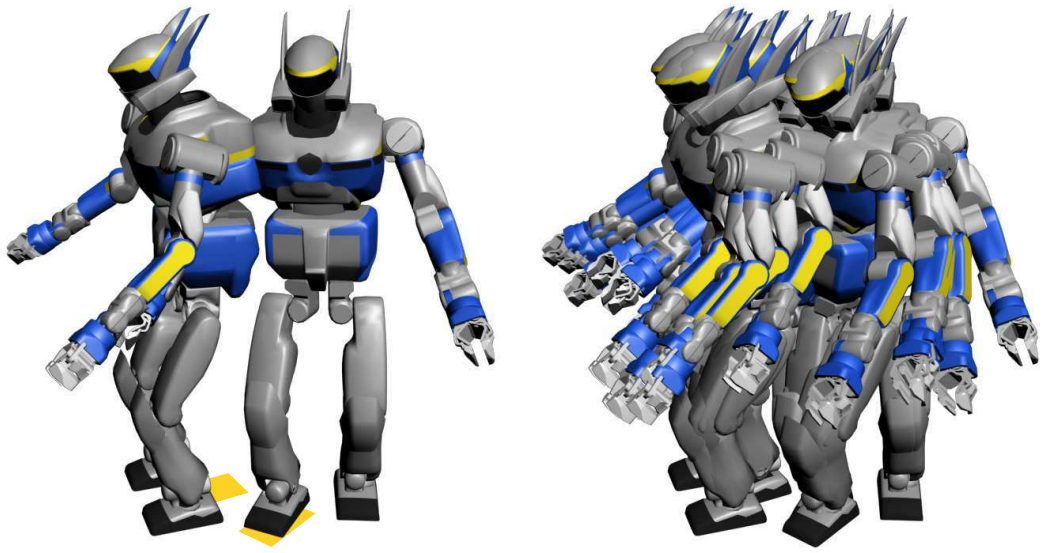


FIG. *3.8 – Génération de postures multiples pour faire marcher le robot en arrière et en tournant en 4 pas. A gauche : les positions initiale et finale, ainsi que les empreintes intermédiaires des pieds, trouvées par l'optimisation. A droite : les 10 postures résultantes (2 postures par pas pour le transfert du centre de masse, plus les postures initiale et finale). On notera que pour cet essai il n'y avait pas de contrainte d'évitement de collision. Pieds gauche et droit se percutent à une étape.*

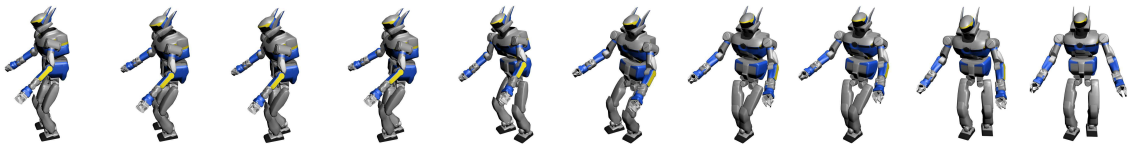


FIG. *3.9 – Les 10 postures séparées.*

Comme pour la génération de trajectoire, le nombre de variables augmente rapidement dans ce problème. Les temps de calculs montent alors à plusieurs secondes dans des cas comme présentés Fig. 3.8. Là encore, FSQP ne sait pas tirer parti de la structure fondamentalement creuse des matrices.

3.5.3 Limitations

La plus grande limitation de notre approche pour la génération de posture tient à la structure hautement non-convexe du problème due en particulier aux contraintes de non-collision. D'un point de vue purement mathématique, cela signifie qu'il n'y a aucune garantie de finir au minimum global du critère. L'expérience montre néanmoins que, quand convergence il y a, celle-ci se dirige vers le minimum auquel l'utilisateur s'attend. Cette remarque est purement empirique et il resterait à mettre en oeuvre des méthodes plus sophistiquées pour trouver de manière garantie le minimum global [Kan 89] et comparer le résultat à celui de notre générateur.
Cependant, le but premier de notre générateur, dans le cadre de la planification, est de trouver une posture sur un sous-espace de l'espace de configuration. A cet égard, le problème issu de la

non-convexité n'est pas tant de ne pas trouver la meilleure solution, que de ne pas trouver de solution alors qu'il en existe manifestement une. Quantifier l'occurrence de ces faux négatifs ferait aussi partie de l'étude évoquée ci-dessus. D'expérience, cela n'arrive pas suffisamment souvent pour gêner l'utilisateur (programme ou humain), et dépend grandement de la façon d'initialiser le problème.

Une autre limitation tient aux problèmes numériques rencontrés lors de l'optimisation et qui peuvent aussi empêcher la convergence alors qu'il existe une solution. Ces problèmes sont majoritairement reliés à un mauvais conditionnement du problème d'optimisation. Cela arrive quand des contraintes sont très proches par exemple (contraintes non qualifiées, cf [Körbel 07]), mais aussi du fait de différences d'échelle trop importantes entre contraintes (ou contraintes et critère). Il y a là un travail à effectuer pour s'assurer automatiquement à chaque construction du bon conditionnement du problème posé.

Ces deux limitations se résument sous le fait que notre générateur de posture n'est pas complet, c'est-à-dire qu'il peut ne pas trouver de solution alors qu'il en existe une.

3.6 Applications

Dans ce paragraphe, nous montrons quelques applications de la génération de posture qui ne sont pas liées à la planification.

3.6.1 Environnement réactif

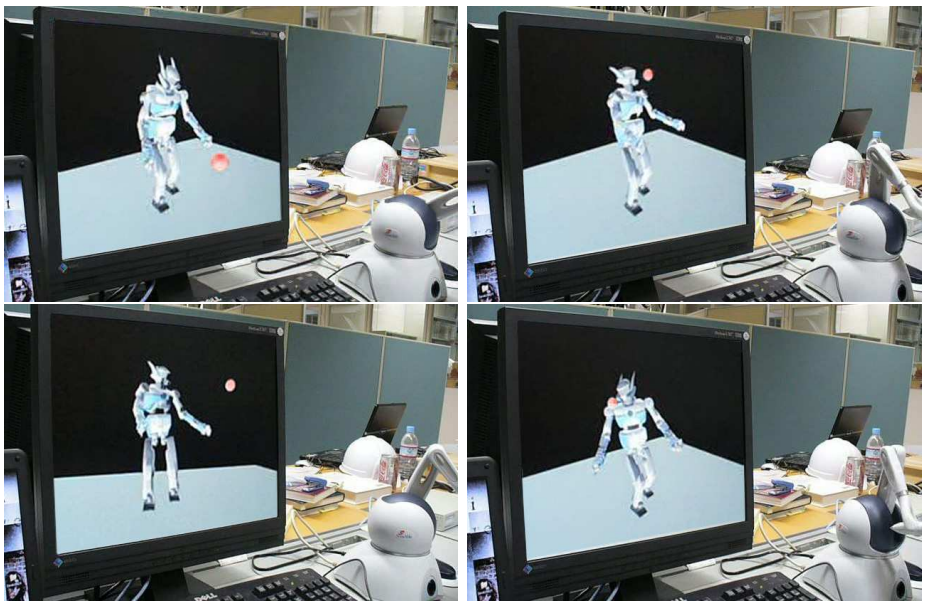


FIG. *3.10 – Génération de posture interactive : le robot doit suivre du regard une balle déplacée en 3D par un dispositif OMNI (en bas à droite de chaque image).*

La première application est l'utilisation du générateur couplée avec celle d'un dispositif haptique au sein de l'environnement de simulation interactive AMELIF [Evrard 08]. La faisabilité a été testée par le biais d'un démonstrateur sur le problème suivant : on considère un

robot (ici HRP-2) les pieds posés au sol, et ayant pour tâche de regarder une balle sans que la tête en soit trop près et sans que la balle ne rentre en collision avec une partie du robot. Cela se fait par l'ajout d'une tâche $\mathcal{T}_{\text{obs}}$ définie ainsi : H et B sont respectivement l'origine de la caméra et le centre de la balle, et $\mathbf{d}_H$ est la direction de la caméra. On veut alors que $\overrightarrow{HB}$ et $\mathbf{d}_H$ soient colinéaires et de même sens et que $HB > \epsilon$, ϵ étant une distance en deçà de laquelle l'objet est considéré comme trop proche pour pouvoir être bien observé. La colinéarité se décrit comme dans le cas des contacts (cf § 3.4.1). La particularité du problème ici est que la balle est déplacée en 3D en temps réel par un dispositif haptique (un OMNI dans notre cas). Entre deux pas de temps cependant, le déplacement n'est pas très grand si bien que deux problèmes successifs de génération de postures sont similaires. De ce fait, utiliser la solution de l'un pour initialiser le suivant engendre des temps de convergence très courts (de l'ordre du centième de seconde), permettant de faire ces générations de posture en temps réel.
Le critère à un temps t est

$$\alpha \sum_{i=1}^{n+6} \left(q_i^t - q_i^{\text{ref}}\right)^2 + (1-\alpha) \sum_{i=1}^{n+6} \left(q_i^t - q_i^{t-1}\right)^2 \tag{3.92}$$

$\mathbf{q}^{t-1}$ étant la posture au temps précédent, et $\alpha \in [0,1]$ un paramètre permettant de pondérer entre la première partie du critère qui est la distance à une posture de référence $\mathbf{q}^{ref}$ (choisie par l'utilisateur) et la seconde qui est la distance à la posture précédente, introduite afin d'adoucir le mouvement.
Quelques images du démonstrateur sont données figure 3.10.

3.6.2 Reconstruction d'objet 3D

La reconstruction d'objet 3D est cruciale pour un grand nombre d'applications relatives à la reconnaissance ultérieure de ces objets [Saidi 06]. Il existe de nombreuses techniques pour construire automatiquement un modèle numérique d'un objet en 3D. Elles ont cependant toutes un point commun : il faut observer cet objet sous plusieurs angles. Quand la (ou les) caméra(s) permettant cette observation sont montées sur un robot, il faut trouver, en plus des position et orientation de la caméra, une posture du robot correspondant à cette position, et soumise à toutes les contraintes classiques de stabilité, non-collision, etc. Nous présentons ici deux applications de la génération de posture en relation avec la reconstruction d'objet. La première se sert de notre générateur pour satisfaire les contraintes. La deuxième tire en plus parti d'un design judicieux du critère d'optimisation pour déterminer la position et l'orientation de la caméra.

3.6.2.1 Postures clé pour l'observation d'objets

Si le choix du point de vue est laissé à un algorithme de décision au-dessus du générateur, il reste néanmoins à celui-ci à trouver une posture correspondante. Il faut alors définir une tâche d'observation adéquate. Dans le cadre de ce travail [Stasse 07], il est demandé au robot de regarder l'objet selon diverses directions. La tâche d'observation se définit par un point X de l'espace de travail et un vecteur de direction du regard $\mathbf{v}$, ainsi qu'une distance minimale d_{min} à respecter entre la tête et l'objet (cf Fig. 3.11). H étant le centre des caméras et $\mathbf{d}_H$ leur direction, on veut alors que $\mathbf{v}$ et $\overrightarrow{HX}$ d'une part, $\mathbf{d}_H$ et $\mathbf{v}$ de l'autre, soient colinéaires

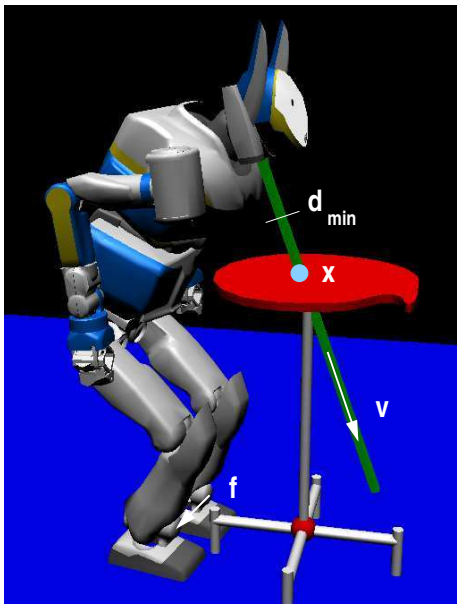


FIG. *3.11 – Observation d'un objet selon une direction. X est un point de l'objet, $\mathbf{v}$ la direction du regard. Les pieds sont fixés l'un par rapport à l'autre par une matrice de transformation f, mais sont libres de se mouvoir sur le sol.*

et de même sens et que $HX \geq d_{\min}$.
Dans ce travail, les pieds ne sont pas fixés dans l'environnement, mais peuvent se mouvoir sur le sol. Cependant, pour faciliter les calculs des contraintes de stabilité, la position relative entre les deux pieds est fixée via une matrice de transformation f. On utilise un contact léger (cf § 3.4.1) pour permettre à l'un des deux pieds de se placer librement sur le sol. Le second est fixé par rapport au premier en utilisant une adaptation de contact fort (qui correspond à fixer un corps dans le repère absolu alors qu'ici on cherche à fixer un corps dans le repère d'un autre), où le repère $(E, \mathbf{d}_E, \mathbf{b}_E, \mathbf{n}_E)$ de l'environnement est remplacé par un repère $(F, \mathbf{d}_F, \mathbf{b}_F, \mathbf{n}_F)$ lié au premier pied, et défini par f.
Enfin, on cherche à favoriser la stabilité en prenant pour critère la distance de la projection du centre de gravité au centre du polygone de sustentation.
La figure 3.12 montre les poses obtenues pour l'observation d'un objet sous des angles allant de 0 à 70 degrés.

3.6.2.2 Maximisation de l'inconnu observé

Si en revanche on ne laisse pas la décision de la direction d'observation à un planificateur, on peut vouloir embarquer ce processus de décision dans l'optimisation même, par le biais du critère. A cette fin, le générateur de posture est utilisé dans les travaux de Foissotte [Foissotte 08] qui se focalisent sur le design d'une fonction objectif adéquate afin de trouver le meilleur prochain point d'observation d'un objet.

L'objet est reconstruit par la méthode dite du "*space carving*" [Kutulakos 99], qui consiste à sculpter un bloc de voxels 3D à partir de la silhouette et de la surface de l'objet vu sous plusieurs angles : pour un point de vue donné, on obtient une image stéréoscopique à partir de laquelle est calculée la surface visible et la silhouette de l'objet. Tous les voxels du modèle

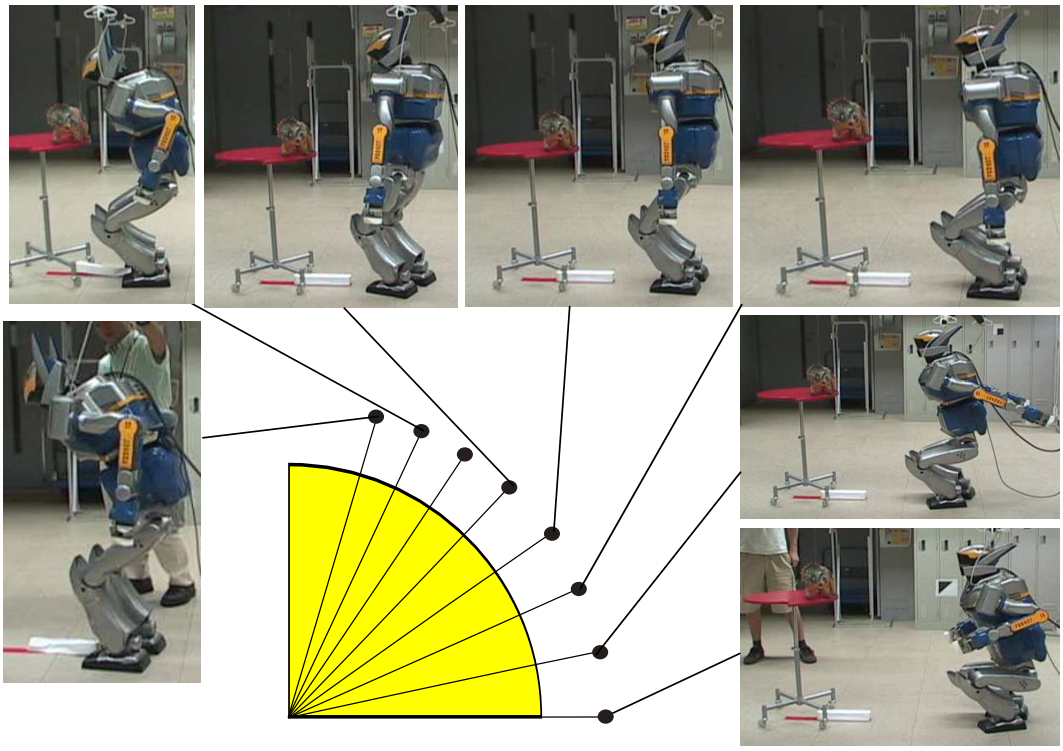


FIG. *3.12 – Poses obtenues sur HRP-2 pour différents angles d'observation d'un objet.*

numérique de l'objet qui se projettent dans l'image hors de cette silhouette sont éliminés, ainsi que tous ceux qui sont devant la surface de l'objet. Seuls restent les voxels de l'objet et ceux qu'il cache (cf Fig. 3.13). On prend alors un autre point de vue. Après un certain nombre d'itérations, qui est d'autant plus petit que les points de vue sont bien choisis, on obtient ainsi une bonne approximation de l'objet. Ce bon choix se base sur la quantité de voxels qu'un point de vue permettra d'observer alors qu'ils ne l'avaient pas été lors des itérations précédentes.

On appelle voxel *inconnu* un voxel qui n'a pu être observé jusqu'à l'itération en cours. Par opposition les autres voxels sont *connu*. Une prise de vue de la scène correspond à une projection des voxels sur le plan image, en tenant compte des occlusions. Le critère est une mesure de la surface de cette image occupée par la projection des voxels inconnus. On cherche à maximiser cette surface, afin d'obtenir le plus d'information possible lors du prochain point de vue.

Ici, les pieds sont complètement libres de leurs mouvements sur le sol (contact léger pour chacun). Pour pouvoir cependant avoir un critère de stabilité calculable sans trop de difficulté, celui-ci est choisi de manière restrictive comme étant l'appartenance de la projection verticale du CdM à la ligne joignant le centre des deux pieds. Un exemple de posture obtenue est montré figure 3.14.

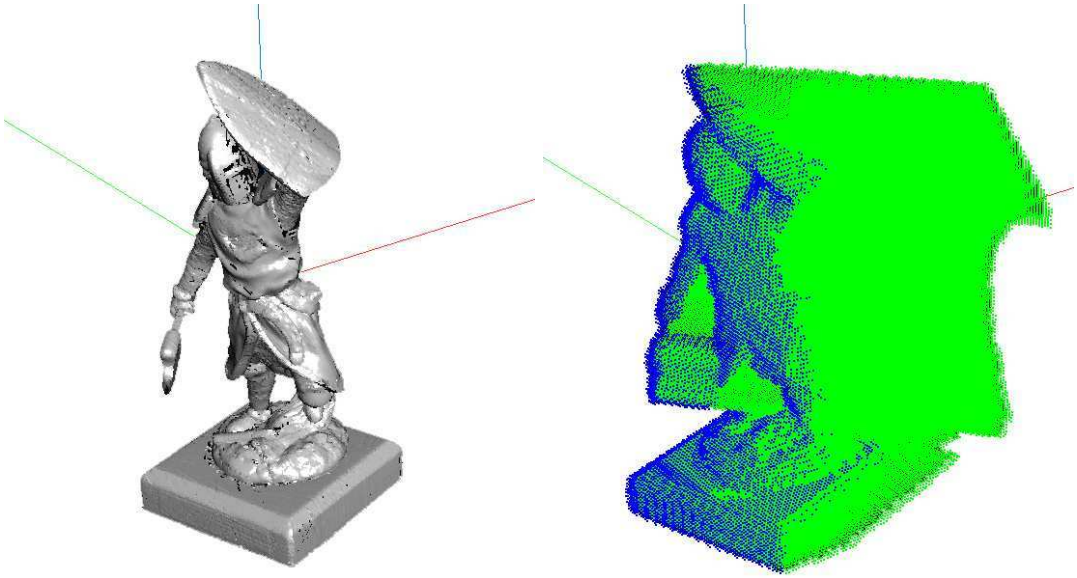


FIG. *3.13 – Une figurine de soldat (à gauche) est observée et reconstruite par la méthode du space carving (à droite, après une itération pour une position des caméras face à l'objet). Les voxels bleus sont ceux détectés comme appartenant à la surface de l'objet, les verts sont ceux qui n'ont pas encore été observés, car cachés par cette surface.*

3.7 Conclusion

Nous avons décrit dans ce chapitre une méthode pour générer des postures stables, sans collision et réalisant un certain nombre de tâches imposées par l'utilisateur. Elle repose sur l'écriture et la résolution d'un problème d'optimisation sur des variables liées au robot (degrés de liberté principalement, mais aussi éventuellement forces de contact,...). Nous avons donné la façon mathématique dont s'écrit ce problème et ses diverses composantes, ainsi que la forme de son implémentation, qui permet une description par des objets de haut niveau.

Cette méthode va nous permettre de trouver des configurations sur des sous-espaces de l'espace de configuration correspondants à des ensembles de contact. Elle repose cependant sur un problème d'optimisation dont l'implémentation de la résolution requiert l'utilisation de fonctions à gradient continu. La fonction de distance euclidienne entre deux objets, utilisée pour les contraintes de non-collision, pose des problèmes à cet égard. Nous exposons ces problèmes et une manière de les résoudre dans le chapitre suivant.

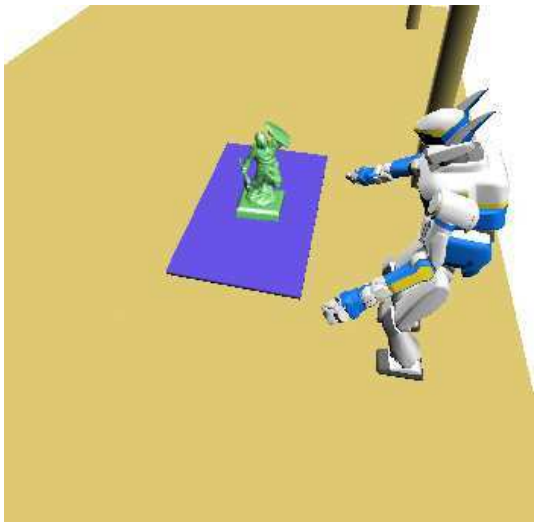


FIG. *3.14 – HRP-2 observant le soldat.*

Chapitre 4

Volume englobant assurant la continuité du gradient de la distance

Nous avons vu dans le chapitre précédent une méthode pour obtenir des postures pour un ensemble de contacts spécifiés, et plus généralement pour trouver des postures sur un sous-espace de l'espace de configuration défini par un ensemble de tâches. En plus de ces tâches, données par l'utilisateur, un certain nombre de contraintes doivent être respectées, dont celles de non-collision. Celles-ci se basent majoritairement sur la distance entre objets. Le calcul de distance entre objets est d'un très grand intérêt en planification, contrôle et optimisation de mouvements. De nombreuses stratégies utilisées dans ces cadres, y compris la génération de posture présentée précédemment, se basent sur des méthodes de descente de gradient et requièrent que le gradient de la distance soit continu. Pour être aussi rapide que possible, le calcul de distance s'effectue toujours entre des volumes englobants ayant des formes simples, un pour chaque objet considéré. Cependant, les volumes englobants existants soit n'offrent pas une bonne approximation de l'objet original, soit ne garantissent pas la continuité du gradient. En se basant sur les causes de discontinuité de la distance, nous présentons dans ce chapitre un volume englobant composé de parties de sphères et de tores (d'où l'acronyme anglophone de STP - sphere torus patches) qui résout ce problème. Ce volume est aussi une très bonne approximation de l'objet initial et offre des temps de calcul de distance comparables à ceux obtenus avec les algorithmes et volumes classiques.

4.1 Introduction

Les volumes englobants (VE) sont des représentations géométriques simplifiées d'un objet, utilisées généralement pour accélérer des calculs dont le résultat approximatif suffit ou est passé à un autre niveau de calcul plus fin (comme par exemple la détection de collision).

Parce qu'ils cachent des détails négligeables dans un calcul rapide, les VE classiques ont des formes géométriques simples et sont généralement convexes pour être mieux exploités dans des calculs d'optimisation. Cependant, le choix d'une géométrie particulière peut produire une simplification telle que la forme de l'objet initial est méconnaissable ; c'est la raison pour laquelle la représentation en VE peut être hiérarchisée. Les hiérarchies de VE (HVE) ont prouvé qu'elles étaient un bon compromis entre approximation fidèle de la forme originale et simplicité. Elles sont très utilisées pour accélérer la détection de collision ou le calcul de proximité dans de nombreux domaines. Les recueils scientifiques récents [van den Bergen 04] [Ericson 05] offrent une excellente revue des méthodes sur les VE et HVE.

En infographie, les VE sont principalement utilisés pour la détection de collision dans des moteurs physiques pour l'animation. En robotique, ils sont largement utilisés en planification [LaValle 06] pour couvrir des obstacles ou des parties du robot, afin d'offrir des stratégies rapides de détection de collision ou d'auto-collision, mais ils trouvent aussi un domaine d'application important dans la commande réactive ou l'optimisation de mouvement. Selon la méthode choisie pour ces applications, l'évitement de collision entre VE peut induire des contraintes supplémentaires : de nombreuses méthodes sont locales et basées sur une descente de gradient. Il est alors important d'assurer la continuité des distances qui interviennent dans les contraintes assurant l'évitement de collision, afin de garantir la continuité de la vitesse.

Clairement, mais aussi par habitude, les VE polyédriques comme les boîtes englobantes isothétiques (AABB) ou orientées (OBB) [Gottschalk 96], ou encore les $k-$Dops [Klosowski 98], qui sont des polytopes à k faces orientées (k atteint son maximum avec l'enveloppe convexe), sont de bonnes solutions pour envelopper finement un objet, tout en restant simple. Cependant, comme nous le démontrerons plus tard dans ce chapitre, la distance obtenue entre de tels objets n'est pas garantie d'avoir un gradient continu. Nous montrerons en effet que cette propriété n'est assurée que pour des VE strictement convexes.

D'autres classes de VE, comme les sphères, les ellipsoïdes ou super-ellipsoïdes sont bien strictement convexes, et en tant que tel sont candidats pour assurer la continuité du gradient de la distance. Cependant, une simple sphère ou ellipsoïde n'offre qu'un ratio très médiocre entre son volume et celui de l'objet englobé, spécialement si celui-ci est long et fin (comme cela est souvent le cas pour les corps d'un humanoïde). Les super-ellipsoïdes voire les superquadriques, quant à elles, sont de très bons candidats, tant pour assurer la continuité que pour leur approximation de l'objet, mais le calcul de distance pour ce type d'objets se révèle être trop coûteux [Chakraborty 06] pour pouvoir être utilisé en contrôle. L'utilisation de HVEs faites des primitives ci-dessus, dans le but d'accélérer les calculs, ne peut pas se faire, car changer de volume au sein d'une hiérarchie implique un saut des points témoins de la distance, ce qui cause une discontinuité de gradient comme on le verra de manière claire dans la suite. Le seul moyen d'éviter ces sauts serait de suivre la distance entre toutes les paires de primitives, ce qui fait perdre l'intérêt des HVEs car étant trop coûteux.

Plutôt que de chercher à modifier les objets sur lesquels on calcule les distances, on pourrait chercher à régulariser la fonction distance directement (et donc utiliser une pseudo-distance avec les propriétés voulues). L'utilisation de R-fonctions pour créer des champs de distance comme proposé dans [Shapiro 99] pourrait être une belle solution à notre problème. Cependant de tels champs sont très coûteux en temps de calcul, comme le fait remarquer [Fuhrmann 03], et ne sont calculés jusqu'à présent que pour la (pseudo-)distance d'un point à l'objet. Il faudrait donc adapter cette méthode à une paire d'objets. Même l'utilisation

de pseudo-distance ne résout pas forcément le problème : dans l'approche de [Zhu 04] une fonction jauge est utilisée à partir de laquelle une pseudo-norme est définie. La pseudo-distance est alors définie à partir de la résolution d'un problème d'optimisation spécifique selon que les objets sont séparés (un simplexe) ou non (plusieurs simplexes), ce qui demande un premier test pour définir le cas. Les auteurs distinguent, parmi les conditions de différentiabilité de leur pseudo-distance, l'unicité des solutions des problèmes d'optimisations associés. Ceci n'est le cas que si l'un des objets est strictement convexe.

La motivation du travail présenté dans ce chapitre est de garantir que le gradient de la distance entre deux objets est une fonction continue de la position relative de ces objets. A l'exception des travaux pionniers de Gilbert et Johnson [Gilbert 85], qui démontrent des propriétés de la distance, et notamment la nécessité de la stricte convexité pour s'assurer la continuité du gradient, la régularité de la distance et de ses dérivées n'a que rarement été discutée, elle a même parfois été simplement supposée établie. Par exemple, dans les travaux proposés dans [Lee 01] [Choset 97], où le problème de continuité est soulevé dans le cas 2D, il est affirmé que la distance entre objets convexes est C^∞ et donc que le gradient est continu. L'erreur dans la démonstration d'une telle affirmation vient de la supposition implicite que les points témoins de la distance sont des fonctions continues de la position des objets. Or tel n'est pas le cas, et le problème de continuité du gradient vient même de ce point précis. L'hypothèse de continuité des points témoins est très souvent faite implicitement dans les travaux qui calculent le gradient de la distance.

Le reste du chapitre est organisé comme suit : dans le paragraphe 4.2, nous montrons le problème de discontinuité par le biais d'un exemple simple, ainsi que sa répercussion dans une optimisation. Il s'ensuit une étude des propriétés de continuité de la distance entre objets convexes (§ 4.3). Celle-ci reprend des résultats introduits dans [Gilbert 85]. Nous démontrons ici de manière différente certains théorèmes obtenus dans cet article dont nous n'avions pas connaissance au début de ce travail. Rusaw, dans [Rusaw 01], a utilisé l'analyse non régulière dans le contexte de la planification basée sur capteurs. Dans [Patel 05], les auteurs utilisent des cylindres comme volumes englobant le robot, et font part de problèmes dans le contrôle lorsque deux cylindres deviennent parallèles. Ils proposent comme solution de limiter la commande de vitesse qu'ils obtiennent alors. Dans un travail récent, Kanehiro *et al.* [Kanehiro 08] montrent qu'il est possible de prendre en compte les contraintes localement dans l'évitement de collisions en mode réactif. Leur méthode consiste à prendre en compte (localement) tous les points témoins possibles entre les primitives proches (pour diverses situations et configurations de primitives géométriques des deux objets). De ce fait, cela ne nécessite pas que les corps soient convexes. Ceci est possible car le champ de distance est par nature continu localement, cependant la complexité de cette approche (il a été dénombré jusqu'à 4000 contraintes par zone potentielle de collision dans les configurations tests utilisant HRP-2) peut être vraiment réduite en considérant le problème comme une discrétisation du champ de distance comme cela a été fait dans [Johnson 01].

Notre approche, au contraire des autres exemples, propose de se défaire de la non-différentiabilité en construisant un nouveau type de VE strictement convexe (§ 4.4). Ce VE est un assemblage de parties de sphères et de tores, qui permet de choisir par réglage d'un paramètre le compromis entre la précision d'approximation de l'objet et la régularisation du gradient. Ce VE est calculé hors ligne (Annexe B.3), de manière à ce que, tout en restant très proche de l'objet original, il permette un calcul en ligne rapide de la distance et de son gradient (§ 4.5). Enfin nous donnons des exemples de génération de posture pour un robot

humanoïde en utilisant ces VE (§ 4.6).

4.2 Motivations

4.2.1 Exemple de discontinuité du gradient de la distance en 2D

Considérons le segment de la figure 4.1. d est la distance entre ce segment et l'axe horizontal et nous nous intéressons à la dérivée de d par rapport à θ. Il a été montré [Rusaw 01] que cette dérivée est discontinue (cf Fig. 4.3(a)). La discontinuité arrive lorsque le segment devient parallèle à l'axe horizontal : dans ce cas, le point témoin de la distance sur le segment saute instantanément entre les deux extrémités de celui-ci (cf Fig. 4.2). Etant donné que la distance entre chacune de ces extrémités et l'axe horizontal change de manière contraire en fonction de θ, ce saut d'un point à un autre entraîne une discontinuité de la dérivée.

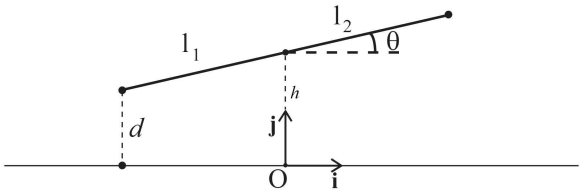
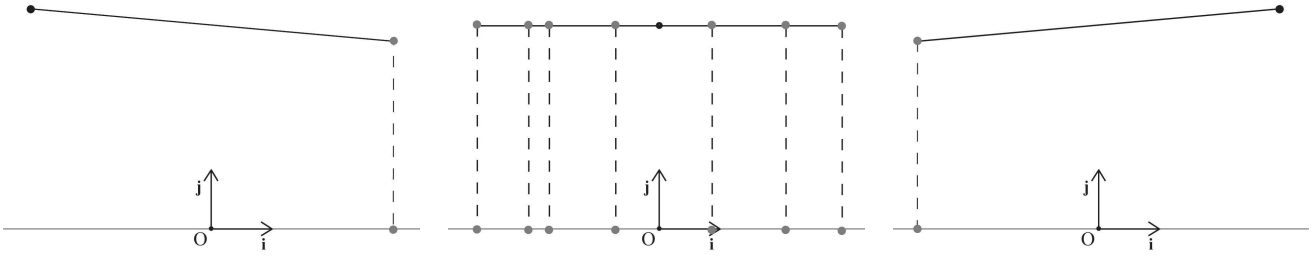


FIG. *4.1 – Un segment qui peut tourner librement autour d'un de ses points. d est la distance à l'axe horizontal.*



(a) $\theta < 0$ (b) parallel case : $\theta = 0$ (c) $\theta > 0$

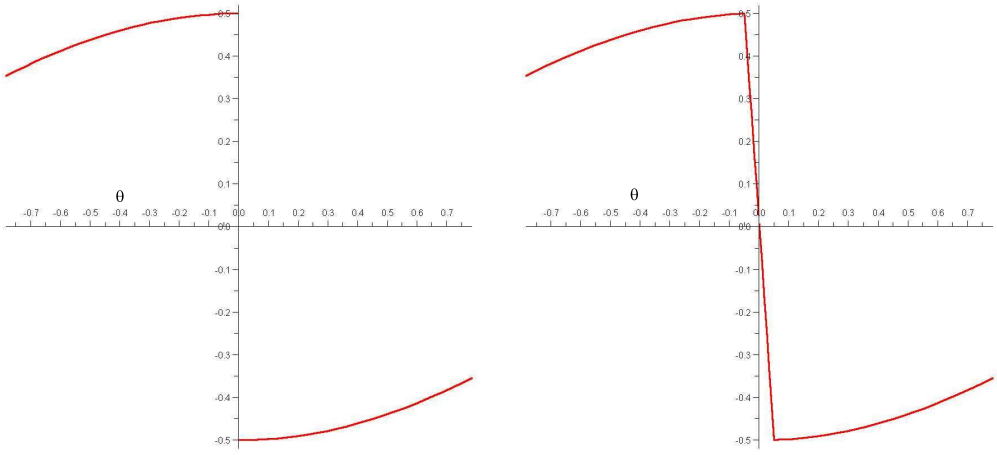
FIG. *4.2 – Quand θ passe par 0 (4.2(b)), les points témoins (en gris) sautent entre les configurations illustrées en 4.2(a) et 4.2(c)*

On a

$$d(\theta) = \begin{cases} h + l_2 \sin\theta & \text{si } \theta < 0 \\ h & \text{si } \theta = 0 \\ h - l_1 \sin\theta & \text{si } \theta > 0 \end{cases} \tag{4.1}$$

et donc

$$\frac{\partial d}{\partial \theta}(\theta) = \begin{cases} l_2 \cos\theta & \text{si } \theta < 0 \\ \text{indéfini} & \text{si } \theta = 0 \\ -l_1 \cos\theta & \text{si } \theta > 0 \end{cases} \tag{4.2}$$



(a) discontinuité de la dérivée (Fig. 4.1) (b) derivée regularisée (Fig. 4.4)

FIG. *4.3 – $\partial d/\partial\theta$ pour $l_1 = l_2 = 0.5$*

4.2.2 Régularisation du gradient

Afin d'éviter les discontinuités de gradient, l'idée est d'empêcher les sauts de points témoins. Ces sauts se produisent lorsque des parties *planes* de chacun des deux objets se retrouvent parallèles. On cherche donc à transformer ces parties planes, en les arrondissant. Dans notre exemple, une possibilité est d'approximer le segment par un arc de cercle de très grand rayon (cf Fig. 4.4). Le point témoin, qui est alors unique, se déplace alors continûment entre les deux extrémités, et les deux parties de la dérivée sont reliées continûment (cf Fig. 4.3(b)). Dans ce cas, on a

$$d(\theta) = \begin{cases} h + l_2 \sin\theta & \text{si } \theta < -\theta_0 \\ h + R\cos(\theta+\theta_0) - R + l_2\sin\theta = h + R\cos(\theta-\theta_0) - R - l_1\sin\theta & \text{si } -\theta_0 \leq \theta \leq \theta_0 \\ h - l_1\sin\theta & \text{si } \theta > \theta_0 \end{cases} \tag{4.3}$$

où R est le rayon du cercle et $\theta_0 = \sin^{-1}\frac{l_1+l_2}{2R}$ est le demi-angle d'ouverture de l'arc de cercle. L'expression de d entre $-\theta_0$ et θ_0 est obtenue en remarquant que le centre du cercle est de coordonnées $\left(\frac{l_2-l_1}{2}, \sqrt{R^2 - \left(\frac{l_1+l_2}{2}\right)^2}\right)^T = (l_2 - R\sin\theta_0, R\cos\theta_0)^T = (-l_1 + R\sin\theta_0, R\cos\theta_0)^T$ dans le repère lié au segment et d'origine son centre de rotation. Il vient ensuite que

$$\frac{\partial d}{\partial\theta}(\theta) = \begin{cases} l_2\cos\theta & \text{si } \theta < -\theta_0 \\ -R\sin(\theta+\theta_0) + l_2\cos\theta = -R\sin(\theta-\theta_0) - l_1\cos\theta & \text{si } -\theta_0 \leq \theta \leq \theta_0 \\ -l_1\cos\theta & \text{si } \theta > -\theta_0 \end{cases} \tag{4.4}$$

Cette approche est l'idée principale derrière la construction du volume englobant présenté dans la section 4.4.

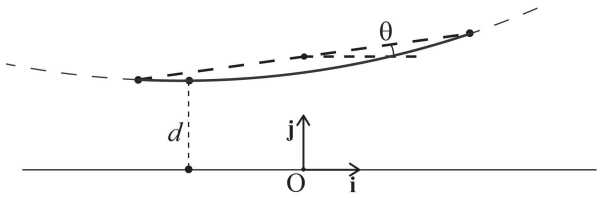


FIG. 4.4 – *Le segment est remplacé par un arc de cercle de grand rayon. Ainsi, lors du passage par $\theta = 0$, la paire de points témoins se déplace de manière continue.*

4.2.3 Exemple d'incidence du problème lors d'une optimisation

Considérons encore une fois le segment de la figure 4.1 et intéressons nous aux problèmes de minimisation suivants :

$$\min_{h,\theta} h^2 + \theta^2 \quad (\mathcal{P}_0)$$

et

$$\min_{h,\theta} h^2 + (\theta - \theta_b)^2 \quad \left(\mathcal{P}_c^{\theta_b}\right)$$
$$d(\theta) \geq d_0$$

La solution (h_0, θ_0) des deux problèmes $(\mathcal{P}_0)$ et $(\mathcal{P}_c^0)$ est évidente : $(0, 0)$ pour le premier, $(d_0, 0)$ pour le second. Cependant, $\partial d/\partial\theta$ est discontinue en $\theta = 0$. L'effet de cette discontinuité se voit clairement sur les résultats de l'optimisation, faite par FSQP, qui sont consignés dans la table 4.2.3 : l'optimisation a du mal à converger, et finalement trouve la solution en un grand nombre d'itérations et un temps bien plus long que pour le problème non contraint. On voit aussi que pour un biais θ_b se rapprochant de 0, et donc pour un optimum se rapprochant de la discontinuité, l'optimisation se fait de plus en plus difficilement.

	$(\mathcal{P}_0)$	$(\mathcal{P}_c^0)$	$(\mathcal{P}_c^{0.01})$	$(\mathcal{P}_c^{0.05})$	$(\mathcal{P}_c^{0.1})$	$(\mathcal{P}_r)$
h	0	0.10000001	0.10000009	0.100011	0.1200149	0.100125
θ	0	$-1.33 \cdot 10^{-7}$	$5.09 \cdot 10^{-8}$	$2.20 \cdot 10^{-5}$	$4.004 \cdot 10^{-2}$	$1.044 \cdot 10^{-9}$
itérations	2	231	126	15	5	12
temps (ms)	0.21	15.3	8.219	1.266	0.516	2.1

TAB. 4.1 – *Optimisation de $(\mathcal{P}_0)$, $\left(\mathcal{P}_c^{\theta_b}\right)$ et $(\mathcal{P}_r)$ pour $l_1 = l_2 = 0.5$, $R = 100$ et $d_0 = 0.1$*

Si on remplace dans $(\mathcal{P}_c^0)$ le segment par un arc de cercle de grand rayon et on considère d' la distance de cet arc à l'axe horizontal, on a le problème régularisé suivant :

$$\min_{h,\theta} h^2 + \theta^2 \quad (\mathcal{P}_r)$$
$$d'(\theta) \geq d_0$$

Les résultats de l'optimisation de ce problème (table 4.2.3) comparés à ceux de $(\mathcal{P}_c^0)$ montrent une nette amélioration du temps de calcul, qui se fait au détriment d'une légère perte de précision sur l'optimum. Le calcul de distance plus coûteux implique aussi que le gain en temps de calcul n'est pas proportionnel au gain sur le nombre d'itérations. Sur un problème plus complexe cependant la différence de temps de calcul de distance tend à être occultée par le temps passé dans l'algorithme d'optimisation même.

4.3 Propriétés de continuité de la distance

4.3.1 Définition du problème et notations

Définition (Convexité et stricte convexité) : Un objet O est dit convexe si et seulement si (ssi) $\forall (A, B) \in O^2$ et $\forall t \in]\,0, 1\,[$ le point P, défini par $P = (1-t)A + tB$ appartient à O. De plus, il est strictement convexe si P appartient à l'intérieur de O.

Dans cette section, nous considérons la distance entre deux objets *convexes* O_1 et O_2. On note δ cette distance. La position relative entre les deux objets est paramétrée par 6 scalaires (3 pour la rotation et 3 pour la translation). On note $\mathbf{q}$ le vecteur de ces scalaires.
δ est une fonction de $\mathbf{q}$ et nous nous attachons à étudier sa continuité.
On appelle points témoins une paire de points de $O_1 \times O_2$ qui sont à la distance δ l'un de l'autre. Sous certaines conditions que nous exposerons plus tard (cf théorème 4.3.1), cette paire de points est unique pour une position relative donnée. On définit alors $\mathbf{p}_{\min}(\mathbf{q}) = (\mathbf{p}_{1\,\min}(\mathbf{q}), \mathbf{p}_{2\,\min}(\mathbf{q}))^T$ la fonction qui à chaque $\mathbf{q}$ associe l'unique paire de points témoins correspondante. Il est aisé de montrer que les points témoins appartiennent à la surface des objets. Or pour un objet, la surface peut être écrite comme une fonction de deux paramètres (par exemple en coordonnées sphériques). Soit $\mathbf{u}$ le vecteur à 4 dimensions regroupant les deux paramètres de chacune des deux surfaces, et soit r_1 et r_2 les fonctions correspondant à O_1 et O_2 (pour simplifier l'écriture, nous définirons ces fonctions comme des fonctions d'un sous-ensemble de $\mathbb{R}^4$ dans $\mathbb{R}^3$, bien qu'elles ne soient réellement que des fonctions d'un sous-ensemble de $\mathbb{R}^2$ dans $\mathbb{R}^3$).
Les points témoins étant sur la surface des objets, on définit $\mathbf{u}_{\min}$ la fonction qui à $\mathbf{q}$ associe le vecteur u de coordonnées des points témoins. En notant $R(\mathbf{q})$ la matrice de rotation paramétrée par $\mathbf{q}$ et $\mathbf{T}(\mathbf{u})$ le vecteur de translation, on a

$$\mathbf{p}_{\min}(\mathbf{q}) = \left(r_1\left(\mathbf{u}_{\min}(\mathbf{q})\right), R(\mathbf{q}) r_2\left(\mathbf{u}_{\min}(\mathbf{q})\right) + \mathbf{T}(\mathbf{q})\right)^T \tag{4.5}$$

Pour finir, nous définissons $f(\mathbf{q}, \mathbf{u}) = r_1(\mathbf{u}) - R(\mathbf{q}) r_2(\mathbf{u}) - \mathbf{T}(\mathbf{q})$, le vecteur distance entre deux points paramétrés par $\mathbf{u}$, tel que

$$\delta(q) = \min_u \; \|f(q, u))\| = \|f\left(q, u_{\min}(q)\right)\| \tag{4.6}$$

On notera que f est une fonction C^∞ de $\mathbf{q}$ car R et $\mathbf{T}$ sont une rotation et une translation, et que si r_1 et r_2 sont des fonctions C^k de $\mathbf{u}$, alors f l'est aussi.

4.3.2 Stricte convexité d'un corps

On admettra dans cette section que les dérivées envisagées existent.
Comme montré dans la section 4.2, une discontinuité du gradient apparaît lorsqu'il y a un saut de la paire de points témoins. Cela arrive autour d'une configuration pour laquelle cette paire n'est pas unique, comme lorsqu'une arête est parallèle à une face. Si l'un des objets est strictement convexe, ce parallélisme ne se produit pas. Les théorèmes suivants font le lien entre la stricte convexité *d'au moins un corps* et l'unicité des points témoins, puis la continuité du gradient :

Thèoreme 4.3.1 (Unicité des points témoins) *Il y a une unique paire de points témoins si au moins un des corps est strictement convexe.*

Preuve: Si les deux objets ne sont pas strictement convexes, il peut y avoir une infinité de paires de points témoins comme le montre la figure 4.2(b).
Sinon, construisons une preuve par l'absurde : O_2 étant strictement convexe, on suppose qu'il existe deux paires distinctes de points témoins $\{A_1, A_2\}$ et $\{B_1, B_2\}$ (cf Fig. 4.5). Les droites (A_1B_1) et (A_2B_2) sont parallèles de telle sorte que si M_1 est un point de $]\,A_1B_1\,[$, sa projection orthogonale sur (A_2B_2) est un point M_2 de $]\,A_2B_2\,[$ et $A_1A_2 = B_1B_2 = M_1M_2$. M_2 est strictement à l'intérieur de O_2, et donc il existe une intersection de $]\,M_1M_2\,[$ avec la frontière de O_2. $M_1 \in O_1$, $M_2' \in O_2$ et $M_1M_2' < M_1M_2 = A_1A_2$ ce qui est contraire au fait que (A_1, A_2) soit une paire de points. □

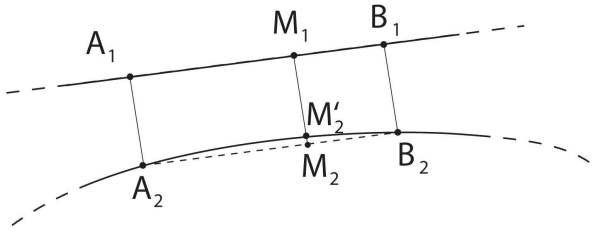


FIG. *4.5 – Avec un objet strictement convexe O_2, si on suppose qu'il existe deux paires de points témoins ($\{A_1, A_2\}$ et $\{B_1, B_2\}$), alors on peut trouver des points ($\{M_1, M_2'\}$) des objets qui sont plus proches amenant à une contradiction.*

Théorème 4.3.2 *Les points témoins de la distance entre deux objets convexes sont des fonctions continues de* $\mathbf{q}$ *si au moins l'un d'eux est strictement convexe (*$\mathbf{u}_{\min}$ *et* $\mathbf{p}_{\min}$ *sont des fonctions de* $\mathbf{q}$*).*

Preuve: Voir annexe B.1. □

Le résultat suivant est presque direct :

Théorème 4.3.3 *La distance entre deux objets convexes est une fonction* C^1 *de* $\mathbf{q}$ *si et si seulement un des objets est strictement convexe.*

Preuve: Dérivons $\delta(\mathbf{q}) = \|f(\mathbf{q}, \mathbf{u}_{\min}(\mathbf{q}))\|$ par rapport à $\mathbf{q}$:

$$\begin{aligned}\frac{\partial \delta}{\partial \mathbf{q}}(\mathbf{q}) &= \left(\frac{\partial f}{\partial \mathbf{q}} + \frac{\partial \mathbf{u}_{\min}}{\partial \mathbf{q}} \cdot \frac{\partial f}{\partial \mathbf{u}}\right)^T \frac{f}{\|f\|} \\ &= \left(\frac{\partial f}{\partial \mathbf{q}}(\mathbf{q}, \mathbf{u}_{\min}(\mathbf{q}))\right)^T \frac{f(\mathbf{q}, \mathbf{u}_{\min}(\mathbf{q}))}{\|f(\mathbf{q}, \mathbf{u}_{\min}(\mathbf{q}))\|} \qquad (4.7)\end{aligned}$$

car $\left(\frac{\partial f}{\partial \mathbf{u}}(\mathbf{q}, \mathbf{u}_{\min}(\mathbf{q}))\right)^T f(\mathbf{q}, \mathbf{u}_{\min}(\mathbf{q})) = 0$
Le résultat s'obtient par composition, f étant une fonction C^∞ de $\mathbf{q}$ et C^0 de $\mathbf{u}$. □
Le point central de cette démonstration est l'élimination du terme avec $\frac{\partial f}{\partial \mathbf{u}}$ de telle sorte que la propriété C^0 de f par rapport à $\mathbf{u}$ est suffisante.

Jusqu'à présent, nous n'avons pas requis d'autres propriétés des objets que celles d'être continus et convexes (strictement pour un). Cependant, si les surfaces de ces objets ont des propriétés de continuité *supplémentaires*, la distance en bénéficiera, comme le montre les théorèmes suivants :

Théorème 4.3.4 *Si la surface des deux objets est C^k, avec $k \geq 2$ alors les points témoins sont des fonctions C^{k-1} de* $\mathbf{q}$.

Preuve: Soit $\mathbf{u}_0$ les coordonnées des points témoins à la configuration $\mathbf{q}_0$.
Nous avons $\left(\frac{\partial f}{\partial \mathbf{u}}(\mathbf{q}_0, \mathbf{u}_0)\right)^T f(\mathbf{q}_0, \mathbf{u}_0) = 0$ (condition d'optimalité du problème (4.6)) qui peut se réécrire $\frac{\partial f^2}{\partial \mathbf{u}}(\mathbf{q}_0, \mathbf{u}_0) = 0$.
Pour $\mathbf{q}_0$ donné, f^2 est le carré de la distance entre deux points de la surfaces des objets, et donc une fonction strictement convexe, d'où $\frac{\partial^2 f^2}{\partial \mathbf{u}^2} \neq 0$.
On note $F(\mathbf{q}, \mathbf{u}) = \frac{\partial f^2}{\partial \mathbf{u}}(\mathbf{q}, \mathbf{u})$. F is C^{k-1}, $F(\mathbf{q}_0, \mathbf{u}_0) = 0$ et $\frac{\partial F}{\partial \mathbf{u}}(\mathbf{q}_0, \mathbf{u}_0) \neq 0$. Alors $\mathbf{u}$ est localement une fonction C^{k-1} de $\mathbf{q}$ (théorème des fonctions implicites). Cela implique que $\mathbf{u}_{\min}$ est une fonction C^{k-1} de $\mathbf{q}$. □

Théorème 4.3.5 *Si les surfaces des deux corps sont C^k, avec $k \geq 2$, alors la distance est C^k.*

Preuve: On utilise encore une fois l'équation (4.7). □
On notera qu'avoir des surfaces C^1 n'améliore pas la continuité par rapport à des surfaces C^0 comme on peut le voir dans l'annexe B.2. Résumons les propriétés de la distance entre deux objets convexes :

- la distance est toujours C^0 (même sans hypothèse de convexité),
- elle est C^1 par morceau avec convexité large. Les discontinuités surviennent quand des faces ou arêtes sont parallèles (convexité non stricte),
- la stricte convexité d'un corps assure que la distance soit C^1. Les discontinuités d'ordre supérieur apparaissent aux endroits de discontinuité des surfaces,
- un des objets étant strictement convexe, le fait que les deux surfaces soient C^k implique que la distance est C^k.

Ces propriétés sont aussi valables localement. En particulier, si les points témoins se déplacent sur l'intérieur de surfaces C^∞, dont une recouvre un objet strictement convexe, la distance est C^∞. Cela est par exemple le cas quand on considère la distance entre deux sphères.

4.3.3 Interpénétration

FIG. *4.6 – Un cas dégénéré en interpénétration où il existe une infinité de paires de points témoins (dont certaines sont figurées en gris) malgré la stricte convexité.*

La propriété C^n pour $n > 0$ ne peut pas être atteinte partout dans le cas d'une interpénétration : le problème de minimisation lié à la distance n'est plus convexe. Cela implique que pour certaines configurations, il y a plusieurs (jusqu'à une infinité, cf Fig. 4.6)

paires de points témoins. Les sauts de paires sont donc inévitables. On peut cependant garder les résultats des théorèmes précédents pour un sous-ensemble de cas d'interpénétration. Mais il faut d'abord s'assurer de la continuité au passage entre la non-pénétration et l'interpénétration : on définit la distance de pénétration comme l'opposée de la distance entre la paire de points vérifiant la condition d'optimalité $\left(\frac{\partial f}{\partial \mathbf{u}}\right)^T f = 0$ et étant à la distance minimale parmi les différentes paires solutions ayant des vecteurs normaux opposés (ce qui est en définitive la même définition au signe près que pour le cas sans pénétration). Cette définition est équivalente à la distance minimum de translation classique ([van den Bergen 04], chapter 2) puisqu'elle est basée sur les conditions d'optimalité dérivées de la définition classique de la distance. Comme déjà dit, il peut y avoir plusieurs paires possibles et donc des sauts entre ces paires. Ce n'est cependant pas le cas si la distance de pénétration est plus petite que le rayon de courbure minimal des parties des objets en interpénétration. Sous cette hypothèse d'interpénétration légère, les résultats du cas hors pénétration restent valides.
Si cette hypothèse est violée, on peut alors rencontrer des discontinuités de gradient, mais il faut noter que les configurations pour lesquelles ces discontinuités apparaissent sont répulsives : suivre le gradient de tout point à proximité fait s'en éloigner, si bien que de telles configurations ne devraient pas être un problème dans les applications où une pénétration est possible (par exemple, une optimisation commençant avec un point non admissible).

4.4 Volumes englobants STP

Il a été montré dans le paragraphe précédent que les discontinuités apparaissent lorsqu'il y a des zones planes pour chaque objet, ce qui est souvent le cas puisque la plupart des applications travaillent avec des polyèdres, dont les arêtes et les faces ne sont pas strictement convexes.
Dans ce chapitre, nous proposons une méthode pour arrondir ces parties planes, en construisant un volume englobant proche à partir d'un polyèdre convexe afin de le rendre strictement convexe. De manière simplifiée, on associe à chaque primitive géométrique type composant un polyèdre (c'est-à-dire sommet, arête, face) un nouveau type de primitives géométriques (les faces à plus de trois sommets sont découpées en triangles pour la simplicité de cette discussion) :

- à chaque sommet, on associe une *petite sphère* de rayon r,
- chaque face est recouverte d'une partie de *grande sphère* de rayon R qui est tangente aux petites sphères de ses trois sommets,
- enfin, chaque arête est appariée avec une partie de tore dont le rayon intérieur est R et qui se connecte de manière C^1 aux deux grandes sphères correspondant aux deux faces adjacentes.

Cependant, comme il sera expliqué plus loin, l'association n'est pas aussi directe, car, en fonction des valeurs prises par r et R, certains sommets du polyèdre original peuvent être éliminés lors du processus de construction.
On l'aura compris, les nouvelles primitives géométriques sont donc des bouts de sphères et de tores. L'objet obtenu ainsi est appelé *sphere-torus-patch bounding volume* ou plus simplement STP-BV. On appellera *primitive* chaque partie de sphère ou tore qui le compose. r est la distance minimale entre le polyèdre et son STP-BV. C'est une marge de sécurité, en vue de la détection de collision, définie par l'utilisateur. R contrôle le rayon de courbure maximal

du STP-BV, ainsi que la borne supérieure de la distance entre polyèdre et STP-BV comme montré par le théorème 4.4.4. R doit être au moins aussi grand que le rayon du polyèdre c'est-à-dire le rayon de la plus petite sphère contenant le polyèdre, mais devrait être plus grand de plusieurs ordres afin d'approximer finement le polyèdre. Le choix de la valeur est aussi laissé à l'utilisateur. Quand R devient petit, le STP-BV tend vers cette plus petite sphère. Quand $r \to 0$ et $R \to +\infty$, le STP-BV n'est rien d'autre que l'enveloppe convexe, c'est-à-dire le polyèdre original. On peut donc rendre un STP-BV aussi proche que l'on veut d'une enveloppe convexe. Cela se fait au détriment de la régularisation de la distance.

4.4.1 Construction des grandes sphères

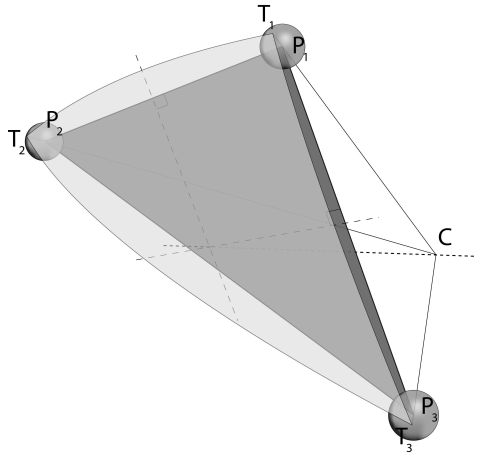


FIG. *4.7 – Construction d'une sphère : la grande sphère est tangente aux petites sphères, ce qui signifie que son centre C est aligné avec chaque P_i et le point de tangence correspondant T_i. C est équidistant aux P_i ce qui implique que sa projection sur le plan de la face est le centre du cercle circonscrit à la face (à l'intersection des médiatrices des arêtes, représentées par les fines lignes pointillées).*

Considérons une face (triangulaire) d'un polyèdre (cf Fig. 4.7). P_1, P_2 et P_3 sont ses sommets, donnés dans l'ordre trigonométrique autour de la normale sortante. T_1, T_2 et T_3 sont les points correspondants où les petites sphères sont tangentes à la grande.
Du fait de cette tangence, C, P_i et T_i sont alignés ($i = 1, 2, 3$). Le problème se réduit alors à trouver la sphère de rayon $R - r$ qui passe par les trois sommets de la face, et se situe *au-dessus* de la face (la direction étant donnée par la normale sortante). Il y a une unique sphère solution à ce problème.
Soit $\mathbf{u} = \overrightarrow{P_1P_2}$, $\mathbf{v} = \overrightarrow{P_1P_3}$, $\mathbf{c} = \overrightarrow{P_1C}$ et $\mathbf{w} = \mathbf{u} \wedge \mathbf{v}$. $\mathbf{w}$ est colinéaire à la normale sortante et de même sens.
C doit être équidistant aux points P_i et donc se trouve sur les plans médiateurs des arêtes. On résout alors le système suivant pour trouver les coordonnées de C dans le repère $(P_1, \mathbf{u}, \mathbf{v}, \mathbf{w})$:

$$\begin{cases} \mathbf{u}.\mathbf{c} &= \mathbf{u}^2/2 & \text{(plan médiateur de } [P_1P_2]) \\ \mathbf{v}.\mathbf{c} &= \mathbf{v}^2/2 & \text{(plan médiateur de } [P_1P_3]) \\ \mathbf{c}^2 &= (R-r)^2 & \\ \mathbf{c}.\mathbf{w} &< 0 & \text{(sphère du dessus, donc centre en dessous)} \end{cases} \tag{4.8}$$

On écrit $c = \alpha\mathbf{u} + \beta\mathbf{v} + \gamma\mathbf{w}$ afin d'utiliser le fait que $\mathbf{w}$ est orthogonal à $\mathbf{u}$ et $\mathbf{v}$ en même

temps.
Les deux premières équations forment alors un système linéaire dont les solutions sont

$$\begin{cases} \alpha &= \dfrac{\mathbf{u}^2\mathbf{v}^2 - \mathbf{u}.\mathbf{v}\cdot\mathbf{v}^2}{2(\mathbf{u}\times\mathbf{v})^2} \\ \beta &= \dfrac{\mathbf{u}^2\mathbf{v}^2 - \mathbf{u}.\mathbf{v}\cdot\mathbf{u}^2}{2(\mathbf{u}\times\mathbf{v})^2} \end{cases}$$

Ce sont les coordonnées du cercle circonscrit au triangle.
En remplaçant α et β dans la troisième équation, on obtient une équation de degré 2 en γ, qui est simplement une façon d'écrire le théorème de Pythagore dans le triangle fait de P_1, C et le centre du cercle circonscrit. Seule une des deux solutions vérifie la quatrième équation :

$$\gamma = -\sqrt{\frac{(R-r)^2 - \alpha^2\mathbf{u}^2 - \beta^2\mathbf{v}^2 - 2\alpha\beta\mathbf{u}.\mathbf{v}}{(\mathbf{u}\wedge\mathbf{v})^2}} \tag{4.9}$$

De la sphère centrée en ce point C, on ne garde pour le STP-BV que la partie à l'intérieur du cône défini par les vecteurs $\overrightarrow{CP_i}$ et de sommet C.

4.4.2 Construction des tores

Grossièrement, on obtient le tore au dessus d'une arête par rotation de la grande sphère d'une face adjacente autour de cette arête et en gardant le volume interne à celui balayé par la sphère. La construction, illustrée sur la figure 4.8, est basée sur le résultat suivant :

Théorème 4.4.1 *La distance entre le centre de la sphère correspondant à une face, et le point milieu d'une de ses arêtes ne dépend que de la longueur l de cette arête et vaut $\sqrt{(R-r)^2 - \frac{l^2}{4}}$*

Preuve: Avec les notations de la figure 4.8, ce résultat provient de l'application du théorème de Pythagore dans le triangle (IC_1P_3). □

Les centres C_1 et C_2 des deux sphères correspondant aux deux faces adjacentes à l'arête que nous considérons sont donc sur un même cercle centré en I et de rayon $\sqrt{(R-r)^2 - \frac{l^2}{4}}$.
Considérons le cercle $\mathcal{C}_1$ de centre C_1 et de rayon R dans le plan défini par C_1 et l'arête. Par construction, ce cercle coïncide avec la sphère centrée en C_1.
En faisant tourner ce cercle autour de l'arête jusqu'à ce qu'il coïncide avec le cercle $\mathcal{C}_2$ de même rayon centré en C_2 et dans le plan défini par C_2 et l'arête, on obtient la part de tore dont nous avons besoin. Il faut remarquer que ce tore n'a pas la forme habituelle d'une bouée puisque son petit rayon habituel est plus grand que son grand rayon (habituel). La partie qui nous intéresse est sur la surface interne du tore entier, comme montré par la figure 4.9.

Le tore et les grandes sphères coïncident au niveau des plans les limitant et sont perpendiculaires à ces plans. Le tore est donc tangent aux deux sphères adjacentes, et la jonction entre le tore et une sphère est alors C^1, ce qui implique que le volume est strictement convexe. Il faut noter que pour avoir cette propriété de jonction, il faut absolument que les rayons de toutes les grandes sphères soient identiques.

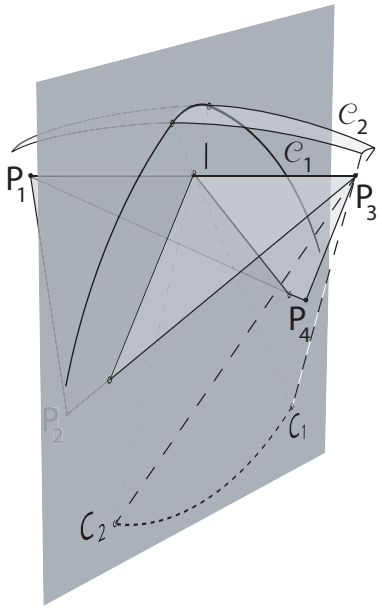


FIG. *4.8 – Construction d'un tore. Le plan gris est le plan médiateur de l'arête* $[P_1P_3]$*, sur lequel sont dessinés ses intersections avec les faces adjacentes à l'arête, les grandes sphères correspondantes et le tore entre les deux. Les cercles* $\mathcal{C}_1$ *et* $\mathcal{C}_2$ *sont les limites entre le tore et les grandes sphères (qui ne sont pas représentées ici) et génèrent le tore par rotation autour de* (P_1P_3)*.*

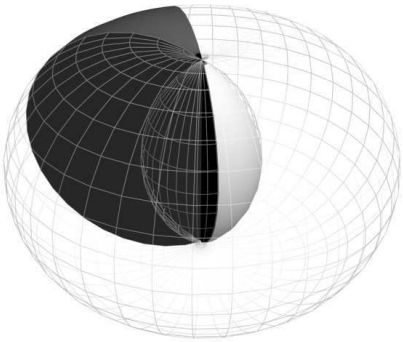


FIG. *4.9 – Les parties noires et blanches forment la portion de tore que nous considérons. Seule la partie blanche est utilisée pour le STP-BV*

4.4.3 Propriétés

Théorème 4.4.2 *Dans un STP-BV, les tores sont tangents aux petites sphères.*

Preuve: Chaque tore est obtenu par la révolution d'un cercle autour d'un axe qui est aussi axe de révolution des petites sphères adjacentes, et ce cercle est tangent, par construction, aux petites sphères. □

Théorème 4.4.3 (Propriétés de continuité et convexité des STP-BV) *Un STP-BV est* C^1 *et strictement convexe.*

Preuve: Toutes les parties d'un STP-BV sont C^1 et strictement convexes, et sont tangentes à leurs jonctions. □

Les STP-BV sont même C^∞ par morceaux, car les tores et les sphères sont des surfaces C^∞.

Théorème 4.4.4 (Marge maximale) *Si a est la longueur de la plus longue arête du polyèdre sous-jacent, la distance maximale entre l'enveloppe convexe du polyèdre et son STP-BV est* $R - \sqrt{(R-r)^2 - \frac{a^2}{12}}$.

Preuve: La marge maximale entre un sommet de l'enveloppe convexe et le STP-BV est r.
Comme une arête est l'axe de révolution du tore qui lui est associé, la marge maximale est atteinte dans son plan médiateur, et est $R - \sqrt{(R-r)^2 - \frac{l^2}{4}}$, où l est la longueur de l'arête.
Pour une face, il faut distinguer deux cas, en fonction de la position de H, centre du cercle circonscrit :

- H est à l'extérieur de la face. La marge maximale est alors atteinte pour la plus longue arête de cette face,
- H est à l'intérieur de la face. Dans ce cas, pour une longueur de la plus longue arête fixée a, la marge maximale est atteinte pour une face équilatérale et est égale à $R - \sqrt{(R-r)^2 - \frac{a^2}{12}}$.

□

A titre d'exemple, la marge maximale est de 1.00208cm pour les valeurs typiques de nos applications ($a = 10$cm, $r = 1$cm et $R = 20$m).

4.4.4 Régions de Voronoi

Les régions de Voronoi sont des données clés pour le calcul de distance avec les méthodes qui exploitent la topologie et la structuration des primitives géométriques de base [Lin 91] [Mirtich 98]. Nous décrivons ici comment ces régions sont délimitées. Les illustrations sont données par les figures 4.10 et 4.11.

4.4.4.1 Limite de Voronoi entre un tore et une petite sphère

Pour plus de clarté et sans perte de généralité, nous nous référerons à la figure 4.8. C_1P_3 est un rayon à la fois de $\mathcal{C}_1$ et de la petite sphère centrée en P_3. De ce fait, dans le plan $C_1P_1P_3$, C_1P_3 est normal au tore et à la petite sphère, et donc est limite de leurs régions de Voronoi. Comme la partie de tore est générée par la révolution du cercle $\mathcal{C}_1$ autour de l'arête P_1P_3, qui est aussi axe de révolution de la petite sphère centrée en P_3, le cône généré par la révolution de C_1P_3 atour de P_1P_3 constitue la frontière entre les régions de Voronoi du tore et de la petite sphère.

4.4.4.2 Limite de Voronoi entre un tore et une grande sphère

Le plan $(C_1P_1P_3)$ contient le centre C_1 de la grande sphère recouvrant la face $P_1P_2P_3$, et est donc perpendiculaire à cette sphère le long de $\mathcal{C}_1$. Le tore est obtenu par rotation de

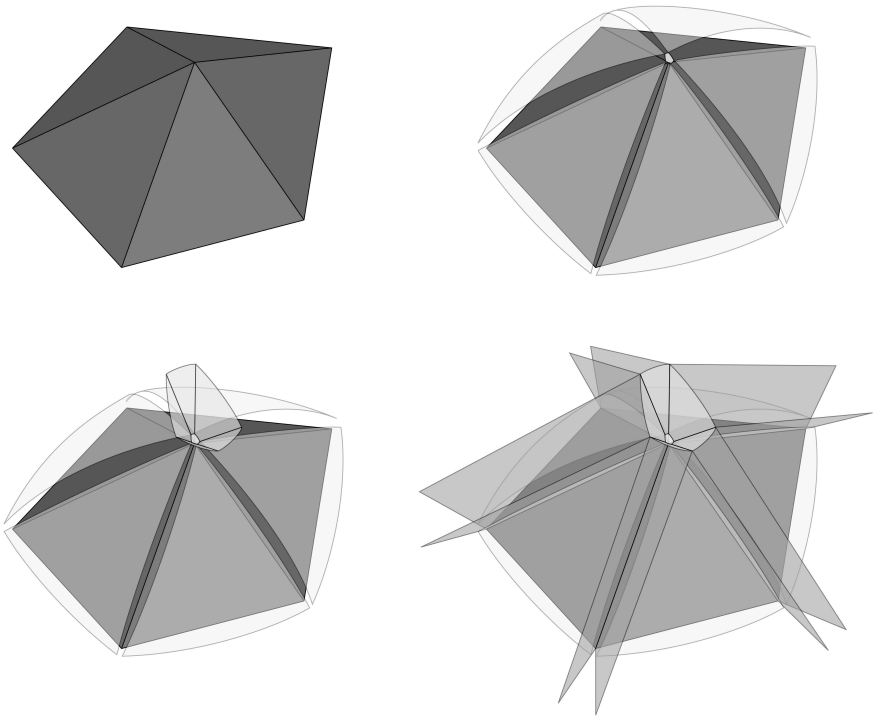


FIG. *4.10 – Régions de Voronoi autour d'un sommet. De gauche à droite et de haut en bas : un sommet et ses faces adjacentes, une petite et des grandes sphères du STP-BV, région de Voronoi de la petite sphère, toutes les limites des régions de Voronoi autour du sommet.*

$\mathcal{C}_1$ autour de (P_1P_3) si bien que $(C_1P_1P_3)$ (qui contient (P_1P_3)) est aussi perpendiculaire au tore le long de $\mathcal{C}_1$. Comme $(C_1P_1P_3)$ est perpendiculaire à la grande sphère et au tore le long de leur limite commune $\mathcal{C}_1$, alors c'est la limite de leurs régions de Voronoi.

4.4.4.3 Région de Voronoi entre une petite et une grande sphère

Il n'y a qu'un seul point commun entre une petite et une grande sphère, celui où elles sont tangentes. La séparation des régions de Voronoi respectives se fait donc par la droite définie par le centre des deux sphères. Cette droite fait aussi parties des de toutes les limites des régions de Voronoi adjacentes.

4.4.5 Remarques sur les STP-BV

Comme il est montré par le théorème 4.3.5, il n'y a pas d'avantage, du point de vue de la continuité de la distance, à avoir des surfaces C^1 plutôt que C^0. Puisque l'enjeu est d'éviter les surfaces planes, nous aurions pu construire un volume englobant uniquement composé de parties de grandes sphères, une au-dessus de chaque polyèdre. La distance aurait alors été quand même C^1, puisque le polyèdre aurait été arrondi.

Il y a deux raisons à notre utilisation des tores et petites sphères : les tores, en arrondissant la jonction entre grandes sphères, permettent au volume englobant de rester très proche de l'objet original (l'utilisation uniquement de sphères dans le cas d'un objet avec un angle très

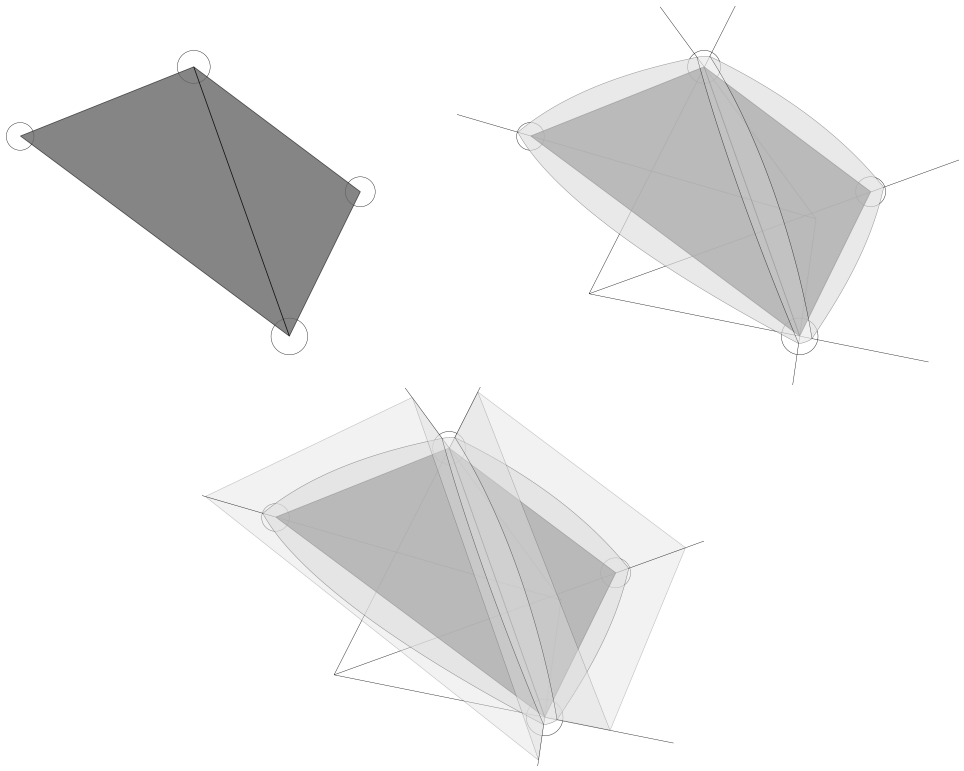


FIG. *4.11 – Régions de Voronoi autour d'une arête. De gauche à droite et de haut en bas : arête avec ses faces adjacentes et les petites sphères correspondant aux sommets, STP-BV, régions de Voronoi des faces.*

aigu entre deux faces peut engendrer une marge maximale allant jusqu'à $2R$). Les petites sphères permettent de visualiser la marge de sécurité. Les utiliser ou ajouter une marge de stabilité a posteriori revient au même en ce qui concerne la distance.

4.5 Calcul de distance entre STP-BV

L'idée principale pour calculer la distance régularisée entre deux objets polyédriques O_1 et O_2 est de s'appuyer sur un algorithme de calcul de distance classique duquel on peut obtenir les primitives géométriques témoins (la paire des primitives - sommet, arête ou face - les plus proches) et les points témoins. Nous ajoutons alors simplement une étape supplémentaire à cet algorithme, qui associe aux primitives témoins du polyèdre celles de son STP-BV.
Avant de faire ce type de calcul, il faut construire les STP-BV, ainsi que certaines données concernant les régions de Voronoi associées. Ces constructions (cf annexe B.3) sont faites hors-ligne pour chaque objet.
Afin d'éviter toute confusion, nous appellerons *primitive polyédrique* une primitive géométrique du polyèdre sous-jacent (i.e. un sommet, une arête ou une face) et *primitive lisse* les primitives du STP-BV (i.e. une partie de petite sphère, grande sphère ou tore).

4.5.1 Algorithme général

Le calcul de la distance régularisée entre deux polyèdres (non nécessairement convexes) O_1 et O_2 se fait en deux étapes, en partant de leurs sommets (cf figure 4.12).

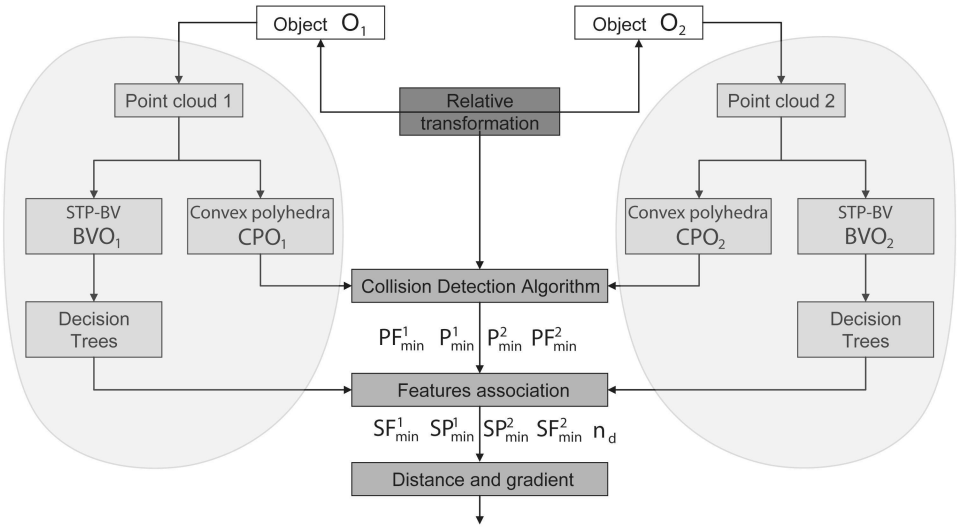


FIG. *4.12 – Algorithme global. Les étapes de la zone grisée sont calculées hors-ligne.*

Pour le calcul de la distance entre une paire d'objets, nous nous sommes inspiré d'un algorithme qui exploite les propriétés structurelles et topologiques de voisinage, à l'instar de celui développé par Lin et Canny [Lin 91] et amélioré par Mirtich [Mirtich 98]. Cependant, pour procéder ainsi, nous avions le choix entre deux façons de faire : (i) à partir des régions de Voronoi des STP-BV, étudier des propriétés qui permettent de dégager une approche semblable à celle de Lin et Canny ou Mirtich, ou (ii) établir une association entre régions des primitives polyédriques avec celles des STP-BV, ce qui impliquerait de se baser sur l'existant et ne pas avoir à reprogrammer tout le logiciel de calcul de distance ainsi que les interfaces développés dans V-Clip [Mirtich 98]. Il suffirait alors d'adjoindre à V-Clip une sorte de *plug-in* (extension) pour les STP-BV. Nous avons opté pour la deuxième solution car le code source du logiciel est à accès libre ; par conséquent l'approche nous semblait moins coûteuse en termes d'investissement logiciel.

Tout d'abord, un calcul hors ligne construit les deux STP-BV BVO_1 et BVO_2, ainsi que les polyèdres convexes sous-jacents CPO_1 et CPO_2, et des données relatives aux régions de Voronoi des STP-BV. Le but de ces données est de précalculer tout ce qui est possible afin de gagner autant que faire se peut sur les calculs en ligne. L'implémentation de cette première partie n'a pas été optimisée puisqu'elle s'effectue hors ligne. La seconde étape est le calcul en ligne : nous faisons tout d'abord appel à un algorithme de calcul de distance classique pour CPO_1 et CPO_2. Il retourne les points témoins $P^1_{\min}$ et $P^2_{\min}$ ainsi que les primitives (polyédriques) les plus proches PF_1 et PF_2. Il faut alors trouver les primitives (géométriques) "lisses" (abus de langage par analogie à *smooth*) les plus proches SF_1 et SF_2 de BVO_1 et BVO_2 à partir de cette sortie. Une fois ces primitives obtenues, on peut calculer la distance $\delta = d(SF_1, SF_2)$, les nouveaux points témoins $SP^1_{\min}$ et $SP^2_{\min}$, et $\mathbf{n}_d$ la normale unitaire de BV_1 au point $SP^1_{\min}$. Ces trois dernières données sont nécessaires pour le calcul

du gradient.

Le calcul de la distance et de ces données peut se faire pour trois différents types de paires de primitives lisses : sphère-sphère, sphère-tore et tore-tore. Le premier cas est trivial, les deux autres sont détaillés au paragraphe 4.5.3.
L'association entre primitives polyédriques et lisses n'est pas immédiate, car les régions de Voronoi de ces primitives ne sont pas équivalentes. L'association se fait sur la base de deux heuristiques :

1. la primitive lisse associée à une primitive polyédrique est soit la primitive lisse qui lui correspond directement (par exemple pour une face, la grande sphère qui lui est appariée), soit une des primitives adjacentes à cette primitive lisse, comme il sera décrit dans les sous-paragraphes suivants,
2. le choix d'une primitive lisse SF_i est basé sur la position de $P^j_{\min}$ vis-à-vis de la région de Voronoi de SF^0_i, la primitive lisse qui est directement attachée à PF_i ($i = 1$ et $j = 2$ ou le contraire).

Par exemple, si PF_1 est un sommet et SF^0_1 la petite sphère qui lui est associée, la position de $P^2_{\min}$ vis-à-vis de la région de Voronoi de SF^0_1 permet de décider si SF_1 est SF^0_1 ou l'un des tores ou grande sphères adjacents.
Comme il est montré dans l'article d'introduction à V-Clip [Mirtich 98], les primitives les plus proches sont atteintes lorsque le point témoin de chaque objet est dans la région de Voronoi de la primitive témoin de l'autre objet. Cette propriété doit être vérifiée par $SP^1_{\min}$, $SP^2_{\min}$, SF_1 et SF_2. Avec les heuristiques ci-dessus, les tests sur des objets tels que ceux utilisés en section 4.6 montrent que l'association faite ne permet pas de vérifier cette propriété dans moins de 0.01% des requêtes de calcul. Dans plus de 99% de ces cas "d'échec", le point témoin se trouve sur une primitive lisse voisine. On s'autorise alors un unique test supplémentaire pour corriger l'erreur. De ce fait l'association est en complexité $\mathcal{O}(1)$.

4.5.2 Association d'une primitive lisse à une primitive polyédrique

4.5.2.1 Association d'une primitive lisse à un sommet

La région de Voronoi de la petite sphère est toujours entièrement à l'intérieur de la région de Voronoi du sommet auquel elle est associée. Si un point témoin polyédrique est à l'intérieur de la région de Voronoi du sommet, alors il peut se trouver soit dans la région de Voronoi de la petite sphère, soit dans une de celles des primitives lisses adjacentes (tores ou grandes sphères des arêtes et faces contenant le sommet).

Le calcul pour faire l'association peut se réduire à un problème en deux dimensions, grâce à la remarque suivante : le sommet appartient à toutes les limites avoisinantes des régions de Voronoi lisses, ainsi qu'à son propre cône de Voronoi. De ce fait, ce sommet peut être considéré comme le point focal d'une projection sur un plan : on choisit un plan au dessus du sommet, dont la normale est à l'intérieur de la région de Voronoi de la petite sphère. La projection des régions de Voronoi lisses sur ce plan correspond à l'intersection de ces régions avec le plan comme montré sur la figure 4.13. L'image de droite de cette figure montre les régions 2D ainsi obtenues, dans lesquelles le point témoin de l'autre objet est projeté. La projection de ce point se trouve à l'intérieur du polygone extérieur, celui-ci étant la projection de la région de Voronoi du sommet. Trouver en 2D dans quelle région la projection du point témoin se

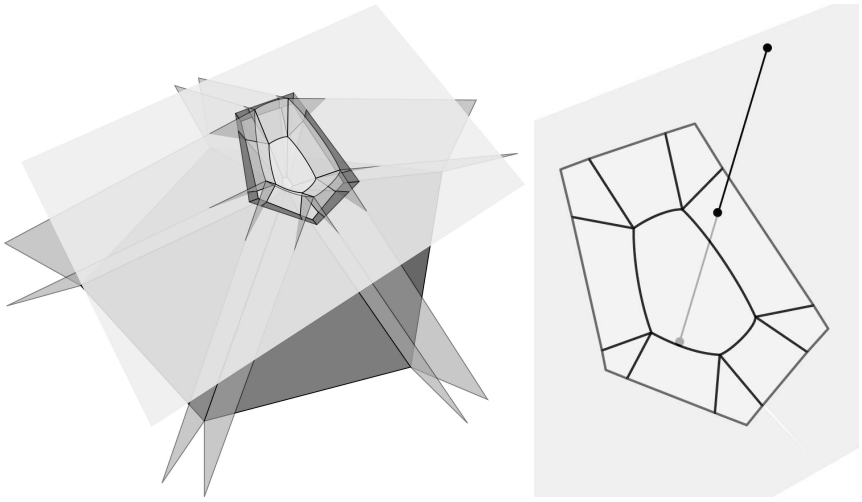


FIG. *4.13 – Intersection d'un plan avec les régions de Voronoi relatives à un sommet. Les régions de Voronoi des primitives lisses sont celles de la figure 4.10, le cône plus foncé décrit la région de Voronoi du sommet.*

trouve est strictement équivalent à trouver dans quelle région de Voronoi lisse ce point se trouve, mais demande moins de calcul. Pour accélérer encore plus les calculs, on construit un arbre de test lors du processus hors-ligne, cela afin de minimiser le nombre moyen de tests à faire pour déterminer une région.

Pour résumer : si PF_i est un sommet, SF_i peut être la petite sphère associée ou une des primitives lisses associées aux arêtes et faces qui contiennent PF_i. $P^j_{\min}$ est projeté sur un plan sur lequel ont déjà été projetées les régions de Voronoi lisses correspondantes. L'endroit où se trouve la projection de $P^j_{\min}$ détermine SF_i.

4.5.2.2 Association d'une primitive lisse à une arête ou une face

Comme la région de Voronoi d'une petite sphère est toujours strictement à l'intérieur de la région de Voronoi du sommet correspondant, les régions de Voronoi des arêtes et des faces ne l'intersectent jamais. Mais les régions de Voronoi des grandes sphères et des faces s'intersectent de plusieurs façons avec les régions des tores et arêtes, selon la forme des faces. Dans tous les cas, les tests à effectuer sont ceux déterminant de quel côté d'un plan limitant des régions de Voronoi lisse du STP-BV se trouve le point témoin.

Lorsqu'une face (triangulaire) n'est pas obtuse, la région de Voronoi lisse associée à la grande sphère correspondante contient sa propre région de Voronoi polyédrique. Le centre de la sphère se projette à l'intérieur de la face, et de ce fait l'angle entre la face et les plans limites de la région de la grande sphère est supérieur à 90 degrés. Il n'y a alors pas de test à faire : la région de Voronoi de la grande sphère contient celle de la face et donc un point témoin dans la région de la face est automatiquement dans celle de la sphère.

Avec la même argumentation, si une arête est entre deux faces non obtuses, ou si elle n'est pas la plus longue arête d'une face obtuse, alors sa région de Voronoi contient celle du tore. Un point dans la région polyédrique peut donc se trouver soit dans la région lisse du tore ou dans celle d'une des deux sphères adjacentes. Il faut donc effectuer un ou deux tests pour savoir de quel coté des plans limites se trouve le point.

Si une face est obtuse, alors sa région de Voronoi polyédrique et la région lisse du tore

associé à sa plus longue arête s'intersectent. Un unique test permet de déterminer si le point se trouve ou non dans la région lisse associée à la grande sphère.

Si une arête est la plus longue d'une face obtuse, alors il n'y a pas de test à faire pour savoir si le point témoin est ou non dans la région de Voronoi lisse de la grande sphère associée à cette sphère.

Un problème survient lorsqu'une face est très fine et qu'autour la forme de l'objet est assez plate : il se peut alors que la région de Voronoi de la sphère associée intersecte plusieurs régions de Voronoi polyédriques en plus de celles envisagées ci-dessus. Cela demanderait donc de faire des tests d'appartenance supplémentaires. Cependant, lorsque le rayon des grandes sphères R est grand par rapport à la taille de l'objet, une telle intersection à peu de chance d'arriver, ou arrivera très loin de l'objet, à une distance où l'erreur engendrée par le fait de ne pas faire ces tests supplémentaires n'aura pas de véritable impact sur la distance ou son gradient. Nous gardons la solution suivante :

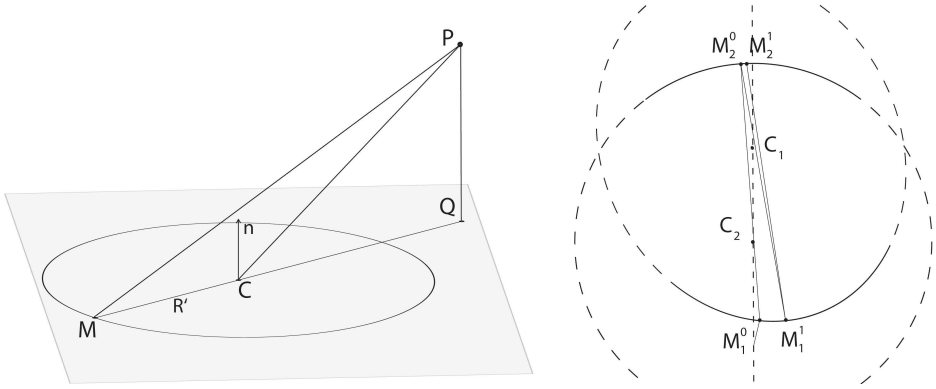
- si PF_i n'est pas une face obtuse, alors SF_i est la grande sphère associée,
- si PF_i est obtuse, il faut alors faire un test pour déterminer si SF_i est la grande sphère ou le tore associé à l'arête la plus longue de PF_i,
- si PF_i est une arête sans être la plus longue arête d'une face obtuse, la position de $P^j_{\min}$ vis-à-vis de deux plans doit être déterminée pour savoir si SF_i est le tore correspondant ou la sphère associée à une des faces adjacentes,
- si PF_i est l'arête la plus longue d'une face, il faut faire un test pour savoir si SF_i est le tore correspondant ou la sphère associée à l'autre face adjacente,
- si PF_i est la plus longue arête de ses deux faces adjacentes, alors SF_i est son tore associé.

4.5.3 Implémentation

Nous basons l'implémentation de notre algorithme sur V-Clip, celui-ci retournant en plus de la distance les points et primitives témoins. Cependant, V-Clip ne prend pas parfaitement en compte les cas de pénétration car il s'arrête dès la première paire de primitives en intersection. Cela suffit dans la plupart des cas pour les pénétrations "légères", mais nous avons rajouté une heuristique pour celles qui sont plus profondes (nous faisons simplement quelques itérations supplémentaires pour essayer d'aller plus profondément dans l'objet).

Distances sphère–tore et tore–tore : Le calcul de la distance entre un tore et une sphère, ou entre deux tores, se réduit respectivement au calcul en 3D de la distance entre un point et un cercle, ou entre deux cercles respectivement. Le premier cas a une solution géométrique simple, il a été prouvé que le second n'a pas de solution analytique [Neff 90]. Une méthode de calcul de distance cercle–cercle efficace et précise a été présentée dans [Vranek 02]. Dans cet article on trouve aussi le calcul point–cercle comme sous problème du calcul de distance cercle–cercle.

Cependant, nous ne pouvons pas utiliser ces algorithmes directement : nous utilisons en effet la partie interne des tores, ce qui demande de calculer la distance maximale à un arc, au lieu de la distance minimale. Pour la distance point-cercle, le point le plus loin est à l'opposé sur le cercle du point le plus proche, il n'y a donc que peu de changement à faire. Avec les notations de la figure 4.14(a), la distance est $\sqrt{R'(R' + 2\,\|CQ\|) + CP^2}$.
Pour la distance maximale cercle-cercle, nous aurions pu utiliser une adaptation de l'algorithme proposé par [Vranek 02] pour trouver un maximum plutôt qu'un minimum, en



(a) point-circle distance (b) circle-circle distance

FIG. *4.14 – Calculs des distances de base.*

s'arrêtant au premier maximum trouvé (puisqu'on n'utilise que des arcs de cercles, il n'y a qu'un seul maximum). Il est cependant apparu que dans notre cas particulier, l'algorithme itératif suivant était plus efficace (cf Fig. 4.14(b)) : on détermine d'abord un point sur le cercle associé à la primitive SF_1 en prenant l'intersection de ce cercle avec la projection de la droite (C_1C_2) sur son plan de support, C_1 et C_2 étant les centres des deux cercles. On choisit l'intersection M_1^0 telle que $C_1C_2.C_1M_1^0 > 0$. A partir de ce point, on calcule alors M_2^0, le point le plus loin sur le second cercle. Puis M_1^1 à partir de M_2^0, et ainsi de suite jusqu'à la convergence.

4.5.4 Calcul du gradient de la distance

Nous avons vu au chapitre 3 (section 3.4.2) comment se calcule le gradient de la distance. Avec les notations courantes, nous avons :

$$\frac{\partial \delta}{\partial \mathbf{q}}(\mathbf{q}) = \mathbf{n}_d^T \left(\frac{\partial SP^1_{\min \in BVO_1}}{\partial \mathbf{q}}(\mathbf{q}) - \frac{\partial SP^2_{\min \in BVO_2}}{\partial \mathbf{q}}(\mathbf{q}) \right) \tag{4.10}$$

$SP^i_{\min \in BVO_i}$ est le point fixe du STP-BV qui coïncide avec $SP^i_{\min}$ à $\mathbf{q}$. Les coordonnées du premier point sont fixes dans le repère de l'objet alors que celles du dernier bougent dans le même objet.
Il est important de noter que pour des raisons de stabilité numérique, nous calculons le vecteur $\mathbf{n}_d$ non pas à partir des points témoins, mais à partir des normales effectives des primitives lisses. En effet lorsqu'il y a contact entre les objets, les points témoins coïncident dans le repère absolu et ne peuvent donc définir un vecteur. Comme dans un processus d'optimisation, les contraintes de collision peuvent empêcher d'atteindre le minimum du critère, un tel cas n'est pas impossible, et nous y avons été confronté lors de génération de postures comme celles présentées à la section 4.6.
Dans certains cas de pénétration cependant, lorsqu'il y a plusieurs paires de points témoins, on ne peut pas définir le vecteur à partir des primitives. Mais dans ces cas dégénérés, le gradient n'est de toutes façons pas défini.
On rappelle que pour un point P de coordonnées fixes (x, y, z) dans le repère local d'un objet

O à une configuration $\mathbf{q}$, tel que $SP^i_{\min \in BVO_i}$, le gradient à la forme suivante :

$$\frac{\partial P}{\partial \mathbf{q}}(\mathbf{q}) = xJ_1(\mathbf{q}) + yJ_2q(\mathbf{q}) + zJ_3(\mathbf{q}) + J_4(\mathbf{q})$$

où les J_i sont les matrices gradients des colonnes de la matrice de transformation du corps, dites *matrice de pré-gradient*, introduites au chapitre 3, section 3.2.3.

4.5.5 Temps de calcul

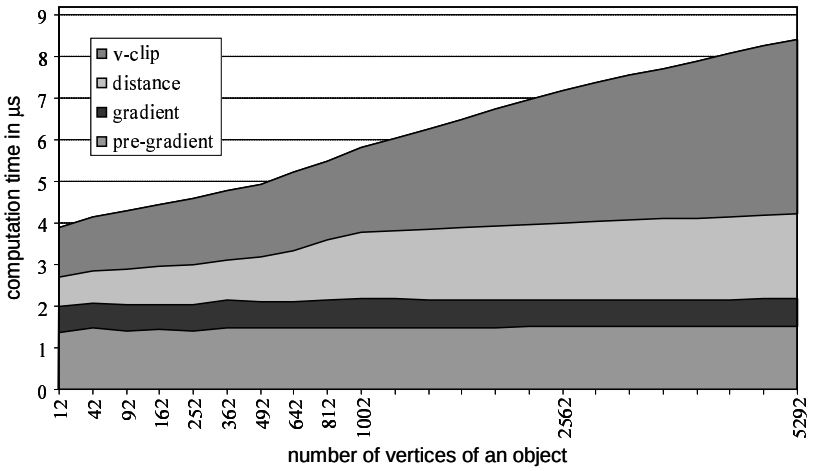


FIG. *4.15 – Temps de calcul pour des petits mouvements relatifs.*
distance *représente le temps de calcul nécessaire pour la surcouche d'association de primitives.* pre-gradient *correspond au calcul des matrices* J_i*, et* gradient *le reste du calcul du gradient, comme présenté à la section 4.5.4.*

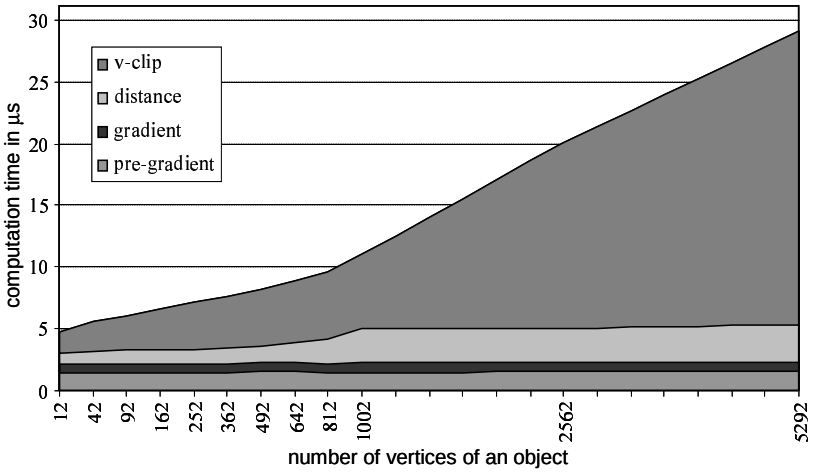


FIG. *4.16 – Temps de calcul pour des grands mouvements relatifs*

Nous avons mesuré le temps de calcul pour différentes tailles de STP-BV et deux amplitudes de mouvements relatifs. La distance a été calculée entre deux sphères géodésiques

(i.e. dont la discrétisation en un maillage est uniforme) de 20cm de rayon, recouvertes d'un STP-BV dont les paramètres sont $r = 1$cm et $R = 10$m. La taille des objets est mesurée par le nombre de leurs sommets (directement relié au nombre de faces : $f = 2(v - 2)$ avec f nombre de faces et v nombre de sommets). Pour chaque taille d'objet, nous avons effectué un million d'appels au calcul de distance et de gradient. Entre chaque appel, les deux objets ont subi une rotation et leur distance relative a été changée dans un intervalle d'amplitude 40cm tel qu'il puisse y avoir des pénétrations légères. Pour les résultats présentés par la figure 4.15, l'angle de rotation était d'environ 2 degrés et la moyenne de déplacement relatif de 2mm. Pour le second graphique (figure 4.16), l'angle de rotation était d'environ 20 degrés et le déplacement de 2cm. Les incréments d'angle ont été choisis de manière à ce qu'une configuration n'apparaisse jamais deux fois.

Dans les deux cas, les résultats sont les temps moyens pour une requête. Comme nous nous y attendions, le coût de V-Clip est sensible à l'amplitude de la variation entre deux configurations consécutives et le coût de la couche du gradient est quasiment constant en fonction de la taille des objets : le calcul des pré-gradients et du gradient se fait en 2 microsecondes (respectivement 1.4 et 0.6). Le calcul de distance, qui correspond principalement en une association de primitives, croit légèrement (presque comme une racine carrée) avec le nombre de sommets. Cette augmentation est certainement due à un plus grand nombre de données attachées aux objets plus complexes, mais elle est faible en comparaison avec celle de V-Clip. Nous n'avons pas jusqu'à présent d'autre hypothèse qu'un dépassement de la mémoire cache pour expliquer l'augmentation rapide entre 812 et 1002 sommets.

Les calculs ont été faits sur un Pentium Xeon 3.4GHz tournant sous Windows XP avec 2Go de mémoire vive.

4.6 Application : génération de posture sans collision pour humanoïde

Le travail présenté dans ce chapitre trouve sa motivation originelle dans le besoin d'intégrer des contraintes de non-collision dans le générateur de posture présenté au chapitre 3. Nous présentons ici quelques résultats de cette intégration. Le critère d'optimisation utilisé pour les générations présentées est le suivant :

$$f(\mathbf{q}) = \sum_{i=1}^{n} \left(q_i - \frac{q_i^- + q_i^+}{4} \right)^2 \tag{4.11}$$

où n est le nombre d'articulations, et q_i^- et q_i^+ sont les limites articulaires de q_i. Ce critère prend pour posture de référence la moyenne entre la configuration avec tous les angles à 0 (cf fig. reffig :hrp2hull) et celle au milieu des limites articulaires.
Le problème de l'évitement de collision dans la génération de posture a été traité dans [AbdelMalek 06] et [Peinado 05]. Les auteurs dans [AbdelMalek 06] génèrent des postures avec un problème d'optimisation ; [Peinado 05] utilise des sphères pour couvrir les objets et définir les contraintes de collision. Le problème de cette méthode est le nombre de sphères qu'il faut considérer pour couvrir un robot en gardant une bonne approximation de la forme, et l'explosion du nombre de paires de sphères qu'il faut prendre en compte. Les auteurs de [Peinado 05] génèrent des postures grâce à la cinématique inverse et des tâches hiérarchisées,

mais la méthode proposée ne garantit pas la continuité pour tous les "observateurs" (objets géométriques simples) qu'ils utilisent. De plus ces observateurs doivent être spécifiés manuellement et dépendent du contexte.
Il existe d'autres approches comme celle de Sentis [Sentis 05] où l'évitement de collision se fait par le biais de champs de potentiels. A l'évidence, assurer la continuité du gradient n'est plus nécessaire si le programme d'optimisation n'a pas besoin des gradients ou de la continuité du gradient comme c'est le cas des méthodes de faisceaux (cf [Bonnans 02]). Néanmoins notre méthode est utilisable au-delà du champ de l'optimisation, dans tout problème où le gradient de la distance doit être continu (e.g. commande réactive).

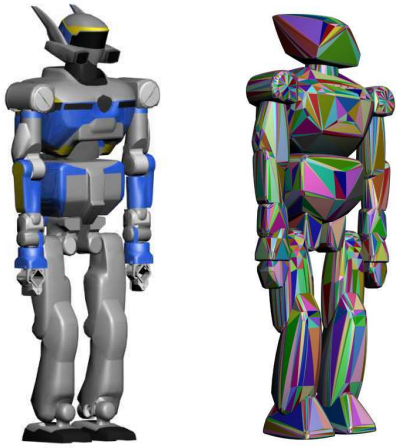


FIG. *4.17 – Le robot HRP-2 et ses STP-BVs*

La figure 4.17 montre le modèle du robot HRP-2 et les STP-BVs qui lui correspondent, calculés hors ligne. Le modèle de chaque corps contient entre 50 et 800 sommets. Les paramètres des STP-BVs sont $r = 1$cm (marge de sécurité pour la collision) et R fixé entre 10m et 100m comme décrit par la table 4.2.

Comment définir les paires de corps dont il faut vérifier la non-collision a fait l'objet de plusieurs travaux dont [Kuffner 02] et plus récemment [Okada 06]. Dans ce dernier article, des tables indexées sont utilisées pour prendre en compte les articulations composées, de façon à ce que les marges de sécurité sur les volumes englobants ne restreignent pas les possibilités de mouvement du robot. Nous nous sommes aussi attaché à ce point, cependant comme il nous faut un gradient continu pour chaque contrainte, l'utilisation de ces tables n'était pas possible. Nous avons donc utilisé des fonctions analytiques spécifiques pour éviter la collision aux hanches, à la taille, au cou et aux épaules. Ces fonctions ont été déterminées soit de manière géométrique, soit par des expérimentations sur le robot réel.
Les obstacles sont eux aussi recouverts de STP-BV et nous définissons les paires corps-obstacle dont il est nécessaire de contraindre la distance.
Pour l'auto-collision, notre problème comporte 117 contraintes dont 7 sont analytiques, les autres étant d'avoir une distance positive entre STP-BV.

Nous avons utilisé la génération de posture avec évitement de collision basé sur les STP-BVs dans trois scénarios. Nous avons étudié l'influence de l'utilisation des STP-BVs pour les deux derniers.

Canette dans un réfrigérateur : on demande au robot d'attraper une canette dans un

FIG. *4.18 – Scénario réfrigérateur. De gauche à droite : illustration du but, posture obtenue sans contrainte d'évitement de collision, posture trouvée avec évitement de collision (vues de côté et de dessus).*

réfrigérateur. Pour cela, les deux pieds sont contraints à un placement donné au sol, tandis que la main gauche doit se trouver autour de la canette, comme montré par la première image de la figure 4.18. Il y a des contraintes d'évitement de collision entre des corps du robot et : les portes du frigidaire, sa cloison gauche, les étagères, la carafe et les clémentines à côté de la canette. En tout, 33 paires de corps. La génération est initialisée avec tous les angles à 0, en face du frigidaire Sur la seconde image, on peut voir le résultat de la génération sans prise en compte des évitements : il y a alors collision entre le genou gauche et la porte du bas, l'avant-bras gauche et la carafe, la poitrine et l'intérieur de la porte du haut, le bras et cette même porte. Les deux images suivantes montrent le résultat lorsqu'il y a évitement de collision.
Dans ce scénario, et encore plus dans le troisième, le robot est assez étiré, et proche de plusieurs limites articulaires à cause des collisions (avec l'environnement dans ce cas, principalement avec lui-même dans le scénario de l'entrepôt). De ce fait l'espace de solution est étroit et le temps de calcul assez important : il est de 0.469 secondes pour ce scénario. Le résultat est obtenu en 74 itérations, avec 11 675 appels au calcul de distance par STP-BV et 9 885 calculs du gradient de cette distance. Sans la vérification des collisions la génération de posture se fait en $32ms$. Le même problème se résoudrait plus rapidement pour un avatar humain grâce à ses nombreux degrés de liberté supplémentaires : le fait qu'il manque un degré de liberté au poignet du HRP-2 est une sérieuse limitation ici et dans le troisième scénario.

Entrée dans une voiture : ici on demande au robot d'avoir les pieds fixés, le gauche à l'extérieur de la voiture, le droit, devant le siège du conducteur. Le robot ne doit compter que sur le pied gauche pour assurer sa stabilité. Il y a 28 paires de collision entre le robot et l'environnement, mettant en jeu le siège du conducteur, la porte, sa vitre, le tableau de bord, le volant et la partie gauche du châssis. Le robot est initialisé dans sa posture 0 et placé au milieu de la voiture (mais sans collision avec les parties de la voiture considérées).
L'image la plus à gauche de la figure 4.19 montre la sortie du générateur lorsque les collisions ne sont pas prises en compte. Dans ce cas, on peut observer une pénétration de la main gauche dans la porte, ainsi qu'une du genou droit dans le tableau de bord. Les trois autres images sont différentes vues de la posture sans collision.

Dans l'entrepôt : HRP-2 est sur une échelle dans un entrepôt et essaie d'attraper une caisse. Pour ce scénario, les pieds sont placés sur la même marche, la main droite agrippe la rambarde et la gauche doit atteindre un point loin devant et à gauche. Les paires de collision

FIG. *4.19 – Scénario voiture. De gauche à droite et de haut en bas : sans contrainte d'évitement de collision, puis trois vues de la posture générée avec ces contraintes.*

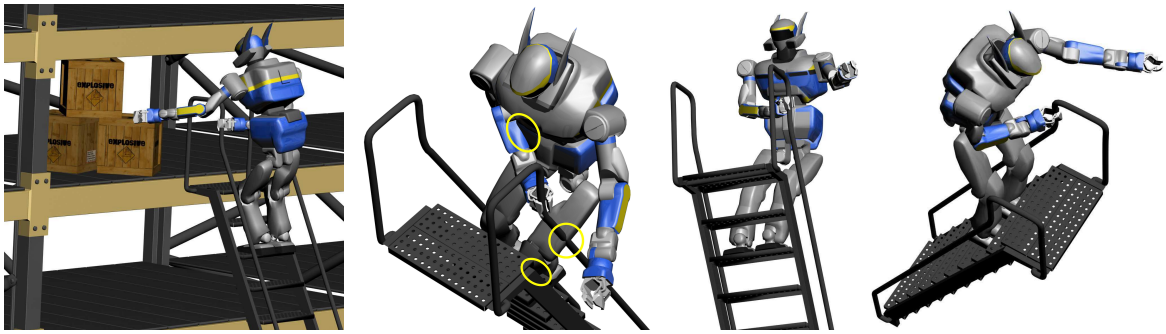


FIG. *4.20 – Scénario entrepôt. De gauche à droite : vue générale du contexte et du résultat, sans contrainte d'évitement de collision, deux points de vue de la posture trouvée avec évitement de collision.*

mettent en jeu les corps du robot avec toutes les parties nécessaires de l'échelle (pas les marches derrière le robot par exemple). La posture d'initialisation est en face de l'échelle, les angles à 0. Ce scénario a été crée pour montrer des auto-collisions, comme on peut en voir sur la seconde image de la figure 4.20. Ces collisions sont évitées dans la posture finale montrée par les trois autres images.

La table 4.2 présente les résultats des générations pour les scénarios *voiture* et *entrepôt*. Dans chaque cas, les résultats sont données avec et sans l'utilisation des STP-BVs pour le calcul de la distance, et pour le second, se déclinent pour plusieurs rayons de grandes sphères R.

Dans le premier scénario, les collisions ne sont pas vraiment un problème et donc la différence due a l'utilisation des STP-BV est faible. On peut quand même observer une augmentation de

	Car		At-Work				
R (m)	20	pas de VE	10	20	50	100	pas de VE
paires de corps	145		132				
durée (s)	0.063	0.064	0.562	0.5	0.67	5.016	35.688
itérations	23	25	103	82	113	393	1745
appel à la distance	2898	3194	14073	11497	14693	88505	268544
appel au gradient	2882	3144	12750	10125	14000	49000	218000
valeur du critère	3.540	3.533	3.542	3.534	3.532	3.566	6.037

TAB. *4.2 – Résultats expérimentaux pour les deux derniers scénarios avec et sans (pas de VE) STP-BVs et pour différents rayons des grandes sphères.*

10% du nombre d'itérations et d'appels à la fonction distance ou son gradient. Dans le second scénario, en revanche, les collisions jouent un rôle beaucoup plus important. L'utilisation des STP-BV améliore alors très fortement le calcul. Dans le cas sans STP-BV, la génération ne converge en plus pas vers le minimum, bien que pensant y être arrivé (car le critère d'arrêt se base sur le fait que les contraintes fournies ont un gradient continu). Le résultat est loin de ce qui est obtenu avec les VE, comme le montre la valeur de la fonction objectif, et la génération s'arrête après bien plus de temps et d'itérations qu'elle ne le fait avec un STP-BV. La table 4.2 montre aussi les effets du rayon des grandes sphères : le temps de calcul diminue de manière générale avec le rayon des grandes sphères. Avec la diminution du rayon R, le gradient est de plus en plus régulier, améliorant le conditionnement du problème. Cela se fait par contre à l'encontre de la qualité de l'approximation. Avec un rayon de $10m$ cependant, les résultats ne semblent pas être aussi bons qu'avec un de $20m$. Nous pensons que avec des petits rayons, quelques erreurs sont introduites par nos heuristiques, ce qui perturbe l'optimisation.

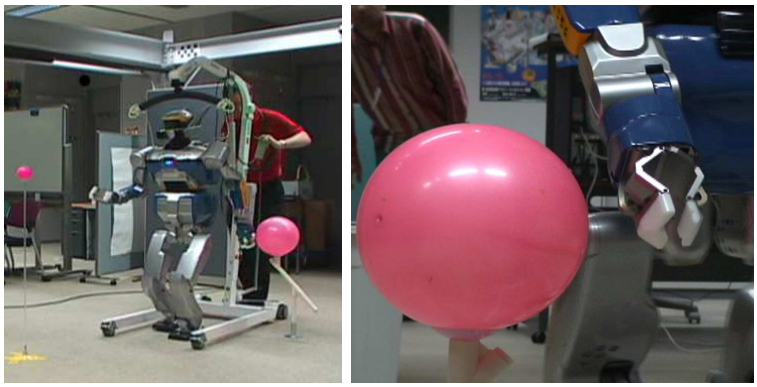


FIG. *4.21 – Évitement de collision avec des STP-BV inclus dans un schéma de contrôle : HRP-2 évite les auto-collisions et la grande balle tout en essayant d'attraper la petite.*

Nous avons aussi intégré les STP-BV pour garantir la continuité de la loi de contrôle évitant les collisions basée sur une pile de tâches [Mansard 07] [Escande 08b] [Escande 08a] (cf Fig. 4.21). Les résultats sont détaillés dans [Stasse 08].

4.7 Discussion et limitations

Avec l'implémentation actuelle, les deux objets doivent être recouverts par un STP-BV. D'après le théorème 4.3.3, cela n'est cependant pas nécessaire : on obtiendrait une distance encore C^1 avec un des deux objets ayant une forme convexe quelconque (par exemple un polyèdre) tant que l'autre est un STP-BV. Cela permettrait de n'avoir des STP-BV que pour le robot, tandis que les objets de l'environnement pourraient être de simples polyèdres. On n'aurait alors pas besoin des pré-calculs de construction des STP-BV pour changer l'environnement, ce qui est particulièrement appréciable si on souhaite faire des changements dynamiques de celui-ci. Cela ne réglera cependant pas des problèmes comme ceux liés aux objets déformables.

Une autre limitation des STP-BV est de ne pas pouvoir approcher correctement les objets non convexes. On peut utiliser des STP-BV pour les objets non-convexes simples, coupés en sous-parties convexes. Mais la généralisation pour des objets plus complexes est un problème ouvert.

Il apparaît que la résolution de ces problèmes est importante dans le contexte de la robotique, car cela ouvre les portes à la conception de lois de contrôle robustes intégrant la distance. La continuité de la distance peut être utilisée pour des tâches autres que l'évitement de collision, comme par exemple des mouvements d'exploration le long d'un objet ou un obstacle. Là encore, la nécessité de ne pas avoir à utiliser des STP-BV pour tous les objets se fait sentir. Nous y travaillons activement avec une approche de calcul basée sur des simplexes, les résultats que nous avons obtenus ont été soumis [Benallegue 09].

4.8 Conclusion

Nous avons mis en évidence ici que la convexité simple de deux objets ne suffit pas à assurer la continuité du gradient de la distance qui les sépare. Cette discontinuité peut s'avérer problématique pour la prise en compte des contraintes d'évitement de collision dans la génération de posture ou les méthodes de contrôle basées sur des optimisations. Nous avons alors montré que la stricte convexité d'un objet suffit à assurer la continuité du gradient dès lors que l'autre objet est convexe, et nous avons proposé un nouveau type de volume englobant strictement convexe nommé STP-BV. Celui-ci se base sur un assemblage de parties de sphères et de tores et peut être rendu aussi proche que voulu de l'enveloppe convexe de l'objet englobé. Nous avons décrit une méthode de calcul de distance entre deux STP-BV qui est une surcouche sur le calcul de distance entre les enveloppes convexes sous-jacentes, et de ce fait donne des temps de calcul très peu supérieurs aux algorithmes de l'état de l'art. Nous avons ensuite montré l'utilisation des STP-BV dans plusieurs cas.
Avec les STP-BV, nous disposons de tous les outils nécessaires pour la génération de posture. Nous pouvons maintenant passer à la planification, qui va s'en servir de manière intensive.

Chapitre 5

Planification d'appuis : algorithmes de base

Nous avons détaillé dans les chapitres précédents des outils qui nous serviront pour la planification. En particulier, nous sommes capables de trouver une posture, quand elle existe, sur un sous-espace $\mathcal{C}_0$ de l'espace de configuration $\mathcal{C}$. Dans ce chapitre, nous nous attaquons à la planification d'appuis à proprement parler. Nous commençons par décrire un principe de planification classique (§ 5.2) basé sur la notion de champ de potentiel (§ 5.1), dont nous nous inspirons ensuite pour construire un planificateur de contact. Nous en décrivons ici les bases et les modules (§ 5.3), laissant l'approfondissement de ces modules pour le chapitre suivant. Enfin nous présentons quelques résultats obtenus avec une première implémentation simple de ces modules (§ 5.4).

5.1 Planification par champs de potentiel

5.1.1 Principe

La notion de champ de potentiel a été initialement proposée par O. Kathib [Khatib 78], dans le cadre du contrôle temps réel d'un robot, afin d'introduire la notion d'obstacle et l'évitement de ceux-ci directement au niveau de la commande [Khatib 80] [Khatib 86]. L'idée est de soumettre le robot à un champ de potentiel artificiel U qui le guide vers son objectif tout en lui faisant éviter les obstacles, l'objectif se situant au minimum global du champ. Pour ce faire, il est classique de construire U en deux parties : un champ attracteur U_a qui attire le robot vers son objectif, et un répulsif U_r l'éloignant des obstacles (cf Fig. 5.1). Dans le cadre de la planification, le potentiel est défini sur l'espace de configuration $\mathcal{C}$ et le robot est une masse ponctuelle dans ce champ.
Quel que soit le potentiel U choisi, le principe est alors de considérer le robot comme étant

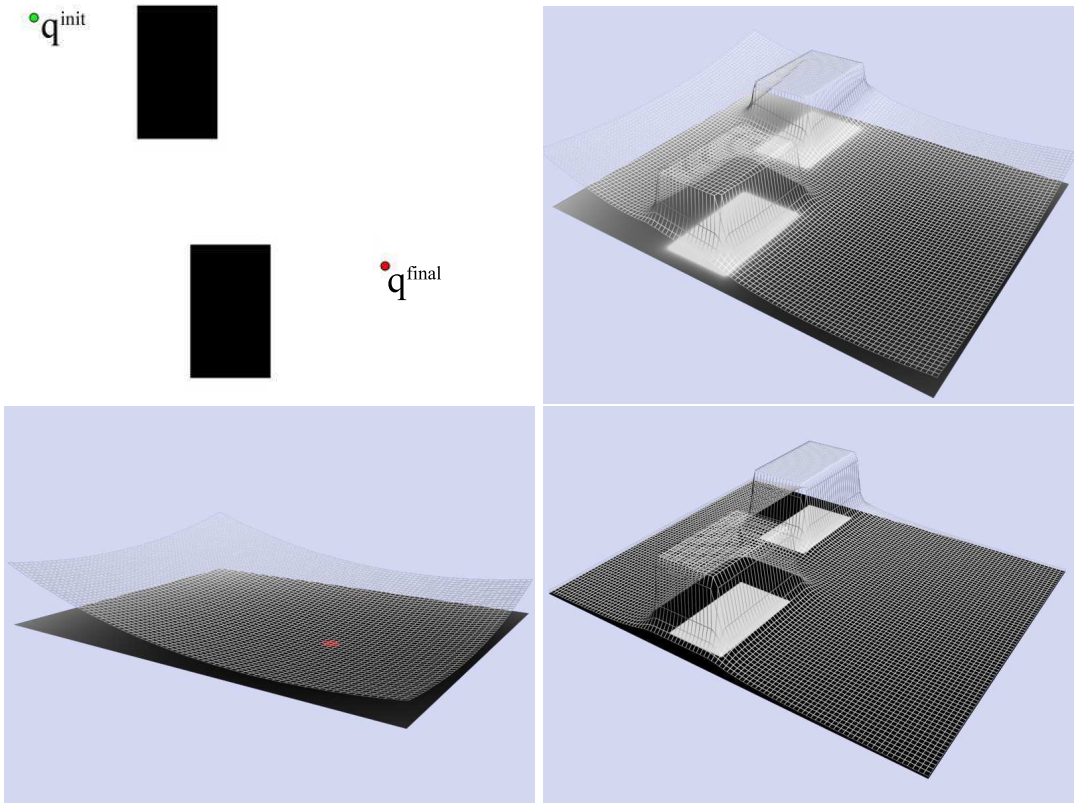


FIG. *5.1 – Construction d'un champ de potentiel dans le cas d'une planification en 2D. On cherche un chemin de* $\mathbf{q}^{init}$ *vers* $\mathbf{q}^{final}$ *en évitant les* $\mathcal{C}$*-obstacles illustrés en noir (en haut à gauche). Pour ce faire, on construit un potentiel attracteur* U_a *(en bas à gauche) dont le minimum se situe en* $\mathbf{q}^{final}$ *(point rouge), et un répulsif* U_r *(en bas à droite). La somme* U *(en haut à droite) servira à une planification par descente de gradient.*

soumis à la force

$$\mathbf{F}(\mathbf{q}) = -\frac{\partial U(\mathbf{q})}{\partial \mathbf{q}} \tag{5.1}$$

Cette force donne la direction du mouvement à chaque instant, selon laquelle le robot se déplace d'une distance proportionnelle à son intensité :

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \delta \mathbf{F}(\mathbf{q}_t) \tag{5.2}$$

δ contrôle la vitesse de déplacement du robot. δ doit être suffisamment grand pour que le mouvement ne soit pas trop lent, mais en même temps doit être assez petit afin de rester dans une zone de $\mathcal{C}$ où le potentiel n'a pas trop varié et donc où $\mathbf{F}(\mathbf{q}_t)$ a encore un sens, et assez petit pour ne pas entrer en collision avec un obstacle en allant trop loin dans une direction.

Ce schéma réplique la méthode de *descente de plus forte pente* (ou *descente de gradient*) connue en optimisation numérique. Le chemin trouvé par cette méthode dans le cadre du scénario présenté figure 5.1 est montré par la figure 5.2. Un tel choix de méthode se justifie pour les besoins d'une commande temps réel et réactive (les obstacles ne sont pas forcément connus à l'avance mais peuvent être rajoutés au champ de potentiel au fur et à mesure de leur détection). Cependant, comme toute méthode de descente de gradient, elle ne garantit

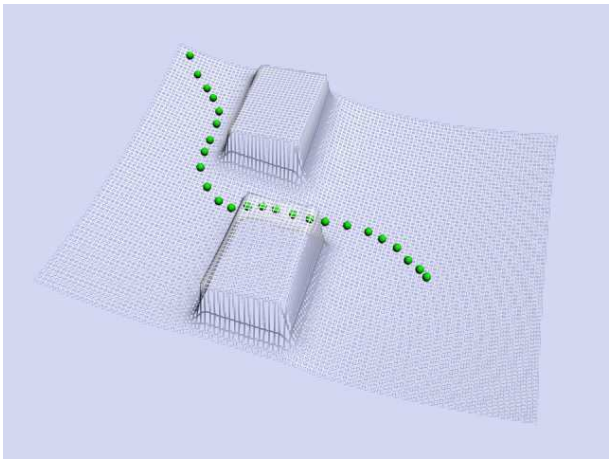


FIG. *5.2 – Chemin obtenu grâce au champ de potentiel construit à la figure 5.1 et en utilisant une méthode de plus forte pente.*

pas l'atteinte du minimum global du potentiel, car elle se fait piéger par les minimums locaux quand elle tombe dans leur bassin d'attraction.

La planification intervient à ce niveau. D'une part, si on a une connaissance parfaite de l'environnement, on peut travailler la formulation du champ de potentiel, d'autre part, en combinant des phases "d'optimisation" avec des techniques de recherche dans un graphe, on obtient des méthodes garantissant d'arriver au but. Nous détaillons au paragraphe 5.2 un tel algorithme.

5.1.2 Potentiel attracteur

La construction du champ de potentiel est particulièrement importante tant en contrôle qu'en planification. Bien que cette dernière offre des outils pour gérer les minimums locaux, ceux-ci restent une source importante de perte d'efficacité. Il faut donc chercher à construire des champs avec aussi peu de minimums que possible. La différentiabilité de tels champs est aussi un point qu'il ne faut pas négliger. Elle peut en effet influer sur la stabilité des algorithmes utilisés. Nous n'entrerons pas dans cette thèse dans les détails des constructions possibles des champs de potentiel. Le lecteur intéressé trouvera dans [Latombe 91] un très bon chapitre couvrant la planification par champ de potentiel. Nous discutons cependant brièvement ici d'une construction possible de champ attracteur que nous adapterons au chapitre 6 pour le suivi d'une trajectoire globale.

Pour avoir un potentiel ayant son minimum en un point $\mathbf{q}^{\text{final}}$ de $\mathcal{C}$, une idée simple et efficace est de baser ce potentiel sur la distance euclidienne à ce point. Deux manières s'imposent alors : utiliser la distance elle-même ou son carré. Les deux méthodes ont des avantages et des inconvénients.
On définit le potentiel attracteur linéaire

$$U_l(\mathbf{q}) = \|\mathbf{q} - \mathbf{q}^{\text{final}}\| \tag{5.3}$$

La force qui en résulte est

$$\begin{aligned}
\mathbf{F}_l(\mathbf{q}) &= -\frac{\partial}{\partial \mathbf{q}} \sqrt{\sum \left(\mathbf{q}_i - \mathbf{q}_i^{\text{final}}\right)^2} && (5.4)\\
&= -\frac{2\left(\mathbf{q} - \mathbf{q}^{\text{final}}\right)}{2\sqrt{\sum \left(\mathbf{q}_i - \mathbf{q}_i^{\text{final}}\right)^2}} && (5.5)\\
&= -\frac{\mathbf{q} - \mathbf{q}^{\text{final}}}{\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\|} && (5.6)
\end{aligned}$$

$\mathbf{F}_l$ est, en tout point $\mathbf{q}$ sauf $\mathbf{q}^{\text{final}}$, le vecteur unitaire d'origine $\mathbf{q}$ et pointant vers $\mathbf{q}^{\text{final}}$. La force est singulière en $\mathbf{q}^{\text{final}}$. Ces propriétés peuvent induire des problèmes de stabilité lorsqu'on arrive à proximité du but.
On peut corriger cela grâce au potentiel attracteur quadratique

$$U_q(\mathbf{q}) = \frac{1}{2}\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\|^2 \tag{5.7}$$

dont la force induite est

$$\begin{aligned}
\mathbf{F}_q(\mathbf{q}) &= -\frac{\partial}{\partial \mathbf{q}}\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\|^2 && (5.8)\\
&= -\frac{1}{2}\cdot 2 \cdot \left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\| \frac{\partial}{\partial \mathbf{q}}\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\| && (5.9)\\
&= -\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\| \frac{\mathbf{q} - \mathbf{q}^{\text{final}}}{\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\|} && (5.10)\\
&= \mathbf{q}^{\text{final}} - \mathbf{q} && (5.11)
\end{aligned}$$

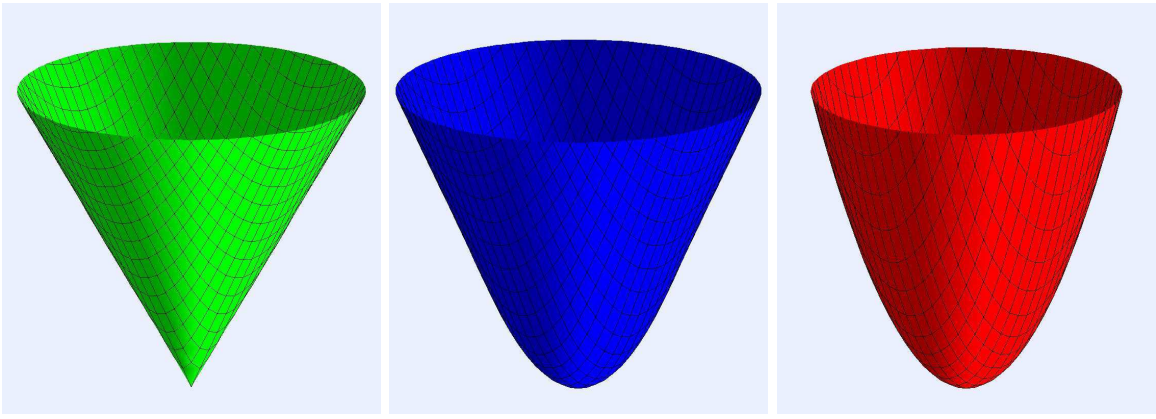


FIG. 5.3 – *Les trois types de potentiels attracteurs à la même échelle : (de gauche à droite) linéaire, mixte, quadratique*

Si cette force pointe toujours bien vers le but, sa norme, en revanche, tend vers 0 quand on s'en approche, ce qui permet la régularité de la force en $\mathbf{q}^{\text{final}}$. Ce potentiel a cependant l'inconvénient de croître vite à mesure que l'on s'éloigne du but, ce qui peut induire des mouvements trop rapides, à l'inverse de U_l dont la force induite reste de norme constante.

On construit alors un potentiel mixte pour tirer parti des avantages complémentaires de U_l et U_q (cf Fig 5.3) :

$$U_m(\mathbf{q}) = \begin{cases} aU_q(\mathbf{q}) & \text{si } \|\mathbf{q} - \mathbf{q}^{\text{final}}\| \leq d_0 \\ bU_l(\mathbf{q}) + c & \text{si } \|\mathbf{q} - \mathbf{q}^{\text{final}}\| > d_0 \end{cases} \tag{5.12}$$

ou d_0 est une distance.
Pour assurer la continuité du champ et de son gradient, il faut qu'en $\mathbf{q}$, tel que $\|\mathbf{q} - \mathbf{q}^{\text{final}}\| = d_0$, on ait

$$\begin{cases} aU_q(\mathbf{q}) = bU_l(\mathbf{q}) + c \\ a\frac{\partial}{\partial \mathbf{q}}U_q(\mathbf{q}) = b\frac{\partial}{\partial \mathbf{q}}U_l(\mathbf{q}) \end{cases} \tag{5.13}$$

soit

$$\begin{cases} \frac{1}{2}ad_0^2 = bd_0 + c \\ a\left(\mathbf{q} - \mathbf{q}^{\text{final}}\right) = b\frac{\mathbf{q}-\mathbf{q}^{\text{final}}}{d_0} \end{cases} \tag{5.14}$$

$$\Leftrightarrow \quad \begin{cases} b = ad_0 \\ c = -\frac{1}{2}ad_0^2 \end{cases} \tag{5.15}$$

d'où

$$U_m(\mathbf{q}) = \begin{cases} a\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\|^2 & \text{si } \|\mathbf{q} - \mathbf{q}^{\text{final}}\| \leq d_0 \\ ad_0\left\|\mathbf{q} - \mathbf{q}^{\text{final}}\right\| - \frac{1}{2}ad_0^2 & \text{si } \|\mathbf{q} - \mathbf{q}^{\text{final}}\| > d_0 \end{cases} \tag{5.16}$$

Nous gardons a comme facteur d'échelle pour les cas où ce potentiel sera additionné avec d'autres termes.

Signalons à titre d'exemple le type de champ de potentiel répulsif utilisé pour la figure 5.1 : pour chaque obstacle i, on définit

$$U_{r_i}(\mathbf{q}) = \begin{cases} \frac{1}{2}\lambda\left(\frac{1}{d_i(\mathbf{q})} - \frac{1}{d_0}\right)^2 & \text{si } d_i(\mathbf{q}) \leq 0 \\ 0 & \text{si } d_i(\mathbf{q}) > 0 \end{cases} \tag{5.17}$$

où $d_i(\mathbf{q})$ est la distance entre le robot à la configuration $\mathbf{q}$ et l'obstacle. Le potentiel répulsif total s'obtient par somme de ces potentiels : $U_r = \sum U_{r_i}$. Les obstacles peuvent cependant être pris en compte de manière différente, par exemple sous la forme de contraintes sur la vitesse normale à chaque obstacle [Faverjon 87].

5.2 Best First Planning

Il existe un certain nombre de méthodes de planification par champ de potentiel, depuis la simple descente selon la plus grande pente jusqu'aux premiers algorithmes faisant appel à de l'aléatoire (Randomized Path Planning, ou RPP, cf [Barraquand 91]). Nous décrivons dans cette partie l'algorithme dit *Best First Planning* (BFP).

5.2.1 Algorithme

Le principe du Best First Planning est de suivre tant que cela est possible la direction de plus grande pente. Si jamais cela fait arriver la planification dans un minimum local, alors l'algorithme cherche à en sortir en remplissant ce minimum jusqu'à arriver à un col d'où il peut recommencer à descendre. On peut illustrer cela en disant que l'algorithme consiste à faire couler de l'eau depuis le point de départ. Celle-ci s'écoule vers le minimum du premier bassin d'attraction qu'elle rencontre. Si ce n'est pas le minimum recherché, on laisse couler jusqu'à déborder du bassin d'attraction. L'eau se dirigera alors vers un autre minimum, et ainsi de suite jusqu'à arriver au but. On regarde alors le chemin parcouru par l'eau. Plus un bassin d'attraction est large, plus il sera long à remplir.
L'algorithme travaille sur une discrétisation de l'espace de configuration. Cette discrétisation est a priori régulière : on s'intéresse aux points d'une grille $\mathbf{q}_{k_1,\cdots,k_{n+6}} = \mathbf{q}_0 + (k_1, \cdots k_{n+6})\delta_q$ où $\mathbf{q}_0$ est l'origine de la grille, $(k_1, \cdots, k_{n+6}) \in \mathbb{Z}^{n+6}$ est le vecteur des coordonnées entières et $\delta_q = diag([\delta_1, \cdots, \delta_{n+6}])$ est une matrice diagonale dont la diagonale est le vecteur des pas de subdivision sur chaque dimension.

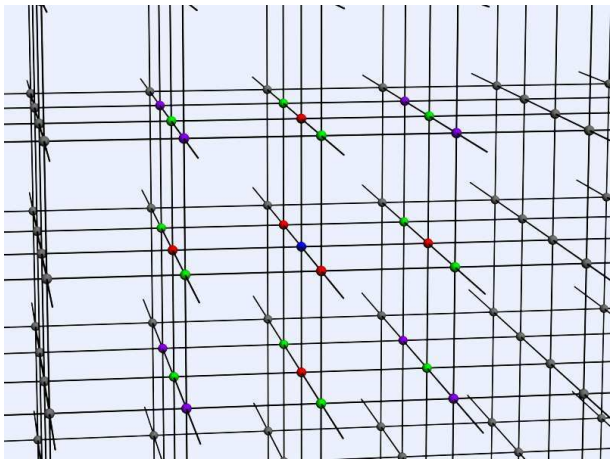


FIG. *5.4 – m-voisins en dimension 3. Si on considère le point bleu, ses 1-voisins sont les points rouges, ses 2-voisins les rouges et verts, ses 3-voisins les rouges, verts et violets*

Sur cette grille on a la définition suivante :
étant donné une configuration $\mathbf{q}_{k_1,\cdots,k_{n+6}}$ de la grille on appelle *m-voisin* $(1 \leq m \leq n+6)$ toute configuration de cette grille qui diffère pour au plus m coordonnées de $\mathbf{q}_{k_1,\cdots,k_{n+6}}$, la différence pour chaque coordonnée étant exactement 1 en coordonnées entières (cf Fig. 5.4). m étant fixé (généralement à 1) on utilisera par la suite le terme voisin pour m-voisin.

Le pseudo-code du BFP est donné par l'algorithme 1. Le principe de cet algorithme est de construire de manière graduelle un arbre ayant pour racine la configuration initiale (que l'on suppose être, tout comme la configuration finale, placée à un nœud de la grille). Pour ce faire, à chaque itération, il considère la meilleure feuille de l'arbre, et construit ses voisins non déjà visités (i.e. non déjà dans l'arbre) et de potentiel inférieur à un seuil M, cela jusqu'à avoir atteint la configuration finale ou ne plus trouver de voisin dont le potentiel soit en dessous de M. Les voisins construits deviennent de nouvelles feuilles tandis que la feuille choisie devient un nœud de l'arbre.
Ce principe de regarder *toutes* les feuilles de l'arbre pour prendre la meilleure permet de

Algorithm:BFP
Data: $\mathbf{q}^{\text{init}}$, $\mathbf{q}^{\text{final}}$,$U(\mathbf{q})$
Result: une suite $\left[\mathbf{q}^0, \mathbf{q}^1, \cdots, \mathbf{q}^l\right]$, avec $\mathbf{q}^0 = \mathbf{q}^{\text{init}}$, $\mathbf{q}^l = \mathbf{q}^{\text{final}}$ et $\mathbf{q}^{i+1}$ voisin de $\mathbf{q}^i$ pour $0 \leq i < l$

```
- L est une liste triée, initialement vide
- T un arbre, initialement vide
begin
    insert(q^init,L)
    add(q^init,T)
    while no isEmpty(L) do
        q ← first(L)
        for each neighbour q' of q do
            if U(q') < M and no isIn(q',T) then
                setFather(q', q)
                add(q',T)
                insert(q',L)
                if q' = q^final then
                    return backtrackPath(q')
                end
            end
        end
    end
    return failure
end
```

Algorithm 1: Algorithme BFP

revenir en arrière quand l'algorithme tombe dans un minimum local.
Pour cet algorithme, chaque configuration est associée à un pointeur vers la configuration précédente dans l'arbre (son père). L'initialisation de ce pointeur se fait par le biais de la fonction $setFather(f, n)$ qui assigne le nœud n comme père de la feuille f.
L est une liste des feuilles de l'arbre, triées par ordre croissant de leur potentiel, pour laquelle trois fonctions sont définies :

- $insert(\mathbf{q},L)$ rajoute la feuille liée à la configuration $\mathbf{q}$ à la liste L, à la place correspondant à son potentiel,
- $first(L)$ retourne le premier élément de la liste (celui de plus faible potentiel) et le retire de celle-ci,
- $isEmpty(L)$ retourne *vrai* lorsque la liste est vide, *faux* sinon.

T représente un arbre. La structure supporte deux opérations :

- $add(\mathbf{q},T)$ ajoute l'élément $\mathbf{q}$ à l'arbre T,
- $isIn(\mathbf{q},T)$ retourne *vrai* si il existe dans l'arbre T un nœud ou une feuille dont la configuration est $\mathbf{q}$, *faux* sinon.

Enfin $backtrackPath(\mathbf{q})$ remonte la filiation de $\mathbf{q}$ à $\mathbf{q}^{\text{init}}$ puis retourne le chemin de $\mathbf{q}^{\text{init}}$ à $\mathbf{q}$

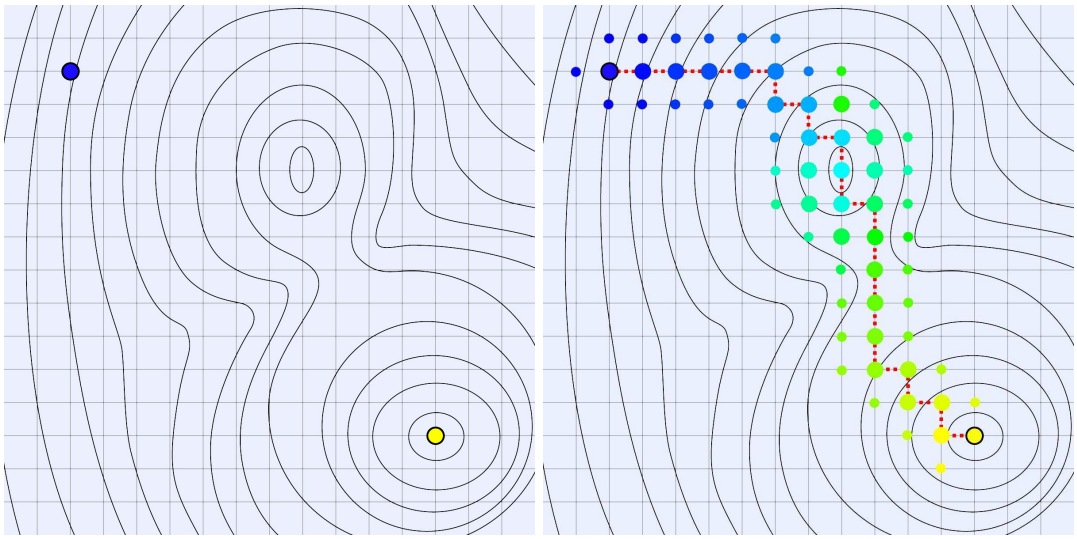


FIG. *5.5 – Exemple de champ de potentiel pour une planification du nœud bleu au nœud jaune et résultat de cette planification (chemin en pointillés rouges).*

La figure 5.5 donne un exemple de planification pour un espace de configuration à deux dimensions. Les lignes de niveau du champ de potentiel sont tracées. Ce champ fait apparaître deux minimums, afin de montrer le principe d'évasion d'un minimum local. Le chemin résultat, ainsi que tous les nœuds et feuilles considérés, sont présentés sur la figure de droite. Les pastilles larges représentent des nœuds de l'arbre (points de la grille dont on a regardé les voisins), les petites, des feuilles. Le gradient de couleur depuis le bleu jusqu'au jaune en passant par le vert indique l'ordre de visite des feuilles et nœuds. Cet ordre est détaillé par les images de la figure 5.6 : en partant du nœud initial, on regarde ses quatre (1-)voisins. Celui de droite étant le meilleur, on passe à lui et regarde ses trois voisins non visités, et ainsi

de suite jusqu'à arriver au minimum local (image en haut à droite). La série de huit images suivante montre le remplissage du minimum : tous les points de la grille dont le potentiel est inférieur à celui se trouvant au niveau du col sont visités un à un, par ordre de potentiel croissant. Le minimum est enfin complètement rempli (en bas à gauche) et l'algorithme peut alors passer le col (bas, milieu) pour aller jusqu'à la configuration finale (bas, droite).

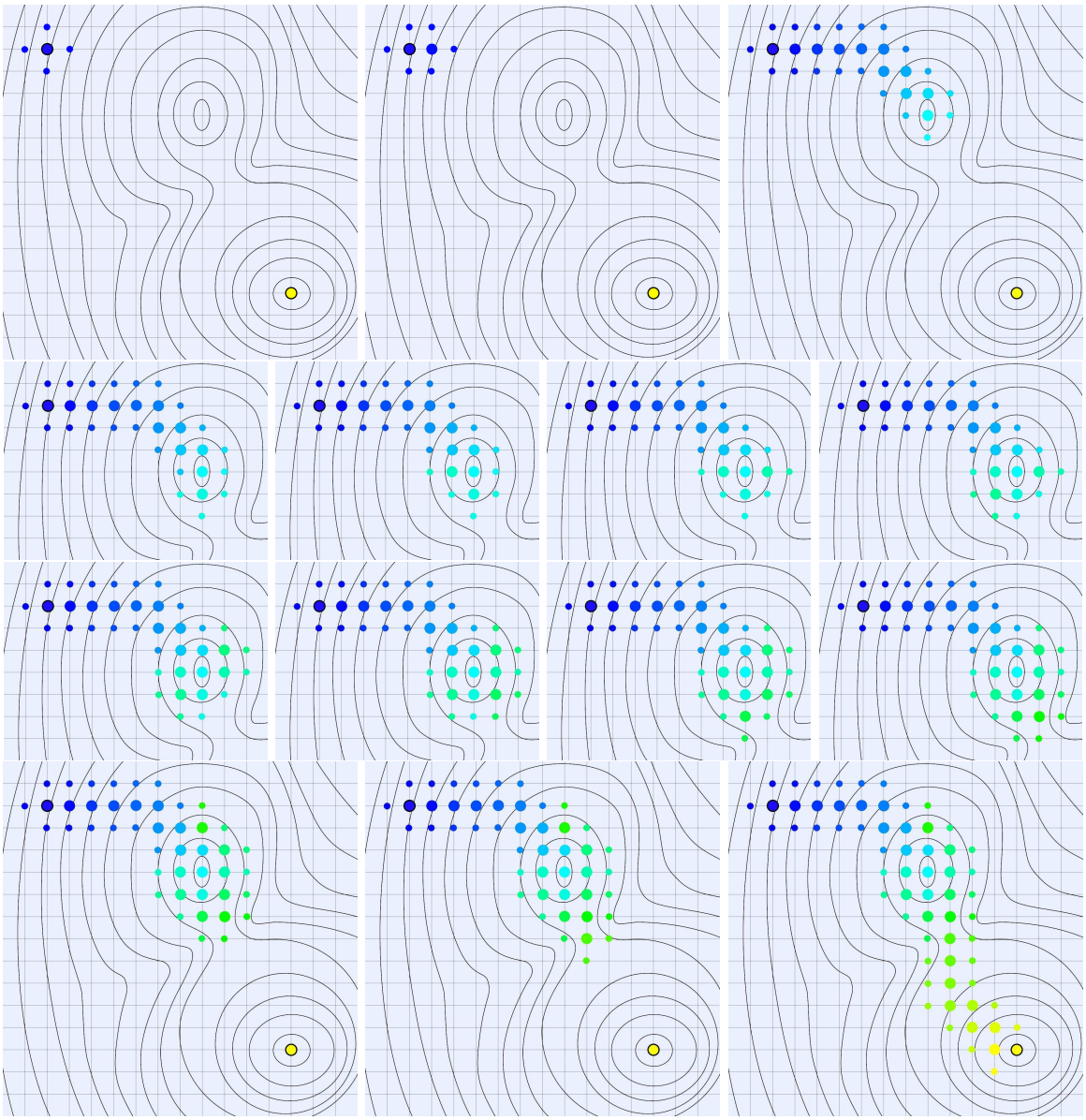


FIG. *5.6 – Détails de la planification.*

5.2.2 Mécanismes présents

Regardons rapidement les mécanismes et hypothèses présents dans l'algorithme BFP. Les identifier nous servira par la suite pour adapter cet algorithme à la planification d'appuis et

faire des parallèles.

5.2.2.1 Champ de potentiel

Le premier élément évident est l'utilisation d'un champ de potentiel. Celui-ci fournit un guide dont il faut noter qu'il n'est pas suivi parfaitement : du fait de l'utilisation d'une grille et de la restriction du déplacement uniquement vers des m-voisins, l'algorithme ne suit quasiment jamais la direction de plus grande pente, il prend seulement la direction possible qui s'en rapproche le plus.

5.2.2.2 Discrétisation

Le choix d'une grille impose une discrétisation explicite de l'espace de configuration. Le choix du pas de discrétisation, éventuellement différent pour chaque dimension, relève d'un compromis entre vitesse et précision : un pas trop petit entraîne un nombre très important de nœuds à visiter, augmentant le nombre d'itérations et le nombre de points dont il faut évaluer le potentiel. A l'opposé, un pas trop grand est susceptible de faire rater des détails du champ, et donc de l'espace de configuration : si l'espace de configuration comporte un passage étroit qu'il faut emprunter et qu'aucun point de la grille ne s'y trouve, alors l'algorithme échouera bien qu'il existe en réalité un chemin (BFP devient complet quand le pas de discrétisation tend vers 0). Si il y a aussi de petits obstacles ou des obstacles fins dans l'espace de configuration, un pas trop grand permettrait d'avoir deux points voisins de part et d'autre de ces obstacles, le chemin les reliant n'étant pas dans $\mathcal{C}_{\text{free}}$.
On remarquera qu'il n'est pas difficile d'adapter l'algorithme pour travailler avec une discrétisation à pas variable, ce qui permettrait de prendre une grille plus précise dans les zones ayant beaucoup de détails. Il suffit pour cela de modifier la définition de voisin. Les critères d'adaptation du pas sont en revanche moins évidents à trouver.

5.2.2.3 Chemin libre entre voisins

L'algorithme BFP fait une hypothèse importante que nous avons soulevée ci-dessus : le chemin entre deux voisins n'est pas en collision avec les $\mathcal{C}$-obstacles. Cela n'est vrai que si la discrétisation est suffisamment fine. Autrement il faudrait rajouter un test dans l'algorithme à la ligne 9 qui deviendrait :

$$\textbf{if } U(\mathbf{q}') < M \text{ and no } \mathit{isIn}(\mathbf{q}',T) \text{ and } \mathit{freePath}(\mathbf{q},\mathbf{q}') \textbf{ then}$$

avec $\mathit{freePath}(\mathbf{q},\mathbf{q}')$, une fonction renvoyant *vrai* s'il existe un chemin libre de collision entre $\mathbf{q}$ et $\mathbf{q}'$, *faux* sinon.

5.2.2.4 Limitation de l'espace de configuration

L'utilisation d'un seuil M impose une limitation implicite de l'espace de configuration pourvu que le champ de potentiel soit bien globalement croissant quand on s'éloigne de l'objectif, ce qui devrait toujours être le cas si on veut pouvoir partir d'une configuration arbitrairement loin de celle finale. L'utilisation du seuil définit l'ensemble des configurations potentiellement intéressantes (en considérant comme forcément inintéressantes celles dont le potentiel se trouve au-dessus) et restreint l'espace de recherche à celles-ci. Sa principale

fonction est de faire s'arrêter l'algorithme en cas d'échec plus tôt que s'il avait fallu visiter tout l'espace de configuration, en vain.

5.2.2.5 Évitement de boucle

Éviter les boucles est un mécanisme majeur quant à la bonne marche et à la garantie de terminaison de l'algorithme. Cela se fait à la ligne 9 grâce à $isIn(\mathbf{q}',T)$ qui veille à ce qu'on ne visite pas deux fois le même point de grille. Si ce mécanisme est ici trivial, dans le cas d'une discrétisation explicite, il faudra veiller à le reproduire correctement dans le cas plus complexe du planificateur que nous présentons au paragraphe suivant.

5.2.2.6 Nouvelles feuilles

Un autre mécanisme qui ici aussi est simple, mais mérite néanmoins qu'on lui donne un nom, est la génération des nouvelles feuilles (ou encore le choix des voisins). Dans l'algorithme présenté, cela se fait par la notion de m-voisins, m étant un entier choisi préalablement. On pourrait cependant choisir d'autres méthodes de sélection des voisins. Par exemple, regarder tous les points de la grille à une distance inférieure à une constante donnée, ou encore considérer des m-voisins, $m \geq 2$ dans un cône d'axe $\mathbf{qq}^{\text{final}}$ et des 1-voisins en dehors... le but d'un choix plus spécifique étant de limiter le nombre d'itérations et de feuilles visitées.

5.2.2.7 Condition de fin

Le dernier mécanisme présent dans BFP est la condition déterminant que l'algorithme a réussi la planification demandée. Cela se fait ici par le biais d'une égalité (ligne 13). Il faut noter que cette condition de fin n'est pas reliée directement au champ de potentiel. Si la configuration finale ne se trouve pas au minimum global du champ de potentiel, l'algorithme fonctionne toujours. Il est cependant probable qu'il sera moins rapide dans ce cas. On retrouve bien le fait que le champ de potentiel n'est qu'un guide.
On peut donc choisir d'autres conditions de terminaison. Par exemple requérir de la feuille considérée qu'elle se trouve dans une région de l'espace de configuration : une région simple serait une sphère centrée en $\mathbf{q}^{\text{final}}$, mais on peut aussi vouloir que la planification arrive dans un demi-espace de $\mathcal{C}$ (ce serait le cas d'un robot devant dépasser une "ligne d'arrivée" dans l'espace de travail).
On pourrait à l'opposé vouloir baser la condition de fin sur le champ lui-même, par exemple sous la forme $U(\mathbf{q}) \leq U_{\min} + \epsilon$, $U_{\min}$ étant la valeur du champ à son minimum global et ϵ un nombre positif.

5.3 Planification d'appuis

Dans cette partie, nous allons proposer une première implémentation de la planification de points de contact. Pour cela, nous nous inspirons de l'algorithme BFP présenté au paragraphe précédent. Nous en gardons le concept de construction séquentielle d'un arbre guidée par un champ de potentiel. Il est par contre nécessaire de retravailler le fonctionnement de la majorité des mécanismes présentés au paragraphe 5.2.2, mécanismes dont nous conservons cependant les grands principes.

Il n'y a plus ici de discrétisation par une grille, la notion de voisin perd donc du sens, et puisqu'il est toujours question d'arbre, on lui préfèrera dans l'algorithme la notion de *fils*, et plus généralement on adoptera le vocabulaire relatif à un arbre. On gardera cependant le terme de *voisin* pour des ensembles de contacts (cf ci-après).

5.3.1 Espace de planification

La planification d'appuis se fait dans l'espace de configuration $\mathcal{C}$ en se restreignant à l'espace de configuration libre en contact $\mathcal{C}_c$ (cf chap. 2). Comme nous l'avons expliqué, tous les contacts ne sont pas intéressants ni faciles à appréhender. On se limite donc à des contacts fixes (pas de contacts roulant ou glissant par exemple) entre des parties prédéfinies du robot et des surfaces de l'environnement.

5.3.1.1 Définitions

Du côté du robot on construit un ensemble de paires (corps r_i, point P_j^i), P_j^i étant un point régulier de la surface du corps. On rappelle (cf chap. 3) que la donnée d'un point sur la surface d'un corps définit automatiquement un vecteur normal en ce point auquel on adjoint un vecteur de direction qui lui est perpendiculaire. On appelle *emplacement de contact* le quintuplet $\left(r_i, \mathbf{P}_j^i, \mathbf{n}_j^i, \mathbf{d}_j^i, h_j^i\right)$ où h_j^i est l'ensemble des points de l'enveloppe convexe délimitant la surface de r_i sur laquelle a lieu le contact associé à P_j^i. On nomme $\mathcal{E}_{\text{contacts}}$ l'ensemble des *emplacements de contact* possibles sur le robot.
Côté environnement, on considère un certain nombre de surfaces (possiblement toutes) régulières. En tout point de coordonnées (x, y) sur une telle surface, on peut définir une normale, et la donnée d'un vecteur jamais colinéaire à une normale de la surface, permet de définir un vecteur de direction perpendiculaire pour chaque normale. On se restreindra pour des raisons de simplicité et de stabilité à des surfaces planes de l'environnement, mais les algorithmes et discussions suivantes s'étendent facilement aux surfaces d'objets convexes. Pour les objets non convexes, il y a un peu plus de travail, notamment pour gérer la collision entre le corps et l'objet en contact. $\mathcal{E}_{\text{surfaces}}$ est l'ensemble des surfaces S^E sélectionnées.

Un contact c se définit par la donnée d'une paire de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ qu'on peut désigner par un indice de 1 à $\text{card}(\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}})$ et un triplet (x, y, θ) de position et orientation relative entre les deux, (x, y) déterminant un point de la surface et θ l'angle entre les directions associées à l'emplacement de contact et à la surface. EC est l'ensemble des contacts c, il s'identifie à un sous espace de $\mathbb{N} \times \mathbb{R}^3$ et est donc de dimension 4.

On définit par $\mathcal{S}$ l'ensemble des i-uplets de EC, $1 \leq i \leq n+1$, ne faisant pas intervenir deux fois le même corps du robot (qui en compte $n+1$, puisqu'ayant n degrés de liberté internes) :

$$\mathcal{S} = \bigcup_{i=1}^{n+1} \left\{(c_1, c_2, \cdots, c_i) \in EC^i\right\} \tag{5.18}$$

C'est l'ensemble des ensembles de contact qu'on peut vouloir imposer au robot (sans certitude que cela soit possible). On notera qu'il ne contient pas l'ensemble vide : le robot a besoin d'au moins un appui pour pouvoir être en équilibre.

Puisque le contact est à la base de la planification d'appuis, l'espace naturel dans lequel planifier est $\mathcal{S}$: amener le robot d'un endroit à un autre de l'espace de travail $\mathcal{W}$ revient

à chercher une succession d'ensembles de contacts $(s_1, s_2, \cdots, s_l)$ *faisable* par le robot (cf paragraphes suivants).

On peut adapter la notion de m-voisin à $\mathcal{S}$: les m-voisins de s sont les éléments de $\mathcal{S}$ qui diffèrent de s par au plus m *changements* de contacts (c'est-à-dire m ajouts ou suppressions de contacts) :

$$s \text{ et } s' \ m\text{-voisins} \Leftrightarrow \text{card}(s \cap s') + m \geq \text{card}(s \cup s') \tag{5.19}$$

On notera que changer un corps en contact de place compte pour deux changements : enlever le contact puis créer le nouveau. Par exemple dans une marche bipède, si $s_1 = \{c_g, c_d\}$ est l'ensemble de contacts lors d'une phase de double support, $s_3 = \{c_g, c'_d\}$ celui à la phase de double support suivante et $s_2 = \{c_g\}$ l'ensemble de contacts pendant le simple support, s_1 et s_3 sont des 1-voisins de s_2, mais s_1 et s_3 sont 2-voisins : un pas se fait en deux étapes.

5.3.1.2 Ensembles de contacts faisables

Travailler dans S permet d'être dans un espace qui a plus de sens que celui de configuration vis-à-vis du problème qui nous intéresse, mais aussi une réduction substantielle de la dimension de l'espace de recherche : en pratique on ne dépasse pas 4 à 5 contacts, ce qui restreint à travailler dans un espace de dimension au plus 20 (5×4) quand l'espace de configuration va d'une dimension 30 à plus de 100.
On ne peut cependant oublier complètement l'espace de configuration puisqu'il nous faut l'état complet du robot pour tester les propriétés physiques et géométriques qui lui sont relatives. On a là la notion d'ensemble de contacts *faisable* :
un ensemble de contacts $s \in \mathcal{S}$ est dit faisable si il existe une posture du robot qui vérifie les contacts de s ainsi que les contraintes géométriques (limites articulaires, non-collision...) et physiques (stabilité, possiblement les limites de couples...). On appellera *posture témoin* une telle posture. La faisabilité d'un ensemble de contacts s se teste grâce au générateur de posture présenté au chapitre 3.
En reprenant les notations de ce chapitre, on appelle $\mathcal{Q}_s$ le problème d'existence d'une posture associée à l'ensemble de contacts s où les tâches T_i sont précisément les contacts $c_i \in s$. Il n'est a priori pas nécessaire d'utiliser un critère d'optimisation pour cette génération de posture. On verra cependant plus loin des cas qui le demandent.

5.3.1.3 Séquences faisables

Une séquence $(s_1, s_2, \cdots, s_l)$, quant à elle, est dite faisable si on peut trouver un chemin (quasi-statique dans ce travail) qui vérifie successivement et sans discontinuité les ensembles de contacts de la séquence :

$$\begin{aligned} \exists \varphi : [1, l] \to \mathcal{C}, & \\ & (1) \quad \varphi \text{ est } C^0, \\ & (2) \quad \forall i \in [|1, l-1|], \ \forall u \in [i, i+1[\ , \varphi(u) \in \mathcal{Q}_{s_i} \\ & (3) \quad \varphi(l) \in \mathcal{Q}_{s_l} \end{aligned}$$

Trouver une telle fonction peut se faire en adaptant des méthodes de planifications classiques ou en résolvant un problème de génération de trajectoire. De telles approches ont cependant un coût prohibitif pour une utilisation comme routine de vérification au sein d'un

algorithme de planification d'appuis. On utilisera plutôt des heuristiques que nous détaillons maintenant.

En premier lieu, les propriétés attachées à la fonction φ impliquent que pour deux ensembles consécutifs s_i et s_{i+1}, l'intersection de $\mathcal{Q}_{s_i}$ et $\mathcal{Q}_{s_{i+1}}$ est non vide. Si elle l'était, alors φ serait nécessairement discontinue en $i+1$.
$\mathcal{Q}_s$ peut s'écrire comme l'intersection de deux ensembles : $\mathcal{Q}_s^{\text{géom}}$, défini par les contraintes géométriques (limites articulaires, collisions, contacts) de $\mathcal{Q}_s$, et $\mathcal{Q}_s^{\text{stab}}$, défini par les contraintes de stabilité.
$\mathcal{Q}_{s_i} \cap \mathcal{Q}_{s_{i+1}} \neq \oslash$ implique entre autre que $\mathcal{Q}_{s_i}^{\text{géom}} \cap \mathcal{Q}_{s_{i+1}}^{\text{géom}} \neq \oslash$. Une façon suffisante d'obtenir cela est de **se restreindre à l'utilisation de 1-voisins**. Pour $m \geq 2$, l'intersection des $\mathcal{Q}_s^{\text{géom}}$ de deux m-voisins peut en effet être obligatoirement vide. Ainsi en est-il pour s_1 et s_3 dans l'exemple de la marche bipède dans la sous-section précédente : $\mathcal{Q}_{s_1}^{\text{géom}}$ et $\mathcal{Q}_{s_3}^{\text{géom}}$ sont des sous-variétés parallèles de $\mathcal{C}$. En revanche si s_i et s_{i+1} sont des 1-voisins, et en supposant (sans perte de généralité) que $s_i \subset s_{i+1}$, on a $\mathcal{Q}_{s_i}^{\text{géom}} \supset \mathcal{Q}_{s_{i+1}}^{\text{géom}}$ et donc l'intersection de ces deux ensembles n'est pas vide.
Remarquons pour la suite qu'on a également $\mathcal{Q}_{s_i}^{\text{stab}} \subseteq \mathcal{Q}_{s_{i+1}}^{\text{stab}}$.

Dans la construction de notre suite faisable $(s_1, s_2, \cdots, s_l)$ on ne fera donc qu'ajouter ou enlever un contact entre deux ensembles consécutifs. Intuitivement, lorsqu'on rajoute un contact à s_i, on ne pourra trouver une trajectoire continue depuis $\mathcal{Q}_{s_i}$ que si on peut prendre ce nouveau contact géométriquement, sans s'en servir pour la stabilité. On peut reformuler cela en disant qu'il faut être en équilibre juste avant la prise du contact. De même lorsqu'on retire un contact, il faut pouvoir être en équilibre un temps infinitésimal après l'avoir enlevé. Cette notion de "juste avant", ou "juste après" se retrouve dans la propriété (2) de φ par le fait que les intervalles $[i, i+1[$ sont ouverts à droite.
Plus formellement, pour s_i et s_{i+1} 1-voisins et $s_i \subset s_{i+1}$, on a

$$\mathcal{Q}_{s_i} \cap \mathcal{Q}_{s_{i+1}} \neq \oslash \quad \Leftrightarrow \quad \mathcal{Q}_{s_i}^{\text{géom}} \cap \mathcal{Q}_{s_i}^{\text{stab}} \cap \mathcal{Q}_{s_{i+1}}^{\text{géom}} \cap \mathcal{Q}_{s_{i+1}}^{\text{stab}} \neq \oslash \tag{5.20}$$

$$\Leftrightarrow \quad \left(\mathcal{Q}_{s_i}^{\text{géom}} \cap \mathcal{Q}_{s_{i+1}}^{\text{géom}}\right) \cap \left(\mathcal{Q}_{s_i}^{\text{stab}} \cap \mathcal{Q}_{s_{i+1}}^{\text{stab}}\right) \neq \oslash \tag{5.21}$$

$$\Leftrightarrow \quad \mathcal{Q}_{s_{i+1}}^{\text{géom}} \cap \mathcal{Q}_{s_i}^{\text{stab}} \neq \oslash \tag{5.22}$$

Ce qui signifie qu'un chemin existe entre deux 1-voisins si on peut trouver une posture qui satisfasse aux contraintes géométriques du plus grand (au sens de l'inclusion) de ces deux voisins et aux contraintes de stabilité du plus petit.

On a donc maintenant une heuristique pour construire une séquence $(s_1, s_2, \cdots, s_l)$ en s'assurant de pouvoir trouver un chemin stable et sans collision dans $\mathcal{C}$ entre deux ensembles de contacts *consécutifs* : **utiliser des 1-voisins et vérifier l'existence d'une posture qui prenne en compte géométriquement, mais pas dans la stabilité, le contact différenciant ces deux voisins**.

Il reste que cela n'assure pas l'existence d'un chemin complet : pour $2 \leq i \leq l-1$, si Q_{s_i} n'est pas un sous ensemble connexe de $\mathcal{C}$, il peut ne pas y avoir de chemin entre $Q_{s_{i-1}} \cap Q_{s_i}$ et $Q_{s_i} \cap Q_{s_{i+1}}$. Cela peut par exemple arriver à cause d'obstacle long et fin (cf Fig 5.7), mais reste cependant un cas rare en pratique. **On supposera qu'il existe toujours un chemin entre $Q_{s_{i-1}} \cap Q_{s_i}$ et $Q_{s_i} \cap Q_{s_{i+1}}$.**

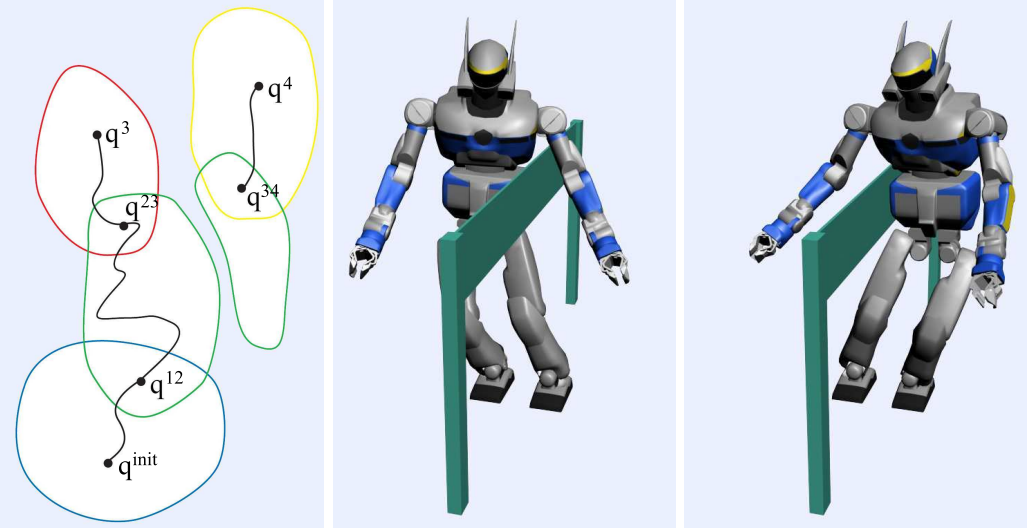


FIG. 5.7 – *Problème lié à la non-connexité d'un ensemble $\mathcal{Q}_{s_i}$ et exemple où $\mathcal{Q}_{s_i}$ est non-connexe. Du fait de la non-connexité de $\mathcal{Q}_{s_2}$ (figure de gauche, en vert), il n'est pas possible de trouver un chemin de $\mathcal{Q}_{s_1}$ (bleu) à $\mathcal{Q}_{s_4}$ (jaune) bien qu'on soit capable de trouver un chemin de $\mathcal{Q}_{s_1}$ à $\mathcal{Q}_{s_2}$ et de $\mathcal{Q}_{s_2}$ à $\mathcal{Q}_{s_4}$. En revanche on peut trouver un chemin de $\mathcal{Q}_{s_1}$ (bleu) à $\mathcal{Q}_{s_3}$ (rouge) car $Q_{s_1} \cap Q_{s_2}$ et $Q_{s_2} \cap Q_{s_3}$ sont sur une même composante connexe de $\mathcal{Q}_{s_2}$. Les deux figures à droites illustrent un cas où l'espace $\mathcal{Q}_{s_i}$ n'est pas connexe à cause d'un obstacle fin des deux côtés duquel on peut trouver des postures valides, sans qu'il existe de chemin entre elles (le robot ne pouvant passer sous l'obstacle sans perdre l'équilibre).*

5.3.2 Algorithmes

5.3.2.1 Contacts BFP

L'algorithme 2 présente la routine principale de notre planification d'appuis, appelée *Contacts Best First Planning*. Celle-ci est, comme nous l'avons annoncé, proche dans son principe de l'algorithme 1. Sa mise en œuvre est cependant plus complexe.

En premier lieu, on travaille avec des ensembles de contacts faisables. On associera donc systématiquement une posture témoin à un ensemble. Les entrées et sorties de l'algorithme sont donc basées sur des couples $(s, \mathbf{q}) \in \mathcal{S} \times \mathcal{C}$.
Le champ de potentiel U est lui aussi construit en toute généralité sur $\mathcal{S} \times \mathcal{C}$, même si on peut vouloir se restreindre à $\mathcal{S}$ ou $\mathcal{C}$. Si le potentiel fait intervenir la configuration, il sera alors préférable d'utiliser un critère d'optimisation pour la génération de posture, afin de mieux maîtriser le choix de $\mathbf{q} \in \mathcal{Q}_s$ pour lequel on calculera la valeur du champ de potentiel. Ainsi, parce que deux ensembles de contacts proches s_1 et s_2 auront leurs postures témoins $\mathbf{q}^1$ et $\mathbf{q}^2$ proches dans la majorité des cas, $U(\mathbf{q}^1, s_1)$ et $U(\mathbf{q}^2, s_2)$ seront proches. Sans critère, la posture témoin trouvée est beaucoup plus dépendante du schéma de résolution des contraintes et de l'initialisation du générateur de posture, ce qui est susceptible d'engendrer des différences de potentiel importantes pour deux ensembles de contacts proches.

Puisqu'on travaille dans l'ensemble des ensembles de contacts, il n'est plus question de donner la condition de fin par une posture à atteindre, et comme on n'utilise pas (explicitement) de discrétisation de $\mathcal{S}$, donner un point s de cet espace comme but n'est pas non plus évident car l'algorithme aura du mal à y arriver précisément. Nous pouvons en revanche profiter de l'utilisation du générateur de posture et spécifier la condition de fin par une tâche

Algorithm: Contacts BFP
Data: $(\mathbf{s}_{\text{init}}, \mathbf{q}^{\text{init}})$, $\mathcal{T}_{\text{final}}$,$U(\mathbf{q}, \mathbf{s})$
Result: une suite $\left[(\mathbf{s}_0, \mathbf{q}^0), (\mathbf{s}_1, \mathbf{q}^1), \cdots, \left(\mathbf{s}_l, \mathbf{q}^l\right)\right]$, avec $(\mathbf{s}_0, \mathbf{q}^0) = (\mathbf{s}_{\text{init}}, \mathbf{q}^{\text{init}})$, $\mathbf{q}^l$ vérifiant $\mathcal{T}_{\text{final}}$ et $\mathbf{s}_{i+1}$ voisin de $\mathbf{s}_i$ pour $0 \leq i < l$

```
- L est une liste triée, initialement vide
- l est une liste
- n et n' sont des nœuds
begin
    n ← newNode(s_init, q^init)
    insert(n,L)
    while no isEmpty(L) do
        n ← first(L)
        if allowsToReachGoal(n) then
            return backtrackPath(n)
        end
        l ← generateSons(n)
        for each node n' in l do
            if U(n') < M and trajectoryExists(n') then
                insert(n',L)
            end
        end
    end
    return failure
end
```

Algorithm 2: Algorithme Contacts BFP

$\mathcal{T}_{\text{final}}$: l'algorithme termine avec succès pour l'ensemble s_l si $\mathcal{Q}_{s_l} \cap \mathcal{T}_{\text{final}}$ est un ensemble non vide. La tâche doit bien sûr être faisable elle-même.

L'algorithme prend donc en entrée un couple (ensemble de contacts faisables, posture témoin) comme point de départ, un ensemble de contraintes (égalités et/ou inégalités) $\mathcal{T}_{\text{final}}$ spécifiant le but à atteindre, et un champ de potentiel U sur $\mathcal{S}\times\mathcal{C}$ pour guider la planification. La sortie est une séquence de couples $(s, \mathbf{q})$ commençant par le couple d'entrée et dont la posture témoin du dernier vérifie la tâche $\mathcal{T}_{\text{final}}$.

Il s'agit toujours de faire grandir un arbre. Un nœud contient un couple $(s, \mathbf{q})$, la valeur du champ de potentiel en ce couple et un pointeur vers son père. Par abus de langage, pour un nœud n contenant $(s, \mathbf{q})$, on définira $U(n) = U(s, \mathbf{q})$ et on parlera de potentiel du nœud. La fonction *newNode*$(s, \mathbf{q})$ construit un nœud basé sur le couple $(s, \mathbf{q})$.

On utilise toujours une liste L ordonnée par ordre de potentiels décroissants. Elle supporte les mêmes fonctions *insert*, *isEmpty* et *first*. La seule différence est dans la nature des nœuds manipulés.

Pour un nœud n contenant $(s, \mathbf{q})$, la fonction *allowsToReachGoal* ajoute la tâche $\mathcal{T}_{\text{final}}$ à $\mathcal{Q}_s$ et essaye de trouver une posture témoin $\mathbf{q}^{\text{final}}$ pour ce nouveau problème, grâce au générateur de posture. Elle renvoie *vrai* en cas de succès, *faux* sinon. Elle peut en plus tester l'existence d'une trajectoire entre $\mathbf{q}$ et $\mathbf{q}^{\text{final}}$ avant de renvoyer *vrai*. Nous supposons cependant ici que cette trajectoire existe toujours.
La fonction *allowsToReachGoal* remplace donc le test de la condition de fin dans l'algorithme BFP. On remarquera qu'elle se situe avant la génération des fils, contrairement à ce qui se passe dans BFP. De ce fait, on évite de faire un test pour chaque fils, une génération de posture pouvant parfois être lourde.

La fonction *generateSons*(n) construits les fils du nœud n. Elle est détaillée à la section suivante. C'est là que se trouvent les mécanismes d'évitements de boucle, de discrétisation (dans une certaine mesure) et bien sûr de génération de feuilles.

La fonction *trajectoryExists*(n) fait écho à l'existence d'un chemin libre entre deux voisins dans l'algorithme BFP. Elle renvoie *vrai* si il existe une trajectoire entre la configuration témoin de n et celle de son père, *faux* sinon. Néanmoins en se basant sur les heuristiques présentées précédemment, nous ne l'utilisons pas : nous construisons les fils de façon à ce que cette trajectoire existe dans l'immense majorité des cas, si bien que dans notre implémentation cette fonction renvoie toujours *vrai*.

5.3.2.2 Génération des fils

La génération de fils pour un nœud n contenant un couple $(s, \mathbf{q})$, se base sur les 1-voisins de s, pour les raisons expliquées à la section 5.3.1.3. La construction de fils se fait donc naturellement en deux grandes étapes : l'ajout et la suppression d'un contact à s. s étant un ensemble fini, la suppression d'un contact ne pose pas de problème spécifique : il suffit d'essayer chaque contact l'un après l'autre. L'ajout de contact en revanche est plus délicat car, sauf à être un des éléments de plus grand cardinal de $\mathcal{S}$, s a une infinité de 1-voisins. Avec la génération d'un bon champ de potentiel, la génération de nouveau contact est la partie la plus complexe de notre planification.

En section 5.3.1.3, nous avons vu que lorsqu'un contact c est enlevé à un ensemble s, on cherche à générer une posture pour un problème $\mathcal{Q}_s$ modifié où c n'intervient pas dans la stabilité. De même si on rajoute un contact le problème est $\mathcal{Q}_{s\cup c}$ sans que c ne participe à

l'équilibre. Il faut donc rajouter à la structure de contact un booléen indiquant si celui-ci est à prendre en compte ou non dans la stabilité.
De plus, puisque lorsqu'on enlève un contact à s, celui-ci est quand même gardé au niveau géométrique, il n'est véritablement retiré qu'au niveau des petits-fils de s. Afin de savoir si un contact doit être retiré, on utilise un second booléen.

L'algorithme 3 décrit la façon de générer les fils d'un nœud n du Contacts BFP. Il renvoie une liste de ces fils. Il utilise la classe *PostureGenerator* décrite au chapitre 3, dont une instance est associée à chaque nœud. Cette classe est la version informatique du problème $\mathcal{Q}_s$.

L'algorithme se découpe en trois parties. La première (lignes 5 à 13) construit les bases du problème $\mathcal{Q}$ qui servira pour tous les fils. Pour ce faire elle élimine un contact si il est noté comme étant relâché (ligne 9), et force tous les autres à participer à la stabilité (ligne 10).
La seconde (lignes 14 à 25) s'attaque à la suppression de contact. Elle marque un à un les contacts comme étant relâchés, pourvu qu'il y ait strictement plus d'un contact ($\mathcal{S}$ ne contient pas l'ensemble vide) et cherche dans chaque cas s'il existe une posture témoin.
La troisième (lignes 26 à 33) s'occupe de l'ajout de contact. Pour cela, elle fait appel, pour chaque couple de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$, à la sous-fonction *generateNewContacts* qui est décrite par l'algorithme 4 et que nous détaillons après.

Un élément de type *PostureGenerator* supporte plusieurs opérations :

- *initCalculator*(pc) prépare un nouveau *PostureCalculator*. Elle crée entre autre les bases du problème d'optimisation en ajoutant les contraintes toujours présentes telles que les limites articulaires ou les collisions. Elle initialise aussi ce problème avec la posture témoin de n. Cette posture vérifie en effet déjà la majorité des contraintes des problèmes de génération à résoudre pour les fils, et n'est généralement pas loin de la solution. Cela permet d'accélérer la génération de posture,
- *getContacts*(pc) renvoie le vecteur des contacts présents dans pc,
- *addContact*(c, pc) rajoute le contact c au problème pc, uniquement pour les contraintes géométriques. La construction des contraintes de stabilité et la mise à jour des contraintes d'évitement de collision se fait plus tard,
- *getNumberOfContacts*(pc) renvoie le nombre total de contacts présents dans pc, qu'ils participent à la stabilité ou non, qu'ils soient marqués comme relâché ou non,
- *copy*(pc) renvoie une copie de pc,
- *releaseIthContact*(pc,i) met à *vrai* pour le i$^{\text{ème}}$ contact de pc, le booléen indiquant que ce contact est en train d'être enlevé, et à *faux* le booléen de participation à la stabilité,
- *newNode*(pc) crée un nouveau nœud basé sur pc et donc sur l'ensemble de contact associé. La posture témoin n'est pas encore forcément trouvée à ce moment,
- *isAlreadyInContact*(i, pc) renvoie *vrai* si le corps correspondant au i$^{\text{ème}}$ élément de $\mathcal{E}_{\text{contacts}}$ est déjà engagé dans un contact. Ce test fait écho à l'une des propriétés de construction de $\mathcal{S}$ disant qu'un même corps du robot n'apparaît pas deux fois dans un ensemble de contacts,
- enfin *generate*(pc) cherche une posture témoin pour l'ensemble de contacts s correspondant à pc. Pour cela une phase de mise à jour de pc est nécessaire : d'une part il faut générer les contraintes de stabilité en fonction des contacts qui y participent, de l'autre il faut supprimer certaines contraintes de collision correspondant aux paires (corps du robot, objet de l'environnement) qui découlent des contacts, contact et non-collision étant contradictoires. Une fois cette mise à jour effectuée, le générateur de posture est

```
   Algorithm:generateSons
   Data: n, un nœud
   Result: l, un vecteur de feuilles descendant de ce nœud
 1 - pc et pci sont des structures de type PostureCalculator
 2 - contacts est un vecteur de contacts
 3 - N est un ensemble sans duplicata contenant tous les nœuds ou feuilles déjà générés
 4 begin
 5     l ← ∅
       // Calculateur de base
 6     initCalculator(pc)
 7     contacts ← getContacts(getCalculator(n))
 8     for each contact c in contacts do
 9         if no isBeingReleased(c) then
10             participateToStability(c)
11             addContact(c,pc)
12         end
13     end
       // Suppression contact
14     if getNumberOfContacts(pc) ≥ 2 then
15         for i ← 1 to getNumberOfContacts(pc) do
16             pci ← copy(pc)
17             releaseIthContact(pci,i)
18             n' ← newNode(pci)
19             if add(n',N) then
20                 if generate(pci) then
21                     add(n', l)
22                 end
23             end
24         end
25     end
       // Ajout contact
26     for i ← 1 to card(E_contacts) do
27         if no isAlreadyInContact(i,pc) then
28             for j ← 1 to card(E_surfaces) do
29                 l' ← generateNewContacts(pc,i,j)
30                 l ← l ∩ l'
31             end
32         end
33     end
34     return(l)
35 end
```

Algorithm 3: Génération des fils

appelé. Si une posture est trouvée, alors *generate* retourne *vrai*, *faux* sinon.

La structure de contact, de son côté, est une implémentation de la classe *ConstraintSet* et supporte deux fonctions donnant l'état des deux booléens associés au contact : *isBeingReleased* et *participateToStability* renvoie *vrai* ou *faux* respectivement selon que le contact est marqué comme relâché ou non et qu'il participe à la stabilité ou non.

Dans les algorithmes 3 et 4, on a le même mécanisme en deux étapes pour valider un nœud. On vérifie d'abord que l'ensemble de contacts du nœud n'est pas trop proche d'un autre déjà existant. Si tel n'est pas le cas alors on tente une génération de posture. En cas de réussite le nœud est valide (son ensemble de contacts est faisable) et ajouté à la liste que retourne la fonction.

Le premier des deux tests est la transposition du mécanisme d'évitement de boucle à notre planificateur. Sa mise en oeuvre ne passe pas directement par un test d'égalité, mais par la définition d'une relation d'ordre stricte totale $\prec$ sur $\mathcal{S}$ (et par extension, sur l'ensemble des nœuds basés sur les éléments de $\mathcal{S}$) décrite à la section 5.3.3. On peut alors utiliser un arbre rouge-noir pour stocker les nœuds générés et effectuer rapidement (en $\mathcal{O}(\ln n)$) le test d'existence d'un nouveau nœud.
L'ensemble N, accessible par les deux fonctions, implémente un tel arbre. La fonction *add*(n,N) tente d'insérer le nœud n dans N. Si elle ne trouve pas de nœud n' égal à n au sens de $\prec$ (c'est-à-dire que l'on a soit $n \prec n'$ soit $n' \prec n$), elle insère effectivement le nœud dans N et renvoie *vrai*. Elle retourne *faux* dans le cas contraire et le nœud n'est pas inséré.

Deux fonctions de l'algorithme 4 demandent à être expliquées.
La fonction *forbiddenContact*(i,j) cherche dans une liste de paire (corps du robot, surface de l'environnement) F spécifiée par l'utilisateur si elle trouve le couple (r,s) où r est le corps mis en jeu par le i$^{\text{ème}}$ élément de $\mathcal{E}_{\text{contacts}}$ et s la j$^{\text{ème}}$ surface de $\mathcal{E}_{\text{surfaces}}$. Cette liste L permet à l'utilisateur d'interdire certains types de contact. Cela peut-être motivé par des questions de sécurité, d'aspect visuel du résultat ou d'insertion de connaissance dans la planification pour accélérer le processus. Ainsi on pourrait par exemple vouloir empêcher un robot de mettre ses pieds sur une table, ou dans un scénario faisant majoritairement intervenir de la marche interdire le contact des mains avec le sol.
La fonction *getContactCandidates*(i,j) retourne une liste de contacts candidats. L'obtention de cette liste est discutée à la section 5.3.4. Pour ces contacts le booléen de participation à la stabilité est mis à *faux*.

5.3.3 Relation d'ordre sur les ensembles de contacts

Nous avons vu qu'un contact pouvait s'identifier par un quadruplet (i, x, y, θ) où i est un indice identifiant une paire de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$. Nous allons baser la comparaison entre deux ensembles de contacts sur ces quadruplets et particulièrement sur les indices. Pour $s \in \mathcal{S}$, on définit $ind(s)$ comme la liste des indices des contacts de s, ordonnés par ordre croissant.
Si pour deux éléments s et s' de $\mathcal{S}$ on a $ind(s)$=$ind(s')$=$[i_1, .., i_k]$, alors on peut définir les conditions d'égalité entre ces deux éléments : on construit le vecteur des paramètres de contact $p_s = (x_1, y_1, \theta_1, \cdots, x_k, y_k, \theta_k)$ à partir des triplets (x, y, θ) rangés d'après l'ordre qu'ont les indices dans $ind(s)$. On construit de la même façon $p_{s'}$. Alors, s et s' sont égaux si et seulement si $\|A\,(p_s - p_{s'})\| \leq \epsilon$, avec A, matrice diagonale et ϵ une constante positive. Le rôle de la matrice A est d'introduire une échelle entre les angles et les distances. Il faut noter que l'égalité ainsi définie n'est pas une relation transitive.

```
   Algorithm:generateNewContacts
   Data: pc, une structure de type PostureCalculator
   Data: i et j, indices respectivement de l'emplacement de contact et de la surface
   Result: l, un vecteur de feuilles descendant de ce nœud
 1 - pci est une structure de type PostureCalculator
 2 - contacts est un vecteur de contacts
 3 - N est un ensemble sans duplicata contenant tous les nœuds ou feuilles déjà générés
 4 begin
 5     l ← ∅
 6     if forbiddenContact(i,j) then
 7         return(l)
 8     end
 9     contacts ← getContactCandidates(i,j)
10     for each contact c in contacts do
11         pci ← copy(pc)
12         addContact(c,pci)
13         n' ← newNode(pci)
14         if add(n',N) then
15             if generate(pci) then
16                 add(n', l)
17             end
18         end
19     end
20     return(l)
21 end
```

Algorithm 4: Génération de nouveaux contacts pour un couple (emplacement de contact, surface)

On peut maintenant définir la relation $\prec$ sur $\mathcal{S}$:
$s \prec s'$ si et seulement si
- $\text{card}(s) < \text{card}(s')$
- ou $\text{card}(s) = \text{card}(s')$ et $ind(s) < ind(s')$
- ou $ind(s)=ind(s')$ et $p_s \neq p_{s'}$ et $p_s < p_{s'}$

L'inégalité sur deux vecteurs s'entend comme un ordre lexicographique : on compare d'abord le premier élément de chaque vecteur ; si il y a égalité, on passe au suivant, et ainsi de suite.

Par le biais de la condition d'égalité, on obtient, avec le mécanisme de validation d'un nœud expliqué avant, une discrétisation implicite de $\mathcal{S}$ dont l'échelle est définie par A et ϵ. En effet quand un nœud est inséré dans n, il définit une région de $\mathcal{S}$ où l'on ne peut plus choisir d'ensemble de contacts. Si les surfaces sur lesquelles le robot prend contact sont bornées, on a donc un nombre fini de contacts possibles. Ce nombre reste cependant potentiellement très grand. La place de ces contacts dans $\mathcal{S}$ est déterminée par la façon de générer les contacts candidats et leur ordre de génération.

5.3.4 Génération de contacts candidats

Pour un couple de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ donné, la génération de contacts candidats revient au choix de triplets (x, y, θ).

Idéalement, le choix des candidats doit vérifier ces trois points :
- (i) assurer une bonne couverture des possibilités, pour ne pas ignorer des possibilités du robot,
- (ii) être en aussi petit nombre que possible, afin d'éviter une explosion combinatoire du nombre de nœud,
- (iii) correspondre au maximum à des contacts faisables. En effet, le temps passé à vérifier la faisabilité d'un ensemble de contacts est la partie la plus coûteuse de la planification, il est donc préférable de ne passer du temps que pour des ensembles susceptibles de servir. De plus, le générateur de posture est généralement plus lent lors d'un échec que d'un succès.

Déterminer une stratégie efficace pour répondre à ces trois points est une étape cruciale et peu évidente de notre algorithme de planification. Une telle stratégie doit être suffisamment simple pour que son coût de calcul ne dépasse pas les gains qu'elle permet d'effectuer, en particulier vis-à-vis du point (iii). Parmi les possibilités que nous avons étudiées, une notion tentante est celle de surface atteignable : étant donné un emplacement de contact du robot et une surface de l'environnement, quelle partie de la seconde le premier peut-il atteindre. Trouver une telle surface ou une bonne approximation de celle-ci permettrait en effet de résoudre le point (iii). Il s'est cependant avéré que la détermination d'une telle surface était à la fois complexe et beaucoup trop lourde en temps de calcul. Et encore n'avons nous pas pris en compte l'orientation θ.

L'idée de réduire la surface considérée est cependant nécessaire. Cela peut-être fait grossièrement mais à faible coût en considérant l'intersection de la surface avec une sphère centrée à la racine de la chaîne cinématique impliquée (bras où jambe par exemple) et de rayon judicieusement choisi.

Une première stratégie possible de choix de candidats consiste à prédéfinir des triplets pour chaque surface de $\mathcal{E}_{\text{surfaces}}$. Un bon choix permet de vérifier au moins le point (i), voir le (ii). Une telle stratégie permet d'introduire de la connaissance dans le système, ce qui peut se

révéler utile dans des passages difficiles. Elle demande cependant des choix spécifiques pour chaque scénario et une intervention de l'utilisateur. En cela, elle s'oppose à une utilisation complètement automatique de la planification.

La première idée pour avoir une stratégie rendant la planification entièrement autonome est de générer des triplets aléatoires. La seule maîtrise que l'utilisateur a alors, vis-à-vis des points (i) et (ii) se fait via le choix de la densité du tirage. Le contrôle de (iii) ne se fait encore qu'en réduisant "au mieux" la surface sur laquelle on tire. Concernant la densité, il s'est avéré expérimentalement que le tirage d'un nombre de triplets proportionnel à la racine carrée de l'aire de la surface considérée donnait de meilleurs résultats qu'avec un nombre directement proportionnel à l'aire. Cela permet en effet d'avoir suffisamment de triplets pour une petite surface, sans en avoir trop pour une grande.

Nous verrons au paragraphe 5.4 une mise en oeuvre de ces deux stratégies dans deux scénarios différents. Une partie du chapitre suivant sera dédié à l'exposition d'une stratégie plus avancée et plus efficace.

5.3.5 Champ de potentiel

La construction d'un champ de potentiel guidant efficacement la planification est le second point crucial de notre travail. Une solution élégante sera proposée au chapitre suivant. Dans ce paragraphe nous proposons un potentiel simple et discutons des possibilités offertes par l'extension du domaine de définition du champ à $\mathcal{S}$.

L'idée est d'avoir avant tout un guide dans l'espace de travail : on ne cherche pas (généralement) à aller vers une posture précise, mais plutôt vers un endroit de l'espace. La posture que l'on cherche à obtenir est précisée (de manière incomplète) par la tâche de condition finale $\mathcal{T}_{\text{final}}$. De ce fait si l'on veut baser le champ de potentiel sur la posture, on se restreindra à considérer la position et éventuellement l'orientation de la base du robot, d'un corps spécifique ou la position du centre de gravité. Un potentiel simple, efficace dans des scénarios peu complexes, est le potentiel mixte présenté au paragraphe 5.1.2 sur l'espace à trois dimensions des positions de la racine du robot. Ce potentiel est évalué pour les configurations témoins de chaque nœud.
Alternativement si on base le champ sur les contacts, c'est principalement la position de ceux-ci qui sera intéressante pour guider. Par exemple, on peut considérer le barycentre B des points de contacts P_j^i dont les coordonnées sont exprimées dans le repère absolu et construire le champ de potentiel sur la distance à un barycentre objectif.

L'avantage de ne définir le champ de potentiel que sur $\mathcal{S}$ est de le rendre indépendant de la posture témoin trouvée, cependant la définition d'une distance entre deux ensembles de contacts qui ne font pas intervenir les mêmes paires de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ est délicate si l'on souhaite avoir ensuite un bon comportement de notre planificateur avec un champ de potentiel basé sur cette distance. Ainsi avec l'utilisation du barycentre, la prise d'un nouveau contact, pourtant utile pour avancer, peut faire reculer B par rapport à l'objectif (par exemple si le robot doit prendre une prise avec sa main derrière lui pour assurer sa stabilité et avancer un pied vers l'objectif). De tels phénomènes arrivent aussi pour des potentiels construits sur la distance de la racine à un objectif (par exemple, quand le robot doit faire des pas chassés vers sa gauche, il a besoin par moment de transférer son poids sur la jambe droite pour pouvoir bouger la gauche, et ce faisant s'éloigne globalement de l'objectif). Cependant ils sont moins marqués.

Les types de champs qui viennent d'être évoqués ont l'avantage d'être simples, mais ils ont aussi un domaine d'application réduit. Leur principal problème est de ne pas prendre du tout en compte les obstacles. Comme nous l'avons déjà indiqué, les obstacles ne doivent pas être complètement évités car il faut parfois pouvoir s'appuyer dessus, cependant il serait utile que le champ de potentiel embarque des informations sur la façon de passer à proximité d'eux. Une solution à cela est proposée au chapitre suivant.

La grande utilité de l'usage de s dans la définition d'un champ de potentiel tient surtout à l'utilisation de sa partie discrète, c'est-à-dire le nombre et la nature des contacts. Cela permet d'introduire toute une série de bonus ou malus visant à favoriser ou au contraire pénaliser certaines classes d'ensemble de contacts. Cette utilisation est motivée, encore une fois, par des raisons pouvant aller de la sécurité à l'esthétique en passant par l'introduction de connaissances ou d'une expertise.
Voici quelques exemples de bonus ou malus que nous utilisons selon les scénarios proposés dans ce travail en les ajoutant à un champ défini sur la posture :

- préférence pour un nombre k de contacts, écrite sous la forme $a\,|k - k_0|$ avec k_0 le nombre de contacts désirés. Si $k_0 < 2$, cela revient à minimiser le nombre de contacts. On peut aussi faire une différence entre les contacts pris en compte dans la stabilité, les nouveaux, et ceux marqués comme retirés,
- préférence pour certains corps en contact, ou éviter certains autres corps. Cela se fait par l'ajout d'une constante au potentiel, négative ou positive selon le but recherché, lorsque le corps est présent dans les contacts de l'ensemble s considéré. Cela permet de privilégier la façon de résoudre le problème sans pour autant forcer le robot à se restreindre complètement à certains corps. On pourrait par exemple privilégier le fait que le robot se déplace debout,
- malus pour des ensembles ne faisant intervenir que certains corps. Typiquement, il peut ne pas être souhaitable que le robot ne se maintienne en équilibre qu'avec ses mains, surtout si l'on veut faire jouer le résultat sur un robot réel ensuite. Dans ce cas, on donne un fort malus aux ensembles de contacts ne faisant intervenir que les mains. Ce malus empêche alors un feuille correspondant à un tel ensemble d'être sélectionnée, mais ne désavantage pas tous les ensembles faisant intervenir les mains.

5.4 Premiers résultats

Cette section présente les résultats de notre planificateur pour deux scénarios avec le robot HRP-2 [Escande 06a] [Escande 06b]. Dans les deux cas le potentiel est construit sur la distance du centre de gravité à un point de l'espace de travail, et il y a un léger malus sur le nombre de contacts.
Nous utilisons aussi un critère d'optimisation pour la génération de posture :

$$f(\mathbf{q}) = \sum_{i=1}^{n} \left(q_i - \frac{q_i^- + q_i^+}{4} \right)^2 \tag{5.23}$$

qui est simplement la distance à une configuration de référence. Celle-ci est la moyenne entre la posture avec tous les angles à 0 et celle se situant au milieu (c'est-à-dire le plus loin) des butées angulaires.

Pour ces deux scénarios, il n'y a pas de véritable contrainte d'évitement de collision. On en utilise juste une ad hoc pour les deux pieds. Dans le deuxième, les surfaces de contact S^E de l'environnement sont définies en conséquence pour qu'il n'y ait pas de collision des pieds avec les marches.

5.4.1 Attraper une canette sur une table

Le premier scénario fait commencer le robot à un bout d'une table sur laquelle se trouve une canette qu'il doit attraper. Celle-ci se situe trop loin pour qu'il puisse directement la prendre sans perdre l'équilibre. Il lui faudra donc soit s'appuyer sur la table, soit en faire le tour. C'est la première solution que le planificateur trouvera. La table étant un peu trop haute (70cm pour un robot de 1,54m) pour permettre des contacts entre les jambes et son bord, le robot est placé sur une estrade qui le surélève de 10cm.

Dans ce scénario, $\mathcal{E}_{\text{contacts}}$ comprend six emplacements de contacts : un par pied, un pour chaque main, en faisant intervenir l'extrémité, et un par cuisse. $\mathcal{E}_{\text{surfaces}}$ compte 4 surfaces : le dessus de la table, son bord faisant face au robot, le dessus de l'estrade et le sol. Des contacts y sont prédéfinis. La tâche $\mathcal{T}_{\text{final}}$ demande à la main gauche d'être autour de la canette, en forçant un point lié à la main à être au centre de la canette, et un vecteur de la main à être selon l'axe de la canette.

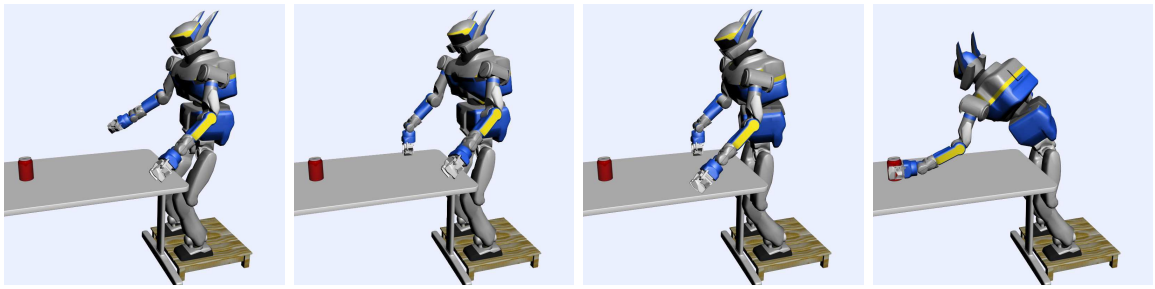


FIG. *5.8 – La séquence de postures témoins trouvée par le planificateur pour le scénario de la canette sur la table.*

La figure 5.8 présente la séquence de postures témoins résultat, obtenue en 5 secondes. Le robot commence sur ses deux pieds et pose sa main droite sur la table. Il appuie alors sa cuisse gauche contre la table puis retire sa main droite (cette étape n'est pas illustrée car la posture témoin est la même que la précédente) avant d'attraper la canette.

5.4.2 Marche généralisée

Le deuxième scénario a pour but de montrer un cas particulier de la planification d'appuis, à savoir la marche. Ici le robot doit arriver en haut d'un escalier. La condition de fin est d'avoir le centre de gravité dans une sphère en haut des escaliers. Les marches sont hautes de 15cm, larges de 80cm et longues de 30cm (un pied du robot fait environ 24cm par 14cm). Le robot HRP-2 est capable de passer au dessus d'un obstacle d'au plus 20cm, statiquement [Guan 05] ou dynamiquement [Verrelst 06].
Ici, $\mathcal{E}_{\text{contacts}}$ se réduit aux deux pieds. $\mathcal{E}_{\text{surfaces}}$ comprend le sol et le dessus de chaque marche, soit six surfaces. La stratégie de génération de contacts candidats fait appel à l'aléatoire. Les coordonnées (x, y) sont tirées au hasard sur l'intersection de chaque surface avec une sphère

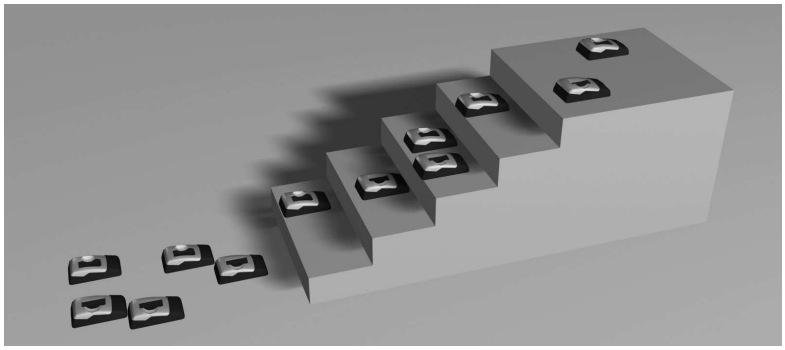


FIG. *5.9 – Les empreintes lors de la montée de l'escalier.*

de rayon 1m centrée à la racine (le bassin) du robot dans la configuration témoin du père. Le nombre de points tirés est proportionnel à la racine carrée de l'aire de cette intersection. La séquence finale est trouvée en 12 secondes et compte 22 éléments. Les postures témoins sont montrées par la figure 5.10. Les différentes positions des pieds au cours de cette séquence sont données par la figure 5.9. La moitié du temps de calcul est passée à trouver comment se positionner devant l'escalier pour mettre un premier pied dessus. On remarquera qu'au passage de la troisième marche, le planificateur ne trouve pas directement une solution pour passer à la marche d'après et doit donc poser les deux pieds dessus.

5.5 Conclusion

Dans ce chapitre, nous avons introduit les bases d'un planificateur d'appuis inspiré des méthodes par champ de potentiel, et détaillé ses modules. Les deux parties les plus importantes sont la génération de contacts candidats et la construction d'un champ de potentiel permettant une planification efficace. Nous avons proposé ici des méthodes simples pour ces deux modules et montré de premiers résultats dans des environnements simples, en particulier parce qu'ils sont peu contraints. Dans le chapitre suivant, nous en étudions des implémentations plus avancées et nous proposons des améliorations et extensions de notre planificateur.

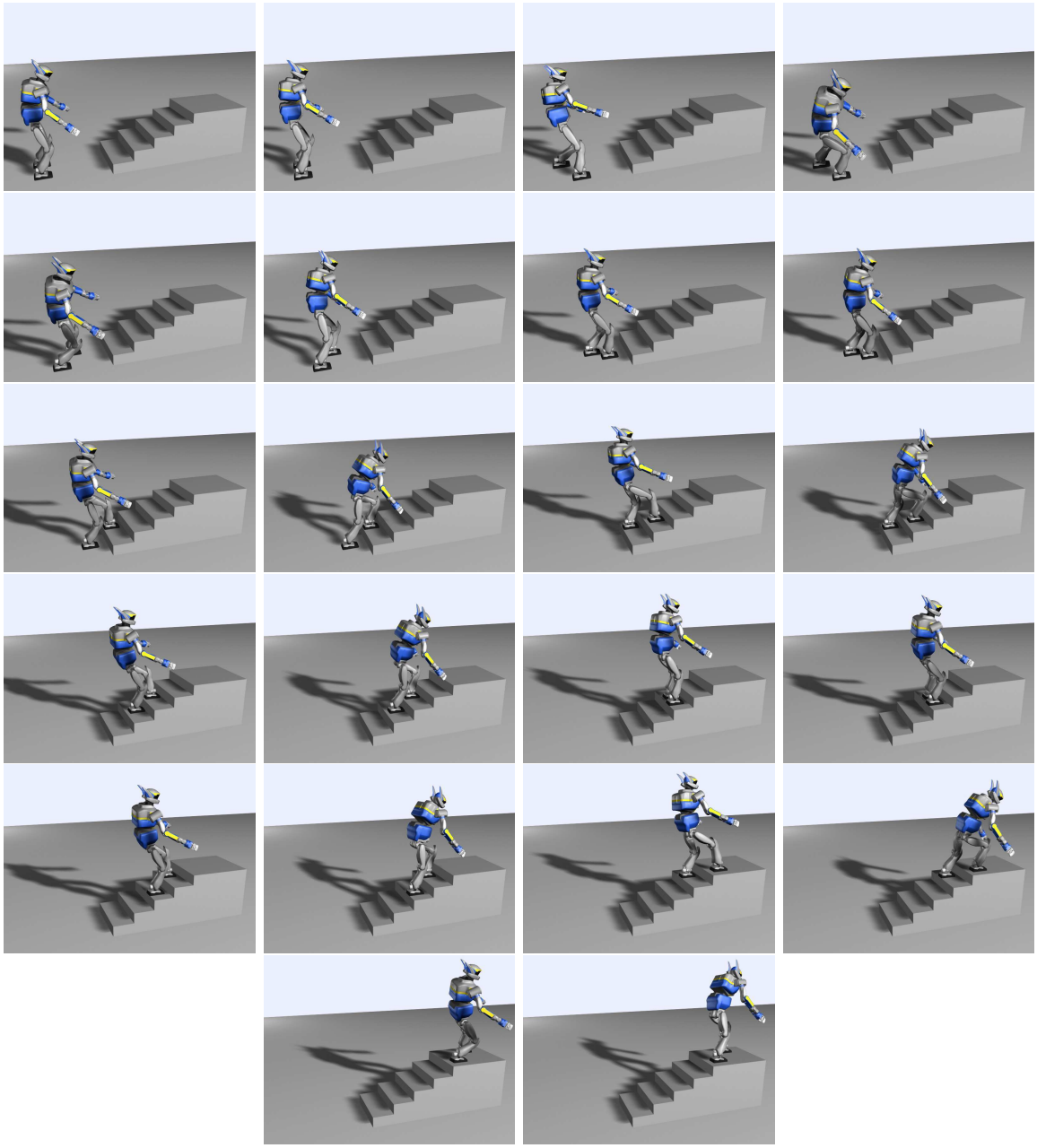


FIG. *5.10 – La séquence de postures témoins trouvée par le planificateur pour le scénario de l'escalier.*

Chapitre 6

Planification d'appuis : modules avancés

Faisant suite aux principes du planificateur d'appuis proposé dans le chapitre précédent, nous présentons ici des méthodes pour ses deux modules les plus cruciaux que sont la génération de nouveaux contacts et la construction du champ de potentiel, et proposons une façon de réduire la quantité de nœuds à générer. Grâce à ces méthodes notre planificateur peut alors s'attaquer à des problèmes complexes. Nous évoquons certains de ces scénarios ici, la plupart d'entre eux seront détaillés dans le chapitre suivant.

Ce chapitre s'organise comme suit : nous étudions d'abord le module de génération de points d'appuis (§ 6.1). Nous proposons ensuite un mécanisme qui permet d'économiser une grande partie de l'effort de calcul (§ 6.2), puis montrons comment construire un champ de potentiel efficace à partir d'une trajectoire guide (§ 6.3) dont une méthode de génération automatique est exposée au paragraphe 6.4. Enfin, profitant de l'utilisation du générateur de posture comme élément de base de notre planificateur, nous présentons une extension de ce dernier permettant d'enrichir la planification (§ 6.5).

6.1 Nouveaux contacts

6.1.1 Espace de recherche

Commençons par quelques rappels : on donne le nom de *contact fort* à un ensemble de contraintes qui fixent complètement la position et l'orientation d'un corps r_i du robot dans l'espace de travail (cf chap. 3). Ces contraintes reviennent à faire correspondre un repère $(P, \mathbf{d}_P, \mathbf{b}_P, \mathbf{n}_P)$ lié au corps r_i à un repère de l'environnement $(E, \mathbf{d}_E, \mathbf{b}_E, \mathbf{n}_E)$. De tels repères liés à des corps du robot sont définis par les éléments de $\mathcal{E}_{\text{contacts}}$ (le vecteur $\mathbf{b}_P$ s'obtenant par le produit vectoriel de la normale $\mathbf{n}_P$ et la direction $\mathbf{d}_P$).
Pour une surface régulière S^E de l'environnement seuls trois paramètres sont nécessaires pour définir entièrement le repère $(E, \mathbf{d}_E, \mathbf{b}_E, \mathbf{n}_E)$: deux coordonnées (x, y) et un angle θ,

par exemple. En effet, on a

- $\mathbf{E} = \mathbf{p}(x, y)$ où $\mathbf{E}$ est le vecteur de coordonnées du point E et $\mathbf{p}$ une paramétrisation de la surface,
- $\mathbf{n}_E = \mathbf{n}(\mathbf{E}) = \mathbf{n}(x, y)$, $\mathbf{n}$ renvoyant un vecteur normé pointant vers l'extérieur de l'objet dont S^E est une surface,
- un vecteur $\mathbf{d}_0$ étant donné, tel que pour tout couple (x, y), $\mathbf{n}(x, y)$ et $\mathbf{d}_0$ ne sont pas colinéaires (on suppose que la surface est telle que $\mathbf{d}_0$ existe), $\mathbf{d}_E = \mathbf{d}(x, y, \theta)$ où $\mathbf{d}$ est une fonction telle que (le symbole $\angle(a, b)$ désignant l'angle entre a et b) :

$$\begin{aligned} \|\mathbf{d}(x, y, \theta)\| &= 1 \\ \mathbf{d}(x, y, \theta).\mathbf{n}(x, y) &= 0 \\ \angle\left(\mathbf{d}(x, y, \theta), \mathbf{n}(x, y) \wedge \mathbf{d}_0 \wedge \mathbf{n}(x, y)\right) &= \theta \end{aligned}$$

- $\mathbf{b}_E = \mathbf{b}(x, y, \theta) = \mathbf{n}(x, y) \wedge \mathbf{d}(x, y, \theta)$.

En particulier, si S^E est un plan d'équation $ap_x + bp_y + cp_z = d$, $\mathbf{p}(x, y)$ se déduit de cette équation, $\mathbf{n}(x, y)$ est constant et égal au vecteur normé colinéaire à $(a, b, c)^T$ et pointant vers l'extérieur de la surface. $\mathbf{d}(x, y, \theta)$ et $\mathbf{b}(x, y, \theta)$ ne sont alors plus que des fonctions de θ.

De part les relations données ci-dessus, le simple choix d'une surface avec laquelle on veut mettre en contact un emplacement de contact de $\mathcal{E}_{\text{contacts}}$ engendre automatiquement un ensemble de contraintes qui correspond à ce que nous avons appelé un contact léger : le corps du robot est fixé dans trois de ses six degrés de liberté que sont la distance à la surface et les rotations autres que celle autour de la normale à la surface. Il lui reste trois degrés de liberté : deux en translation le long de la surface et un en rotation autour de la normale.

Pour un couple de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ donné, résoudre un problème de génération de posture faisant intervenir un contact léger permet de montrer qu'il est possible d'avoir un contact (fort) entre le corps et la surface du couple.

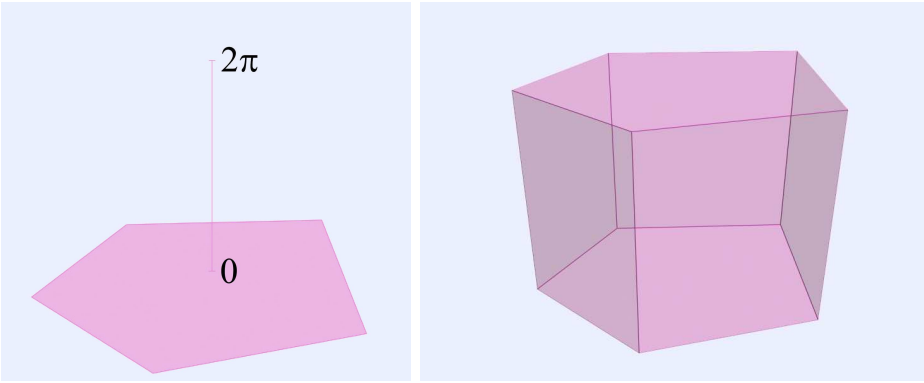


FIG. *6.1 – Construction d'un volume de recherche.*

Un contact léger est donc un contact fort paramétrable de paramètres (x, y, θ). L'ensemble de ces paramètres est un sous-ensemble de $\mathbb{R}^3$ qui s'identifie au produit $S^E \times [\,0, 2\pi\,[$ (cf Fig. 6.1). Un couple c de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ étant donné, cet ensemble, qu'on dénotera Υ^c, est l'espace de recherche de nouveaux contacts. Un contact fort entre le corps et la surface de ce couple se représente par un point dans Υ^c. La recherche de contacts candidats se ramène à la recherche de points dans cet espace.

6.1.2 Génération

6.1.2.1 Problème

Nous cherchons à résoudre au mieux le problème de génération de contacts candidats pour l'ajout de nouveaux contacts dans notre planificateur (fonction *getContactCandidates* de l'algorithme 4). Cette génération, comme on l'a vu, revient au choix de points dans un ensemble à trois dimensions Υ, choix qui doit répondre à trois contraintes que nous pouvons réécrire ainsi :

- (i) assurer une bonne couverture de Υ,
- (ii) ne pas générer de points trop proches entre eux,
- (iii) générer autant que possible des points qui correspondent à des contacts (forts) faisables.

C'est une redéfinition des critères donnés au chapitre 5. Bien que poursuivant les mêmes buts, les deux définitions ne sont pas strictement équivalentes. En particulier, on n'a plus complètement la notion du nombre de points aussi petit que possible, car un point étant intéressant ou non par les possibilités qu'il offre pour la suite, il est difficile de juger a priori de son utilité. On doit donc envisager suffisamment de points. (ii) en réduit cependant le nombre maximal.

Les points (i) et (ii) peuvent être atteints de différentes façons. Au chapitre précédent, nous avons par exemple utilisé un tirage aléatoire. Il suffirait de tirer à nouveau un point lorsqu'il est trop proche d'un autre.
Une autre idée est de prendre une discrétisation explicite de Υ. Le réglage du pas de cette discrétisation permet de répondre correctement à (i) et (ii).

Le problème reste le point (iii) : rien dans les méthodes proposées ci-dessus ne permet d'agir dessus. Pour $s \in \mathcal{S}$ donné, la projection du problème $\mathcal{Q}_s$ sur Υ est un sous-ensemble qui, si il n'est pas vide ou trivial, n'a aucune raison d'être simple, en particulier d'être convexe ou même connexe. Il est donc très difficile de biaiser le tirage aléatoire ou d'éliminer a priori des points de la discrétisation pour restreindre les points envisagés à cette projection.

6.1.2.2 Choix d'un point

Nous proposons ici une solution dont la motivation première est de résoudre le point (iii). L'idée est de laisser le générateur de posture choisir les points de Υ. Ceux-ci répondront alors directement à (iii).
Pour cela, nous définissons $\mathcal{Q}_{s,\Upsilon^c}$ comme étant le problème de génération de posture construit sur les contacts de s auquel on ajoute un contact léger basé sur $c \in \mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$, les paramètres du contact fort correspondant au résultat devant être dans Υ^c. Pour ne pas compliquer outre mesure le problème de génération, on supposera que Υ^c est convexe et délimitée par des plans. Au besoin, on découpera les surfaces de contact pour satisfaire cette hypothèse.

Cet ajout correspond à une tâche $\mathcal{T}_{\Upsilon^c}$ reprise du contact léger et définie ainsi (pour une

surface plane)

$$
\begin{aligned}
(\mathbf{P}-\mathbf{E})\,.\mathbf{n}_E &= 0 && (6.1)\\
\mathbf{d}_P.\mathbf{n}_E &= 0 && (6.2)\\
\mathbf{b}_P.\mathbf{n}_E &= 0 && (6.3)\\
-\mathbf{n}_P.\mathbf{n}_E &\leq 0 && (6.4)\\
(\mathbf{P}-\mathbf{E})\,.\mathbf{a}_i - b_i &< 0 \quad \forall i \in [|1,k|] && (6.5)\\
\theta^- \leq \angle(\mathbf{P}-\mathbf{E},\mathbf{d}_E) &\leq \theta^+ && (6.6)
\end{aligned}
$$

où E est un point de la surface, dont la normale est $\mathbf{n}_E$, $\mathbf{d}_E$ est le vecteur de direction associé à la surface et les $\mathbf{a}_i$ et b_i sont les paramètres des plans limitant cette surface.

Le problème $\mathcal{Q}_{s,\Upsilon^c}$ correspond à une génération de posture avec un contact *glissant*. Si une solution existe, la position de ce contact est définie par le critère d'optimisation de la génération. Dans le cas qui est le notre où nous utilisons une posture de référence, la solution renvoyée pour le problème $\mathcal{Q}_{s,\Upsilon^c}$ correspond au choix dans Υ^c du contact qui permet d'être le plus proche de cette référence.

6.1.2.3 Choix de plusieurs points

Nous avons jusqu'ici le moyen de choisir un point dans un ensemble Υ qui correspond à un contact faisable, nous donnant ainsi la possibilité de répondre au point (iii). Il nous reste à voir comment choisir plusieurs points en respectant (i) et (ii).

Étant donné un ensemble s de $\mathcal{S}$ et un couple c de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$, la résolution de $\mathcal{Q}_{s,\Upsilon^c}$ nous donne un point C_0. On veut par la suite pouvoir trouver d'autres points dans Υ^c, n'étant pas trop proches de C_0. Pour cela, on définit un volume V_0 interdit autour de C_0. Pour trouver un second point il suffirait alors de résoudre le problème $\mathcal{Q}_{s,\Upsilon'}$ avec $\Upsilon' = \Upsilon^c \backslash V_0$. On obtiendrait un point C_1, en déduirait un volume V_1 à retirer à Υ', etc. Cependant Υ' n'est pas convexe, et cela est susceptible de poser des problèmes à la génération de posture. De fait, nous avons observé, en retirant des ellipses autour des points trouvés, que cette méthode manquait des possibilités en s'égarant dans les minima locaux créés par ces volumes retirés. Ceci pouvait alors donner des résultats contraire au point (i).
Nous adoptons donc la stratégie suivante : lorsqu'un point est trouvé dans un ensemble Υ, on retire un volume polyédrique V autour de lui et on découpe $\Upsilon \backslash V$ en ensembles convexes Υ_i (cf Fig. 6.2). On résout alors une génération de posture dans chacun de ces nouveaux ensembles. Pour chaque Υ_i pour lequel une solution existe, on recommence alors : soustraction d'un volume puis découpage en sous-ensembles convexes Υ_i, si le volume à découper n'est pas vide (ce qui arrive si le volume à retirer contient entièrement l'espace de recherche). Et ainsi de suite jusqu'à ne plus avoir de volume Υ_i (soit parce qu'on ne trouve plus de posture, soit parce qu'on a retiré tous les volumes possibles).
En choisissant un volume polyédrique à retirer V dont les plans frontières sont soit du type $\theta = \theta_0$, soit $a_1x+a_2y = b$, on peut découper $\Upsilon \backslash V$ en volume Υ_i qui ont la même expression et donc avoir des tâches $\mathcal{T}_{\Upsilon_i}$ s'écrivant de la même façon que $\mathcal{T}_{\Upsilon^c}$. Nous utilisons un parallélépipède rectangle de taille $\delta l \times \delta l \times \delta\theta$. δl et $\delta\theta$ permettent de contrôler le densité des contacts trouvés. On obtient donc l'algorithme 5. Il remplace l'algorithme 4.

La fonction *testPreContact*(i,j) sera expliqué au paragraphe 6.1.3. L'ensemble $\mathcal{V}$ supporte deux fonctions : *first* renvoie le premier élément de $\mathcal{V}$ et le supprime de cet ensemble, *isEmpty*

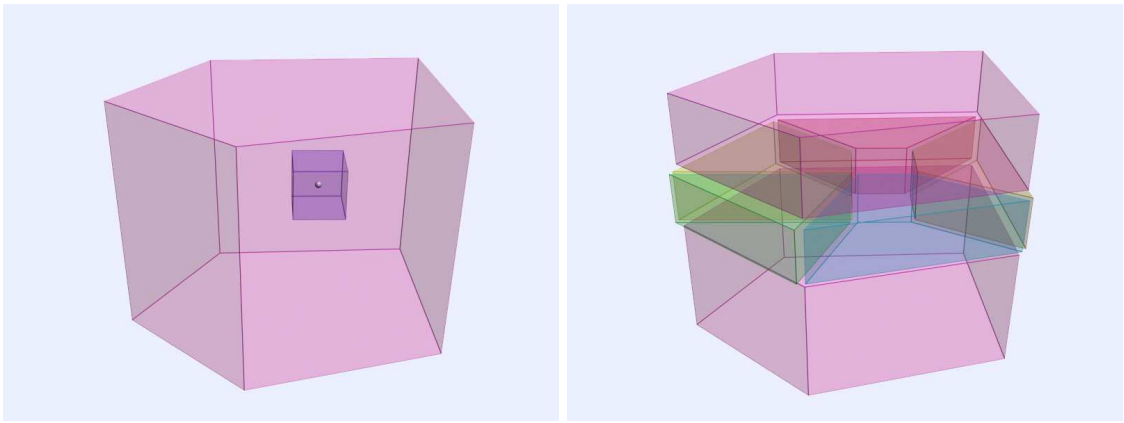


FIG. *6.2 – Subdivision d'un volume de recherche après y avoir trouvé un point.*

renvoie *vrai* si $\mathcal{V}$ est vide, *faux* sinon.
La fonction *constructSearchSpace*(j) renvoie l'ensemble de recherche convexe correspondant à la j$^{\text{éme}}$ surface de $\mathcal{E}_{\text{surfaces}}$. *addLightContact*($c$,$\Upsilon$,pci) rajoute à pci un contact léger correspondant au i$^{\text{ème}}$ corps et à la j$^{\text{ième}}$ de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$, et limité par l'espace de recherche Υ. *updateContact*(pci) réécrit le contact léger de pci en le contact fort correspondant à la solution trouvée par la génération de posture. Enfin *getNewSearchSpaces*(Υ,pci) divise Υ en sous-ensembles convexes après lui avoir retiré un volume autour du point correspondant à la solution de la génération de posture pour pci, et renvoie l'ensemble de ces sous-ensembles.

On remarquera que la construction du nouveau nœud n' et le test d'existence d'un nœud similaire dans N est remis à après la génération de posture. En effet, le nouveau contact n'est fixé qu'une fois la posture connue, la comparaison des ensembles de contacts ne pouvant se faire qu'à ce moment.

6.1.3 Pré-contacts

Nous avons évoqué au chapitre 3 le fait que la génération de posture prend plus ou moins de temps à s'arrêter sur un échec quand elle ne peut pas ou n'arrive pas à retourner une solution, et que ce temps dépend du type d'échec. En particulier, le fait de ne pas réussir à satisfaire les contraintes d'inégalité est généralement plus rapide que celui de ne pas satisfaire les contraintes d'égalités non linéaires. Or un contact, léger ou fort, est avant tout constitué d'égalités non linéaires. Les inégalités qui en font partie ne sont pas très contraignantes.
Avant de chercher à résoudre la tâche liée à un contact léger, on voudrait pouvoir savoir si le corps envisagé peut s'approcher suffisamment de la surface pour avoir de bonne chance qu'une solution au contact léger existe. C'est le but de la fonction *testPreContact*(pc,i,j) : elle construit le problème $\mathcal{Q}_s$ où s est l'ensemble de contacts de pc et lui rajoute un pré-contact tel que défini au chapitre 3 entre l'emplacement de contact et la surface dénotés respectivement par i et j, ainsi que les inégalités limitant la surface d'indice j. La fonction renvoie *vrai* si la génération de posture trouve une solution à ce problème, *faux* sinon.

La définition du pré-contact fait toujours apparaître des égalités sur les directions qu'on pourrait vouloir relaxer en inégalités. Cependant l'inégalité sur la distance du corps à la surface est suffisamment discriminante par elle même pour que la fonction *testPreContact*

Algorithm:generateNewContacts2
Data: *pc*, une structure de type PostureCalculator
Data: i et j, indices respectivement de l'emplacement de contact contact et de la surface
Result: l, un vecteur de feuilles descendant de ce nœud

```
1  - pci est une structure de type PostureCalculator
2  - V et V' sont des ensembles d'espaces de recherche
3  - N est un ensemble sans duplicata contenant tous les nœuds ou feuilles déjà générés
4  begin
5      l ← ∅
6      if testPreContact(pc,i,j) then
7          return(l)
8      end
9      if forbiddenContact(i,j) then
10         return(l)
11     end
12     V ← constructSearchSpace(j)
13     while no isEmpty(V) do
14         Υ ← first(V)
15         pci ← copy(pc)
16         addLightContact(i,j,Υ,pci)
17         if generate(pci) then
18             updateContact(pci)
19             n' ← newNode(pci)
20             V' ← getNewSearchSpaces(Υ,pci)
21             V ← V ∪ V'
22             if add(n',N) then
23                 add(n', l)
24             end
25         end
26     end
27     return(l)
28 end
```

Algorithm 5: Génération de nouveaux contacts pour un couple (emplacement de contact, surface) avec les espace de recherches.

remplisse bien son rôle d'évitement de calculs inutiles.
En cas de réussite de la génération de cette fonction, le calcul n'est pas perdu car la posture trouvée peut servir d'initialisation aux générations suivantes dans l'algorithme 5.

6.2 Very Best First Planning

Avec l'amélioration proposée sur la génération de nouveaux contacts et l'utilisation de bons champs de potentiel comme celui que nous proposons à la section 6.3, notre planificateur

est capable de s'attaquer à des problèmes complexes. Cependant la résolution de ceux-ci est encore très longue (plusieurs heures). Une des raisons à cela est que le programme passe beaucoup de temps pour générer des feuilles qui ne seront pas utilisées par le suite. Le nombre de ces feuilles "inutiles" fluctue de manière importante selon les différents paramètres liés à la discrétisation et la complexité de l'espace environnant. Dans un environnement ouvert avec cependant plusieurs surfaces de contact, on voit apparaître des nœuds avec plus d'une centaine de fils, ce qui n'est pas le cas lorsque l'espace est restreint. Pour de tels nœuds, moins de trois fils sont généralement considérés par le planificateur. Nous exposons ici une méthode pour réduire de manière importante le nombre de feuilles générées, permettant de diminuer le temps de calcul de manière significative.

6.2.1 Principe

Revenons momentanément à l'algorithme BFP. A la $i^{\text{ème}}$ itération de l'algorithme, BFP regarde l'ensemble l_i des voisins non visités de la configuration courante $\mathbf{q}^i$, puis les insère dans une liste L triée par ordre croissant de potentiel, dont elle prendra le meilleur élément $\mathbf{q}^{i+1}$ pour l'itération suivante. A l'itération $i+1$, ce meilleur élément de L est soit le meilleur élément de l_i, l_i^0, soit le meilleur de L au début de l'itération précédente, après le retrait de $\mathbf{q}^i$. A ce moment, les autres élément de l_i n'ont pas d'intérêt et peuvent ne pas être rajoutés à L.
Si en revanche à l'itération k, $k > i$, l_i^0 est sélectionné, il est retiré de L et le second élément de l_i, l_i^1 est susceptible d'être meilleur que le meilleur élément de L. Il faut donc, pour la bonne marche de l'algorithme qu'il fasse partie de L. De même il faudrait ajouter l_i^2 à L si l_i^1 venait à être sélectionné.
A tout instant, le meilleur élément de L est le meilleur des meilleurs voisins non visités et non sélectionnés de chaque nœud interne de l'arbre.
On rappelle que non visité signifie qu'un voisin n'est pas déjà le voisin d'un nœud précédemment considéré et non sélectionné désigne une feuille qui n'a (par définition) pas été choisie, et n'est donc pas un nœud interne de l'arbre construit par BFP.

Prenons maintenant l'hypothèse suivante : on dispose d'un oracle qui nous permet de savoir, sans faire les calculs, dans quel ordre de potentiel sont rangés les voisins d'une configuration. En particulier, on peut savoir quel est le meilleur voisin vis-à-vis du champ de potentiel. Alors, au vu des remarques précédentes, il n'est pas besoin de construire entièrement les listes l_i à chaque itération. On peut les construire à la demande, de façon à ce que chacune ait toujours son meilleur élément non sélectionné dans L, sauf à avoir été entièrement construite et avoir eu tous ses éléments sélectionnés. En appliquant cela on peut donc se restreindre sur le nombre de configurations à visiter et en lesquelles il faut évaluer la valeur du champ de potentiel. La limite de cette restriction est la suivante : il faut, tant que cela est possible, que chaque nœud interne de l'arbre ait à tout instant un fils dans L. Cela est indispensable pour pouvoir revenir en arrière quand l'algorithme a fait fausse route du fait d'un minimum local.

L'algorithme modifié en conséquence, pour tirer parti de ces remarques, commence toujours par le *meilleur des meilleurs*. Nous lui donnons donc le nom de *Very Best First Planning*. Il est présenté par l'algorithme 6

La première partie de la boucle *while* assure, tant que cela est possible, qu'il y a le représentant d'une fratrie dans L. Pour cela la fonction *bestYoungBrother*($\mathbf{q}$) renvoie le

Algorithm:VBFP
Data: $\mathbf{q}^{\text{init}}$, $\mathbf{q}^{\text{final}}$,$U(\mathbf{q})$
Result: une suite $\left[\mathbf{q}^0, \mathbf{q}^1, ..., \mathbf{q}^l\right]$, avec $\mathbf{q}^0 = \mathbf{q}^{\text{init}}$, $\mathbf{q}^l = \mathbf{q}^{\text{final}}$ et $\mathbf{q}^{i+1}$ voisin de $\mathbf{q}^i$ pour $0 \leq i < l$

```
1  - L est une liste triée, initialement vide
2  - T un arbre, initialement vide
3  begin
4      insert(q^init,L)
5      add(q^init,T)
6      while no isEmpty(L) do
7          q ← first(L)
8          q' ← bestYoungBrother(q)
9          add(q',T)
10         insert(q',L)
11         q' ← bestSon(q)
12         add(q',T)
13         insert(q',L)
14         if q' = q^final then
15             return backtrackPath(q')
16         end
17     end
18     return failure
19 end
```

Algorithm 6: Algorithme Very Best First Planning.

meilleur voisin *non visité* du père de $\mathbf{q}$, si il existe.
La seconde partie reprend le cœur de la boucle dans l'algorithme BFP, mais se restreint au seul meilleur voisin *non visité* de $\mathbf{q}$ qui est renvoyé par la fonction *bestSon*($\mathbf{q}$), si il existe.
Dans les deux cas, le concept de *non visité* (qui implique non sélectionné) est testé comme dans l'algorithme BFP, en vérifiant la non appartenance à l'arbre T. Les fonctions *bestYoungBrother* et *bestSon* vérifie aussi que le potentiel de la configuration renvoyée est inférieur au seuil M. Si il n'existe pas d'éléments passant ces deux tests, ces fonctions renvoient $\oslash$. Les fonctions *add* et *insert* ne font rien dans ce cas. Les configurations renvoyées ont leur pointeur vers leur père correctement initialisé.

On remarquera que le test de fin n'est fait ici que sur le meilleur fils : cela suppose que le potentiel est bien minimal en $\mathbf{q}^{\text{final}}$. Pour relâcher cette hypothèse, il faut aussi faire le test pour chaque frère généré.

La figure 6.3 permet de comparer le nombre de configurations mises en jeu par BFP et VBFP dans le cadre du scénario 2D présenté au chapitre précédent. L'arbre dans le cas de BFP compte, hors configuration finale, 28 nœuds internes et 35 feuilles. Celui de VBFP n'a que 23 feuilles. En dimensions plus élevées, la différence est beaucoup plus importante : en trois dimensions, pour un potentiel qui engendrerait la même planification (par exemple un potentiel qui se diviserait en celui de la figure et la distance au plan de la figure), BFP générerait 56 feuilles de plus, soit 91 feuilles, alors que VBFP serait à 28 feuilles (une par

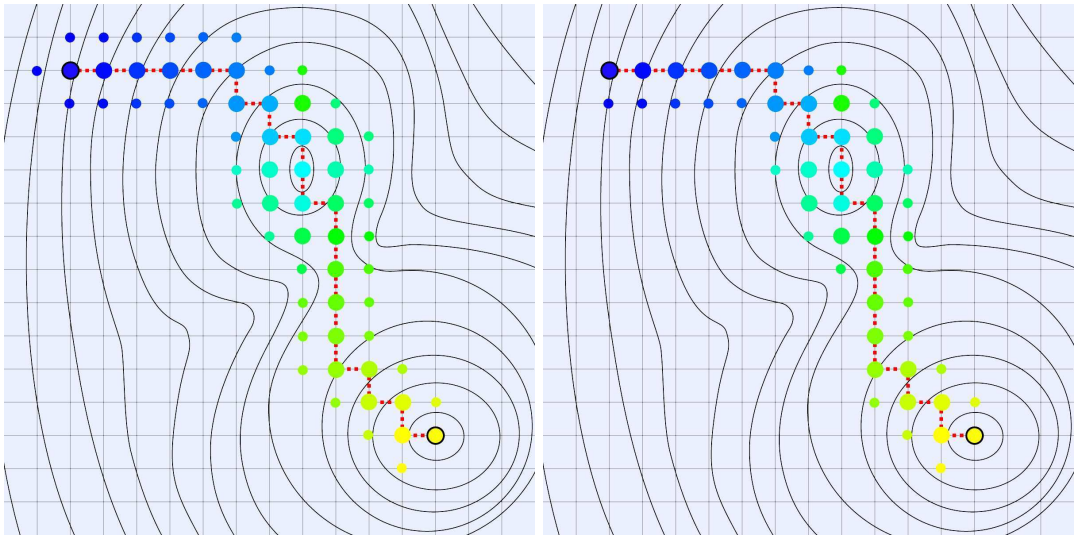


FIG. *6.3 – Différence pour les configurations visitées par BFP (à gauche) et VBFP (à droite).*

nœud, la dernière étant la configuration finale). En quatre dimensions, ce serait 147 feuilles à 28.

Il ne faut cependant pas ignorer que dans le pire des cas (pas de solution), les deux algorithmes vont visiter le même nombre de configurations (toutes). A une échelle moindre, le remplissage d'un minimum local requiert quasiment le même effort de la part des deux algorithmes. La différence se fait sur le nombre de feuilles au bord du bassin d'attraction de ce minimum, différence d'autant moins significative que le bassin est large. Les gains de temps de VBFP se font donc surtout dans les parties où l'algorithme peut suivre la plus grande pente.

6.2.2 Application à la planification d'appuis

Nous l'avons déjà mentionné, la plus grande partie du temps d'exécution de notre planificateur d'appuis est passée dans la validation des ensembles de contacts par la génération de posture. Pouvoir adapter les principes du VBFP à notre cas serait avantageux à deux titres : tout d'abord moins de feuilles à générer, et donc moins de génération de postures, mais en plus, une plus grande proportion d'échecs de génération évités. En effet pour un couple de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$, notre méthode de génération de nouveaux contacts implique qu'on commence par des générations réussies et finissent par des échecs. S'il n'y avait pas besoin de chercher à générer tous les nouveaux contacts possibles pour ce couple, on éviterait de devoir aller systématiquement jusqu'à l'échec.
Mais pour faire cette adaptation, il faut avoir un oracle. La façon dont nous gérons les nouveaux contacts permet d'en avoir un partiel.

Dans la section 6.1, nous utilisons le critère de la génération de posture pour choisir un nouveau contact, sans a priori sur la forme du critère. Pour un espace de recherche donné Υ, le contact obtenu est à alors celui qui permet la meilleure posture au sens du critère d'optimisation. On sait que tous les contacts qui seront trouvés dans les restes de cet espace (les Υ_i) donneront des configurations moins bonnes que celle-ci. Pour avoir les mêmes conclusions au

sens du champ de potentiel, *il suffit d'utiliser celui-ci, ou tout au moins sa partie définie sur $\mathcal{C}$, comme critère d'optimisation pour la génération de posture.*

Nous avons ainsi le moyen d'ordonner partiellement les fils d'un nœud. Ce moyen est partiel pour deux raisons :

- il ne permet de prédire le classement que pour les fils issus d'un même contact léger, on ne peut rien dire pour ceux générés pour des couples de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ différents ou issus du retrait d'un contact,
- pour un même couple, on ne peut classer complètement les fils : on peut juste dire que le contact obtenu dans un espace de recherche Υ sera meilleur que ceux trouvés dans ses sous-espaces fils Υ_i.

Algorithm: Contacts VBFP
Data: $(\mathbf{s}_{\text{init}}, \mathbf{q}^{\text{init}})$, $\mathcal{T}_{\text{final}}$,$U(\mathbf{q}, \mathbf{s})$
Result: une suite $\left[(\mathbf{s}_0, \mathbf{q}^0), (\mathbf{s}_1, \mathbf{q}^1), ..., \left(\mathbf{s}_l, \mathbf{q}^l\right)\right]$, avec $(\mathbf{s}_0, \mathbf{q}^0) = (\mathbf{s}_{\text{init}}, \mathbf{q}^{\text{init}})$, $\mathbf{q}^l$ vérifiant $\mathcal{T}_{\text{final}}$ et $\mathbf{s}_{i+1}$ voisin de $\mathbf{s}_i$ pour $0 \leq i < l$

```
1  begin
2      n ← newNode(s_init, q^init)
3      insert(n,L)
4      while no isEmpty(L) do
5          n ← first(L)
6          if allowsToReachGoal(n) then
7              return backtrackPath(n)
8          end
9          l ← generateBestSons(n) ∪ generateNextBrothers(n)
10         for each node n' in l do
11             if U(n') < M and trajectoryExists(n') then
12                 insert(n',L)
13             end
14         end
15     end
16     return failure
17 end
```

Algorithm 7: Algorithme Contacts VBFP.

L'algorithme 7 reprend l'algorithme 6 en intégrant les modifications rendues possibles par l'oracle partiel. La différence tient à l'ordre de génération des nœuds qui se retrouve à la ligne 9, avec ses deux nouvelles fonctions.
generateBestSons(n) est très similaire à la fonction *generateSons* décrite par l'algorithme 3 : il génère les nœuds dus à la suppression d'un contact mais ne génère qu'un seul nœud par couple de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ (si bien sûr ce nœud existe) lors de la création de nouveaux contacts. Le changement se fait donc uniquement au niveau de la fonction *newContacts*. Celle-ci est une version allégée de la fonction de l'algorithme 5 qui n'effectue qu'une seule itération de la boucle *while* et ne construit pas les sous-ensembles Υ_i. Elle est décrite par l'algorithme 8.

Les sous-ensembles Υ_i auront cependant peut-être besoin d'être calculés ultérieurement, auquel cas il faudra se souvenir de l'espace Υ. C'est la fonction de *remember*(n,Υ) que d'at-

```
Algorithm:generateNewContactsVBFP
begin
    l ← ⊘
    if testPreContact(pc,i,j) then
        return(l)
    end
    if forbiddenContact(i,j) then
        return(l)
    end
    ϒ ← constructSearchSpace(j)
    pci ← copy(pc)
    addLightContact(i,j,ϒ,pci)
    if generate(pci) then
        updateContact(pci)
        n' ← newNode(pci)
        remember(n', ϒ)
        if add(n',N) then
            l ← {n'}
        else
            l ← generateNextBrothers(n') ;
        end
    end
    return(l)
end
```

Algorithm 8: Version de generateNewContacts pour l'algorithme Contacts VBFP.

tacher à un nœud n l'ensemble Υ pour lequel il a été généré.
Si jamais un nœud n' est trouvé par *generateNewContactsVBFP*, mais n'est pas accepté dans N, il faut alors essayer de lui trouver un "remplaçant". C'est le but du *else*.

La fonction *generateNextBrothers*(n) a une structure similaire. Son rôle est de générer les meilleurs frères du nœud n pour le même couple de $\mathcal{E}_{\text{contacts}} \times \mathcal{E}_{\text{surfaces}}$ que n. Pour cela, il n'y a pas d'autre choix que d'essayer une génération dans chacun des Υ_i issus directement du Υ lié à n. En effet, on ne sait pas classer a priori les résultats obtenus pour deux ensembles Υ_i différents. Cette fonction est détaillée par l'algorithme 9.

On sait que *generateNextBrothers* est appelée soit pour un nœud n qui a été généré par appel à *generateNewContactsVBFP*, c'est-à-dire un nœud obtenu pour un ensemble de contacts auquel on a ajouté un contact spécifié par un couple d'indices (i, j) et un espace de recherche Υ, soit pour le frère d'un tel nœud, généré plus tard, pour les mêmes indices mais un Υ différent. *generateNextBrothers* n'a donc en particulier pas à refaire les tests qui sont en entrée de *generateNewContactsVBFP*.
Les fonctions *getIndexes*(n), *getCalculator*(n) et *getSearchSpace*(n) renvoient respectivement les indices (i, j), le *PostureCalculator* lié à n, et l'espace Υ qui a servi à la génération du nouveau contact de n.
La fonction *getBasisCalculator*(n) renvoie le *PostureCalculator* lié à n auquel a été enlevé le nouveau contact qui avait donné naissance à n.

```
   Algorithm:generateNextBrothers
   Data: n, un nœud
   Result: l, un vecteur de frère de ce nœud
 1 begin
 2     l ← ∅
 3     (i, j) ← getIndexes(n)
 4     pc ← getCalculator(n)
 5     ϒ ← getSearchSpace(n)
 6     V ← getNewSearchSpaces(ϒ,pc)
 7     pc ← getBasisCalculator(n)
 8     for each ϒ in V do
 9         pci ← copy(pc)
10         addLightContact(i,j,ϒ,pci)
11         if generate(pci) then
12             updateContact(pci)
13             n' ← newNode(pci)
14             remember(n', ϒ)
15             if add(n',N) then
16                 add(n', l)
17             else
18                 add(generateNextBrothers(n'), l)
19             end
20         end
21     end
22     return(l)
23 end
```

Algorithm 9: Fonction *generateNextBrothers*.

L'utilisation de l'algorithme *Contacts VBFP* permet empiriquement 3 à 5 fois moins de générations de nœuds et une diminution du temps de calcul nécessaire d'autant, pour trouver la solution des différents scénarios que nous avons testés. Du fait du moins grand nombre de nœuds, on observe aussi un gain appréciable de mémoire.

6.3 Trajectoire guide et champ de potentiel

Parce que tout objet de l'environnement offre possiblement une surface d'appuis, nous ne pouvons systématiquement associer un champ de potentiel répulsif à cet objet, en le considérant comme un obstacle. Cependant, baser notre champ de potentiel uniquement sur le but à atteindre n'est pas une solution acceptable dès lors que le scénario envisagé est un peu complexe : même si ils ne sont pas pris en compte dans le potentiel, les objets sont présents dans l'environnement, et tels des contraintes dans un problème d'optimisation, sont susceptibles de créer des minima locaux. Un simple cube dont la longueur du côté est proche de la hauteur du robot, et qui est posé sur le sol entre le robot et son objectif crée un minimum d'autant plus insurmontable que le cardinal de $\mathcal{E}_{\text{contacts}}$ est grand. Nous proposons

dans ce paragraphe une méthode pour créer un champ de potentiel permettant de prendre correctement en compte les objets de l'espace. Nous basons ce potentiel sur une *trajectoire guide*.

6.3.1 Trajectoire guide

Dans ce travail, une trajectoire guide est une trajectoire approximative dans l'espace de configuration, qui donne une idée générale de ce à quoi devrait ressembler le résultat de la planification d'appuis. Elle est approximative au sens où elle n'est pas précise : elle ne correspond pas nécessairement à un robot en contact avec son environnement, elle peut être légèrement en collision avec les $\mathcal{C}$-obstacles, le résultat pourra s'en éloigner momentanément en particulier parce qu'elle ne s'intéresse pas aux mouvements locaux du robot. **Elle doit être un guide pour la planification de contact**.

Par exemple pour un robot debout sur un sol plat dans un environnement sans autre objet que le sol et devant se rendre à un point de cet environnement, une trajectoire guide serait celle ou le robot flotte en ligne droite légèrement au-dessus du sol vers son objectif avec une pose droite. La solution de la planification d'appuis oscillerait autour d'elle, au gré des enjambées qui sont autant d'écarts à la pose droite, cependant la trajectoire serait *globalement suivie*.

Pouvoir donner une trajectoire guide que la planification d'appuis suivrait a plusieurs motivations. La première et la plus importante est de donner une indication sur la manière de contourner des obstacles. En retournant à l'exemple avec le cube, suivre une trajectoire qui ferait passer le robot par la droite ou la gauche empêcherait celui-ci de se retrouver bloqué face au cube.
La seconde motivation tient à la remarque suivante : du point de vue du planificateur d'appuis, tous les chemins menant à l'objectif se valent a priori. Par exemple, si notre robot se trouve dans le jardin d'une maison et doit se rendre au grenier, il pourra passer par la porte, emprunter l'escalier jusqu'au premier étage puis l'échelle de la trappe d'accès du grenier, ou bien monter à l'échelle que le peintre a laissée sous une fenêtre ouverte pour ensuite rejoindre l'échelle de la trappe, ou enfin profiter des belles prises offertes par les pierres de taille aux angles de la maison pour en escalader un, puis passer par le conduit de la cheminée du grenier. Ces diverses possibilités ont une difficulté et une esthétique différentes, elles sont néanmoins toutes valides et si on disposait de suffisamment de temps et de mémoire pour résoudre un tel problème de planification avec un potentiel attracteur simple, on pourrait tomber sur l'une des trois. L'utilisation d'une trajectoire guide permet de choisir quel chemin le robot devra suivre de manière approchée.

Des exemples précédents, il ressort que la position absolue du robot joue un grand rôle : éviter un obstacle, passer par un endroit précis, tout cela se fait dans l'environnement et met en jeu avant tout la position. Cependant, si les passages deviennent plus étroits, l'orientation puis la posture deviennent importantes. Certains passages restreignent fortement la forme possible des postures. On a donc bien besoin d'une trajectoire guide dans l'espace de configuration, pas seulement sur l'espace de travail. De plus, nous voulons construire un champ potentiel sur cette trajectoire guide. Or la nouvelle écriture du planificateur, donnée au paragraphe 6.2, utilise le potentiel comme critère pour la génération de posture. Il faut donc que

ce potentiel, et par construction la trajectoire dont il est issu, intègre une composante sur la posture.

Nous définissons une trajectoire guide T dans $\mathcal{C}$ comme une courbe affine par morceau spécifiée par la donnée de points clés K_i entre lesquels nous interpolons linéairement. Cette courbe peut-être en collision (légère) avec les $\mathcal{C}$-obstacles même aux points clés. Le premier et le dernier point clé sont idéalement respectivement proches de la configuration initiale et de celle finale du problème.

6.3.2 Construction d'un champ de potentiel sur une trajectoire guide

Nous cherchons à construire un champ de potentiel U_T sur $\mathcal{C}$ basé sur la trajectoire T tel qu'il incite le planificateur à rester proche de T et à avancer le long de T. Un tel potentiel a une forme de vallée descendante.
Pour satisfaire ces deux buts, l'idée est de définir U_T en deux parties, basées respectivement sur la distance à la courbe et sur la distance parcourue le long de cette courbe. En termes de distance à la courbe la posture entière nous intéresse, cependant la distance parcourue le long de la courbe a plus de sens si l'on regarde seulement la position et l'orientation absolue du robot.
Soit $\mathcal{F}$ le sous-espace de $\mathcal{C}$ qui se réduit à la position et l'orientation du corps racine du robot, et soit p la projection de $\mathcal{C}$ sur $\mathcal{F}$. On appelle T' l'image de T par p. On suppose que p est bijective de T sur T', c'est-à-dire que la courbe T ne prend pas plusieurs fois la même position et orientation. Cette hypothèse est réaliste : elle est mise à défaut lorsqu'il y a une boucle de T', ce qui signifie que le robot a pris un chemin inutile, ou lorsqu'il doit changer de posture en gardant sa racine fixe. Cependant dans ce second cas on peut arranger la courbe pour que la racine bouge légèrement pendant le changement de posture, la trajectoire n'étant qu'approximative. On note par ailleurs $\mathbf{q}' = p(\mathbf{q})$, $\mathbf{m}'$ le point de T' le plus proche de $\mathbf{q}'$ (point témoin) et l' l'abscisse curviligne de $\mathbf{m}'$. $\mathbf{m}'$ étant un point de T', on peut définir $\mathbf{m} = p^-1(\mathbf{m}')$.
On construit alors le champ de potentiel

$$U_T(\mathbf{q}) = \|\mathbf{q} - \mathbf{m}\|^2 - k \cdot l' \tag{6.7}$$

ou un peu plus évolué :

$$U_T(\mathbf{q}) = \begin{cases} \|\mathbf{q} - \mathbf{m}\|^2 - k \cdot l' & \text{si } \|\mathbf{q} - \mathbf{m}\| \leq d_0 \\ d_0 \|\mathbf{q} - \mathbf{m}\| - \frac{1}{2} d_0^2 - k \cdot l' & \text{si } \|\mathbf{q} - \mathbf{m}\| > d_0 \end{cases} \tag{6.8}$$

Les deux membres du potentiel sont les deux parties évoquées plus haut. k est un paramètre qui permet de moduler l'effet respectif de ces deux contributions. Le fait de passer par $\mathcal{F}$ pour trouver le point témoin donne une signification différente à la position/orientation et à la posture "interne" : on obtient un découpage entre degrés de liberté internes et externes qui peut s'interpréter comme le fait qu'on a une trajectoire en position/rotation et qu'on associe une posture de référence à chaque point de cette trajectoire.

Le champ ainsi défini n'est cependant pas C^0 : les points $\mathbf{m}'$ sautent lorsque $\mathbf{q}$ passe de part et d'autre des hyperplans bissecteurs à chaque angles de T ce qui entraîne une discontinuité de l'. Or on voudrait avoir un potentiel au moins C^1 pour pouvoir s'en servir comme

critère pour le générateur de posture.
Pour avoir un champ de distance à un objet tel que T régularisé, il existe une théorie intéressante qui est celle des R-fonctions [Shapiro 91] [Shapiro 99] [Biswas 04]. L'utilisation de telles fonctions, qui permettent la création de pseudo-distances avec le degré de continuité souhaité, serait parfaite pour la partie distance à la courbe. Cependant, il n'est plus possible avec ces fonctions de définir des points témoins ce qui empêche l'utilisation d'abscisses curvilignes et donc la définition du second terme du potentiel.

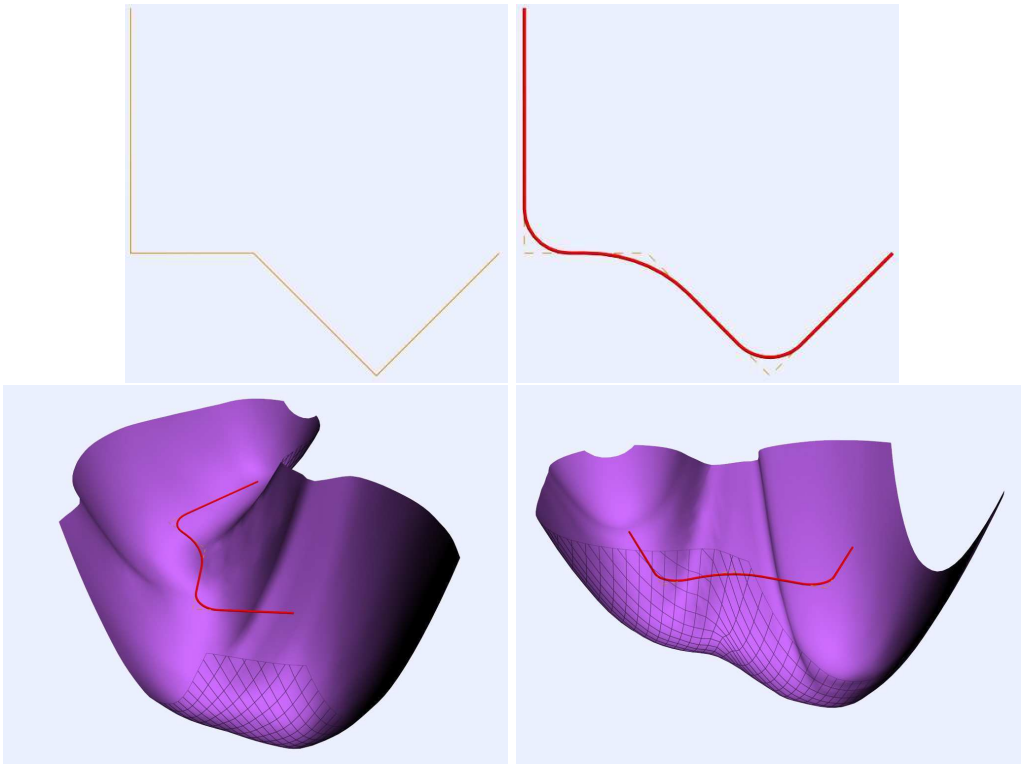


FIG. 6.4 – *Construction d'un champ de potentiel basé sur une trajectoire guide. Une trajectoire affine par morceaux (en haut à gauche) est arrondie à ses coins (haut, droite), avant de servir de base à la construction d'un champ de potentiel en forme de vallée descendante (bas).*

Puisqu'il semble difficile de régulariser le champ lui-même, nous avons choisi de modifier légèrement la trajectoire T pour que le champ soit C^1 au moins lorsque l'on se trouve proche de T. Pour cela, on arrondi les angles de T en utilisant des arcs de cercles en dimension $n+6$. On cherche à prendre les rayons de ces cercles aussi grands que possible sans pour autant que le cercle ne dépasse une certaine distance à la courbe originale : la continuité C^1 est assurée tant qu'on se trouve plus proche de la courbe que le rayon des cercles. Cette continuité "locale" est suffisante pour nous. Avec cette trajectoire modifiée, on garde la même forme de potentiel U_T (cf Fig. 6.4).
On peut par suite définir sur $\mathcal{S} \times \mathcal{C}$ le champ de potentiel $U(s, \mathbf{q}) = U_T(\mathbf{q}) + U_S(s)$, où U_S est un potentiel comportant des bonus ou malus dépendant de s comme discuté au chapitre précédent. Seul U_T est cependant utilisé pour la génération de posture.

6.3.3 Trajectoires individuelles de corps

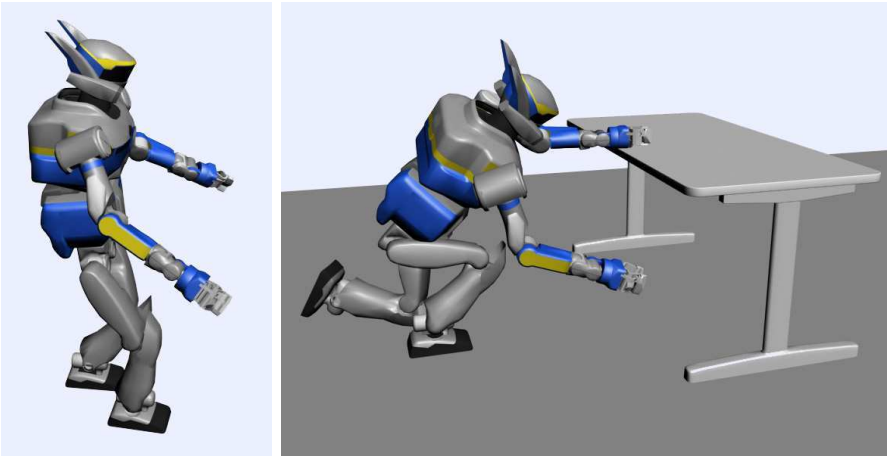


FIG. *6.5 – Deux exemples où pour avancer il faut s'écarter de la posture de référence donnée par la trajectoire guide.*

L'utilisation d'un champ de potentiel basé sur une trajectoire permet de résoudre le problème des minima locaux importants dus aux obstacles, cependant cela n'élimine pas des minima plus petits qui sont liés à la nature intrinsèquement discrète de la planification d'appuis. En effet, parce que les déplacements dans $\mathcal{S}$ contraignent les mouvements possibles dans $\mathcal{C}$, il n'est pas toujours possible de bouger dans le sens de la plus grande pente, on a donc des formations de minima locaux. La projection de $U(\mathbf{q})$ sur $\mathcal{S}$ fait apparaître des minima locaux. Dans le cadre d'un potentiel basé sur une trajectoire, une traduction de ce type de problème est que pour pouvoir suivre la trajectoire il faut parfois s'en éloigner momentanément. La figure 6.5 donne deux exemples de tels cas : faire un pas, comme sur la figure de gauche, demande à s'éloigner de la posture debout de référence. Le terme d'avancement $-k \cdot l'$ aide cependant à faire des pas, mais ceux-ci sont petits. Dans le cas où le robot doit avancer à quatre pattes, les pas sont encore plus petits, car pour un même écart d'angle que dans le cas de la marche, les membres du robot bouge moins. Mais le robot avance quand même.
Le cas de la figure de droite est plus problématique : le robot doit passer sous la table. A cette fin, il lui faut passer d'une posture debout à une posture quatre pattes. La trajectoire guide indique bien cela. Cependant, passer d'une posture à une autre demande à utiliser des postures intermédiaires qui ne sont ni proches de la posture debout, ni proches de celle à quatre pattes. On est là face à un minimum local avec un large bassin d'attraction dont il sera coûteux de s'échapper. De manière générale, le passage entre deux postures suffisamment différentes engendrera ce type de problème. Si en revanche les postures sont proches, comme assis et debout, cette difficulté n'apparaît pas.

Dans les deux cas précédents, le problème peut être interprété comme un manque d'anticipation sur ce qui arrive ensuite. L'anticipation pousserait à franchir les cols des minima. Nous proposons comme solution à cela d'utiliser des potentiels semblables à U_T mais agissant seulement sur certains corps du robot : on déduit les trajectoires (uniquement en position) T_i de chaque corps du robots à partir de T et on définit les potentiels U_{T_i} basés dessus de la même façon qu'on a construit U_T sur T (à la différence qu'il n'y a pas besoin de projeter sur

un espace de position/orientation). On a alors le potentiel total

$$U'_T(\mathbf{q}) = \alpha U_T(\mathbf{q}) + \sum \lambda_i U_{T_i}(\mathbf{q}) \tag{6.9}$$

où α permet de pondérer entre le potentiel global et les potentiels individuels, et les λ_i sont des paramètres d'activation dont la valeur est 1 ou 0 selon que l'on souhaite prendre en compte le i$^{\text{ème}}$ champ de potentiel individuel ou non. Typiquement, nous prenons $\lambda_i = 1$ pour les pieds et les mains, parfois pour les genoux.

Cette méthode permet aux corps sélectionnés d'anticiper, en se trouvant en avance par rapport au mouvement général. Cela permet à un contact qui aiderait à avancer mais demanderait un écart important à la trajectoire T de recevoir une bonne note.

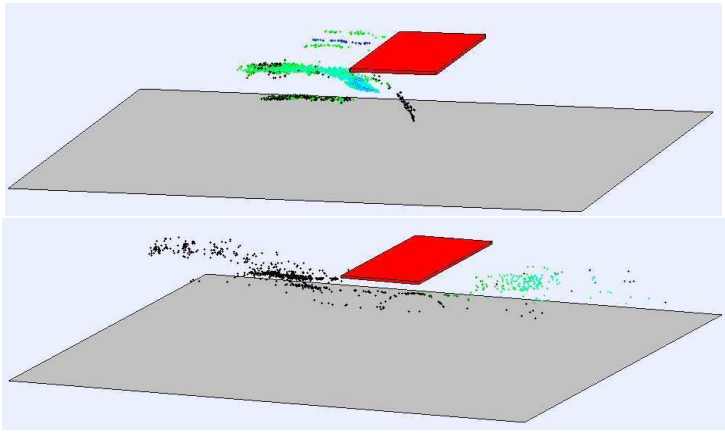


FIG. *6.6 – Position de la racine (bassin) du robot pour chaque nœud généré par le planificateur lors du passage sous une table (figurée en rouge). Le cas du haut correspond à une planification sans utilisation de trajectoires individuelles, pour celui du bas, le champ de potentiel utilise de telles trajectoires. Dans le second cas, le robot commence un peu plus à gauche par un peu de marche.*

Intéressons-nous maintenant au scénario où le robot doit passer sous une table. Dans ce scénario, le planificateur doit à la fois surmonter le problème des petits pas lors de la marche à quatre pattes et surtout celui du changement de posture qui intervient deux fois, pour passer de debout à quatre pattes puis retourner à la posture debout. Une trajectoire T est définie manuellement pour guider le robot. La figure 6.6 illustre les difficultés de planification selon que le champ de potentiel utilise ou non des trajectoires individuelles. Lorsque ces trajectoires ne sont pas utilisées, le planificateur a du mal à passer et avancer sous la table. Il est arrêté après 20000 nœuds générés. On observe un grand nombre de nœuds avant la table du fait de la difficulté à changer de posture.
Au contraire avec l'utilisation de trajectoires individuelles, le planificateur trouve un chemin après 1.3 heures et 2916 nœuds générés. Le résultat est une séquence de 141 nœuds et est montré par la figure 6.7. La moitié du temps de planification est passé avant la table, alors que passer sous la table est fait en 30 nœuds. Des essais précédents, sans potentiel individuel et où le robot commençait déjà à quatre pattes, demandaient une séquence de plus de 200 nœuds pour parcourir le même trajet sous la table.

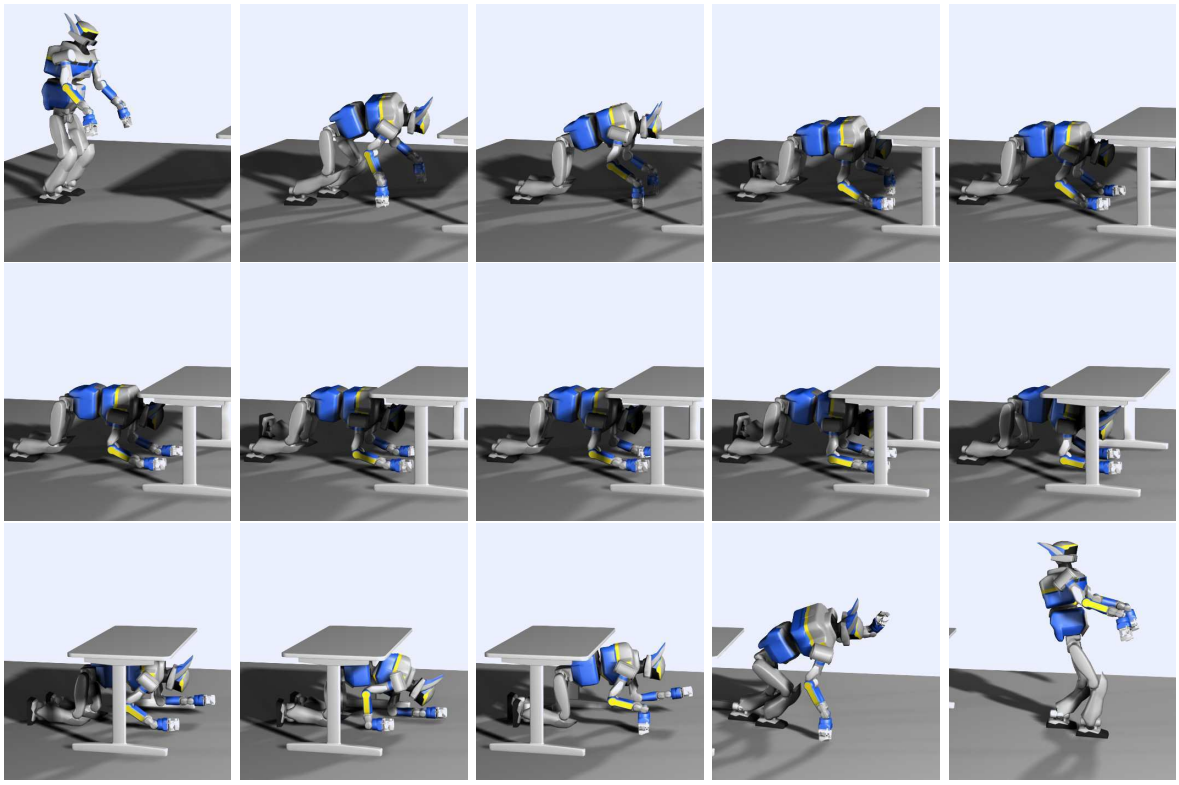


FIG. *6.7 – Passage sous la table en utilisant des potentiels individuels pour les mains, pieds et genoux. Entre chaque image il y a un "temps" de 10 nœuds.*

6.4 Construction de la trajectoire guide

Jusqu'à présent nous n'avons pas évoqué la façon d'obtenir une trajectoire guide. Celle-ci peut être donné à la main par l'utilisateur ce qui est utile pour qui veut contrôler l'allure du résultat final. Cependant on pourrait aussi vouloir construire cette trajectoire de manière automatique. C'est ce que nous présentons dans cette section, un travail conduit en commun avec Karim Bouyarmane [Bouyarmane 09].

6.4.1 Algorithme général

On cherche un chemin approché. Il n'a donc pas besoin de se trouver dans l'espace de configuration en contact $\mathcal{C}_c$. On sait qu'on peut l'autoriser à être en collision avec des $\mathcal{C}$-obstacles, mais rien n'interdit de rester dans $\mathcal{C}_{\text{free}}$. Cette autorisation peut se traduire par un test plus laxiste des collisions (par exemple en prenant des marges de sécurité négatives). On cherche cependant à ce que le chemin reste proche d'au moins certains obstacles, de façon à ce que le planificateur d'appuis puisse ensuite les utiliser. Néanmoins, le fait de chercher un chemin dans $\mathcal{C}_{\text{free}}$ permet d'utiliser les outils de la planification sans collision classique. Au rang de ces outils sont les méthodes basées sur l'échantillonnage telles que les Probabilistic Roadmaps (PRM) [Kavraki 96] ou les Rapidly exploring Random Trees (RRT) [LaValle 98]. Rappelons, pour fixer les notations, que ce type d'algorithme construit progressivement un graphe $\mathcal{G}(V, E)$ dont chaque sommet $v \in V$ représente une configuration $\mathbf{q}$ et chaque arrête

$e \in E$ est un chemin continu de $\mathcal{C}_{\text{free}}$. Cette construction s'arrête quand un chemin est trouvé dans le graphe entre $\mathbf{q}^{\text{init}}$ et $\mathbf{q}^{\text{final}}$. Une implémentation de cela est donné par l'algorithme 10

Algorithm:PRM
begin
  $\mathcal{G}(V \leftarrow \{\mathbf{q}^{\text{init}}, \mathbf{q}^{\text{final}}\}, E \leftarrow \oslash)$
  while *no path found in* $\mathcal{G}$ **do**
    shoot a random configuration $\mathbf{q}$ in $\mathcal{C}$
    if $\mathbf{q} \in \mathcal{C}_{\text{free}}$ **then**
      for *each* $\mathbf{q}_V \in V \cap$ neighbourhood$(\mathbf{q})$ **do**
        if *the direct path* $\tau_d(\mathbf{q}, \mathbf{q}_v)$ *linking* $\mathbf{q}$ *to* $\mathbf{q}_V$ *lies in* $\mathcal{C}_{\text{free}}$ **then**
          add $\mathbf{q}$ to V (if not already in V) and $\tau_d(\mathbf{q}, \mathbf{q}_v)$ to E
        end
      end
    end
  end
end

Algorithm 10: Probabilistic Roadmap.

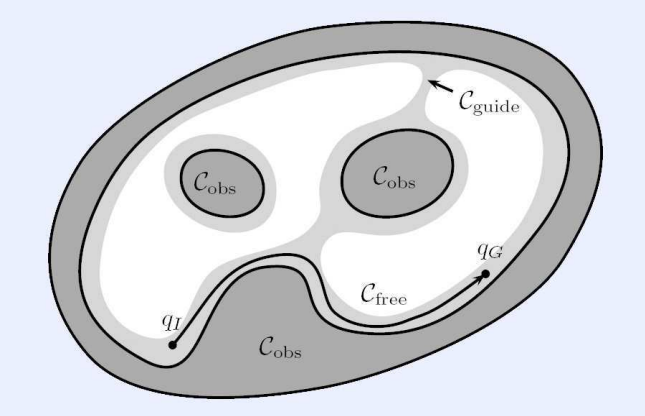


FIG. *6.8 – $\mathcal{C}_{guide}$.*

Le principe de la génération automatique de trajectoire guide va être d'adapter cet algorithme pour obtenir une trajectoire qui reste proche de certains obstacles, c'est-à-dire dans une sous-ensemble de $\mathcal{C}_{\text{free}}$ que nous nommons $\mathcal{C}_{\text{guide}}$. Cet ensemble peut être visualisé comme une couche le long de la frontière de $\mathcal{C}_{\text{free}}$ (cf Fig 6.8).

Avant de construire formellement cet ensemble, donnons quelques définitions.
On dit *pour cette section seulement* qu'un corps r_i du robot est en contact avec la région des obstacles $\mathcal{O}$ si :

$$0 < d(r_i, \mathcal{O}) < \varepsilon_{\text{contact}} \tag{6.10}$$

où d désigne la distance. C'est cette définition qui donnera son "volume" à $\mathcal{C}_{\text{guide}}$ par rapport à la frontière de $\mathcal{C}_{\text{free}}$.
On appelle A_i et $A_{\mathcal{O}}$ les points témoins de cette distance et $\mathbf{n}$ sera le vecteur unitaire normal à $\mathcal{O}$ en $A_{\mathcal{O}}$.

La définition de $\mathcal{C}_{\text{guide}}$ a non seulement besoin d'une notion de proximité aux obstacles, mais aussi d'une notion de stabilité. En considérant qu'il n'y a pas de limite de couple (c'est-à-dire que le robot peut être, à une configuration donnée, considéré comme un solide), les conditions d'équilibre statique s'écrivent

$$\begin{cases} \sum_{\mathbf{f}\in\mathcal{F}} \mathbf{f} + m\mathbf{g} = \mathbf{0} \\ \sum_{\mathbf{f}\in\mathcal{F}} \mathscr{M}_O(\mathbf{f}) + \mathscr{M}_O(m\mathbf{g}) = \mathbf{0} \end{cases} \tag{6.11}$$

où $\mathcal{F}$ est l'ensemble des forces de contact, qui s'appliquent aux corps du robot considérés en contact par la définition précédente comme si ils étaient véritablement en contact. $\mathscr{M}_O$ est le moment d'une force au point $O \in \mathbb{R}^3$, m est la masse du robot et $\mathbf{g}$ le vecteur de gravité. Par soucis de simplicité, nous modélisons toute surface de contact comme un ensemble discret de contacts ponctuels appliqué à des points choisis sur la surface. Chaque force $\mathbf{f} \in \mathcal{F}$ appliquée au robot au point $A \in \partial\mathcal{R}$ avec une normale $\mathbf{n}$ doit appartenir à un *cône de friction* $\mathscr{C}_{A,\mathbf{n},\theta}$, θ étant l'angle d'ouverture du cône qui dépend du coefficient de frottement. A est le sommet du cône et $\mathbf{n}$ en définit son axe.
On dit que le robot est statiquement stable si

$$\forall i \in I(q),\ \exists\, \mathbf{f}_i \in \mathscr{C}_{A_{\mathcal{B}_i},\mathbf{n}_i,\theta_i}, \text{tels que} \begin{cases} \sum_{i\in I(q)} \mathbf{f}_i + m\mathbf{g} = \mathbf{0} \\ \sum_{i\in I(q)} \mathscr{M}_O(\mathbf{f}_i) + \mathscr{M}_O(m\mathbf{g}) = \mathbf{0} \end{cases} \tag{6.12}$$

où

$$I(q) = \left\{ i \in \{1,\dots,m\} \mid 0 < d(\nabla_i(q), \mathcal{O}) < \varepsilon_{\text{contact}} \right\}$$

On peut alors donner la définition de $\mathcal{C}_{\text{guide}}$:

$$\mathcal{C}_{\text{guide}} = \{q \in \mathcal{C}_{\text{free}} \mid \mathcal{R} \text{ est statiquement stable à } q\}$$

En remplaçant $\mathcal{C}_{\text{free}}$ par $\mathcal{C}_{\text{guide}}$ dans l'algorithme 10, on obtient le comportement voulu. Cependant le volume de $\mathcal{C}_{\text{guide}}$ étant très petit devant celui $\mathcal{C}$, la probabilité de tirer une configuration $\mathbf{q}$ dans $\mathcal{C}_{\text{guide}}$ est très faible et donc l'algorithme sera très lent. Nous allons donc faire appel à une projection p, qui sera définie à la section 6.4.3, pour ramener une configuration de $\mathcal{C}_{\text{free}}$ dans $\mathcal{C}_{\text{guide}}$.

L'algorithme 11 montre les modifications qu'il a fallu apporter à l'algorithme 10 pour l'adapter à notre problème.

Cet algorithme rajoute une projection de $\mathcal{C}_{\text{free}}$ dans $\mathcal{C}_{\text{guide}}$ et vérifie que les arêtes de E sont dans $\mathcal{C}_{\text{guide}}$. Les prochaines sections détaillent certaines parties de cet algorithme.

6.4.2 Tirage biaisé de configurations aléatoires

La configuration d'un robot $\mathcal{R}$ se décompose en ses n degrés de liberté internes, 3 degrés de liberté de translation et 3 pour la rotation ce qui peut être résumé par

$$\mathcal{C} = \mathcal{C}_{\text{position}} \times \mathcal{C}_{\text{orientation}} \times \mathcal{C}_{\text{posture}} \tag{6.13}$$

Tirer une configuration aléatoire revient à tirer indépendamment une position, une orientation et une posture : la variable aléatoire Q de ce tirage est un vecteur de trois variables aléatoires indépendantes $Q = (Q_{\text{position}}, Q_{\text{orientation}}, Q_{\text{posture}})$.

Algorithm:PRM Guide
begin
  $\mathcal{G}(V \leftarrow \{\mathbf{q}^{\text{init}}, \mathbf{q}^{\text{final}}\}, E \leftarrow \oslash)$
  while *no path found in* $\mathcal{G}$ **do**
    shoot a random configuration $\mathbf{q}$ in $\mathcal{C}$
    if $\mathbf{q} \in \mathcal{C}_{\text{free}}$ **then**
      $\mathbf{q}' \leftarrow p(\mathbf{q})$
      for *each* $\mathbf{q}_V \in V \cap$ neighbourhood$(\mathbf{q}')$ **do**
        if *the direct path* $\tau_d(\mathbf{q}', \mathbf{q}_v)$ *linking* $\mathbf{q}'$ *to* $\mathbf{q}_V$ *lies in* $\mathcal{C}_{\text{guide}}$ **then**
          add $\mathbf{q}'$ to V (if not already in V) and $\tau_d(\mathbf{q}', \mathbf{q}_v)$ to E
        end
      end
    end
  end
end

Algorithm 11: Probabilistic Roadmap modifiée pour trouver une solution dans $\mathcal{C}_{\text{guide}}$.

6.4.2.1 Tirage de la position

Q_{position} est soit une variable aléatoire uniforme si $\mathcal{C}_{\text{position}} = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$ soit une variable gaussienne à trois dimensions si $\mathcal{C}_{\text{position}} = \mathbb{R}^3$.

6.4.2.2 Tirage de l'orientation

Pour l'orientation, nous souhaitons biaiser le tirage afin de favoriser certaines orientations qui sont plus intéressante dans le cas d'un robot humanoïde, comme une orientation verticale pour la marche, horizontale pour ramper, légèrement penché pour de l'escalade, etc.
$SO(3)$, que nous utilisons pour paramétrer l'orientation, étant homéomorphe à la sphère unitaire des quaternions $\mathbb{S}^3$, nous voulons une variable aléatoire qui soit semblable à une distribution gaussienne sur la sphère $\mathbb{S}^3$ autour d'un de ses points q_0. Une telle distribution s'appelle *distribution de Von Mises - Fisher* [Mardia 00]. Etant donné un vecteur unitaire *moyen* q_0 et un *paramètre de concentration* $\kappa \in \mathbb{R}^+$, la densité de probabilité de la distribution de Von Mises-Fisher sur la sphère $\mathbb{S}^{p-1} \subset \mathbb{R}^p$ est donnée par

$$f_{q_0,\kappa}(q) = C_p(\kappa) \exp\left(\kappa q_0^T q\right) \tag{6.14}$$

avec $C_p(\kappa)$ constante de normalisation

$$C_p(\kappa) = \frac{\kappa^{p/2-1}}{(2\pi)^{p/2} I_{p/2-1}(\kappa)} \tag{6.15}$$

où I_v est la fonction de Bessel modifiée de première espèce et d'ordre v. Le paramètre κ contrôle la concentration de la distribution autour de q_0. La distribution est d'autant plus concentrée que κ est grand. $\kappa = 0$ implique une distribution uniforme sur la sphère. Nous utilisons pour simuler la distribution de Von Mises-Fisher un algorithme donné dans [Wood 94].

6.4.2.3 Tirage de la posture

Nous pourrions simplement prendre pour Q_{posture} une variable aléatoire uniforme sur $\mathcal{C}_{\text{posture}}$, cependant les résultats produits ne sont généralement pas intéressants pour un robot humanoïde en ce sens que très peu de postures obtenues ressemblent à des postures naturelles. Pour résoudre ce problème et en même temps diminuer la dimension de l'espace de recherche, nous effectuons le tirage dans une espace affine généré par la posture debout (tous les angles à 0) du robot $\mathbf{q}_{\text{key}_0}$ et un certain nombre de *postures clés* $\mathbf{q}_{\text{key}_1}, \cdots, \mathbf{q}_{\text{key}_l}$. Ces dernières sont par exemple les postures assise, à quatre pattes, etc. Pour rester dans les limites angulaires, nous considérons l'espace

$$\mathcal{C}'_{\text{posture}} = \left\{ q_{\text{key}_0} + \sum_{i=1}^{l} \lambda_i (q_{\text{key}_i} - q_{\text{key}_0}) \mid (\lambda_i)_i \in (\mathbb{B}_k^l)^+ \right\} \tag{6.16}$$

où $(\mathbb{B}_k^l)^+$ est le cadrant positif de la boule unitaire en dimension l pour la norme k $\|.\|_k$

$$(\mathbb{B}_k^l)^+ = \left\{ (\lambda_i)_i \in [0,1]^l \mid \sum_{i=1}^{l} \lambda_i^k \leq 1 \right\} \tag{6.17}$$

que nous échantillonnons uniformément (avec $k = 1$ dans notre cas).

6.4.3 Projection sur $\mathcal{C}_{\text{guide}}$

Nous construisons la projection d'une configuration sur $\mathcal{C}_{\text{guide}}$ sur l'utilisation d'une pile de tâches. Une tâche se définit comme une fonction f de $\mathcal{C}$ dans $\mathbb{R}$ qu'on veut amener à 0, c'est-à-dire résoudre $f(\mathbf{q}) = 0$.

Supposons qu'on tire une configuration $\mathbf{q}$, pour la ramener dans $\mathcal{C}_{\text{guide}}$ il faut l'approcher de $\mathcal{C}_{\text{obs}}$ et lui faire prendre suffisamment de contact pour qu'elle puisse être stable au sens défini précédemment. On cherche cependant à déformer le moins possible la posture originellement tirée et on cherchera donc à créer aussi peu de contacts que possible. Pour créer un contact entre le corps r_i du robot et $\mathcal{O}$, il faut rapprocher r_i à une distance de $\mathcal{O}$ inférieure à $\varepsilon_{\text{contact}}$.

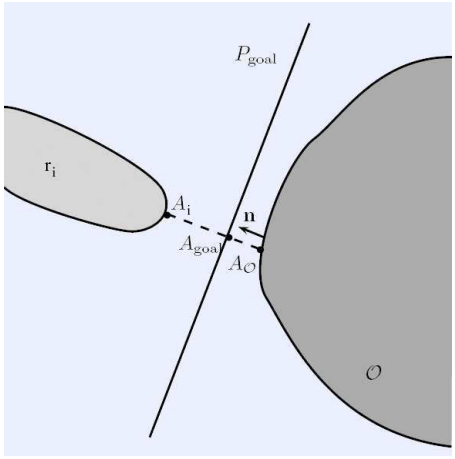


FIG. *6.9 – Définitions du point A_{goal} et du plan P_{goal}, nécessaire pour la projection.*

$A_i(\mathbf{q})$ et $A_{\mathcal{O}}(\mathbf{q})$ étant les points témoins de la distance en question et $\mathbf{n}$ le vecteur normal (cf Fig. 6.9), on définit le point A_{goal} comme le translaté de $A_{\mathcal{O}}(\mathbf{q})$ d'une distance $\varepsilon_{\text{contact}}/2$ selon $\mathbf{n}$:

$$A_{\text{goal}} = A_{\mathcal{O}} + \frac{\varepsilon_{\text{contact}}}{2}\mathbf{n} \tag{6.18}$$

et P_{goal} est le plan de normal $\mathbf{n}$ passant par A_{goal}.

Nous voulons formaliser la tâche "amener le point A_i sur le plan P_{goal}". Cela revient à amener à 0 la fonction

$$f(\mathbf{q}) = \overrightarrow{A_{\text{goal}}A_i(q)}.\mathbf{n} \tag{6.19}$$

Pour résoudre $f(\mathbf{q}) = 0$, nous avons choisi d'implémenter la méthode de Newton permettant de trouver les racines d'une fonction. On linéarise f au voisinage d'une configuration de départ $\mathbf{q}^0$

$$f(\mathbf{q}) \simeq f(\mathbf{q}_0) + J(\mathbf{q}_0).d\mathbf{q} \tag{6.20}$$

avec $d\mathbf{q} = \mathbf{q} - \mathbf{q}_0$ et $J(\mathbf{q}_0) = \frac{\partial f}{\partial \mathbf{q}}(\mathbf{q}_0)$ dont nous calculons la pseudo inverse $J(\mathbf{q}_0)^\dagger$ grâce à la pile de tâches. La solution $\mathbf{q}_s$ de l'équation $f(\mathbf{q}) = 0$ est alors

$$\mathbf{q}_s = \mathbf{q}_0 - J(\mathbf{q}_0)^\dagger f(\mathbf{q}_0) \tag{6.21}$$

La méthode de Newton consiste à construire la suite $(q_k)_{k\in\mathbb{N}}$ de manière récursive

$$\mathbf{q}_{k+1} = \mathbf{q}_k - J(\mathbf{q}_k)^\dagger f(\mathbf{q}_k) \tag{6.22}$$

Cette suite converge en principe vers une solution. Cependant nous n'avons pas besoin de trouver une solution exacte et nous nous arrêtons donc dès que le corps considéré a été amené en contact, au sens de cette section.

Nous venons de voir comment amener un corps en contact. La méthode se généralise pour amener plusieurs corps en contact en résolvant le systèmes $\{f_i(\mathbf{q}) = 0\}$ pour un certain nombre de corps d'indices i. L'utilisation d'une pile de tâche permet de donner une priorité à ces différentes tâches offrant la possibilité d'en résoudre certaines même si il y en a de contradictoires. On donne la priorité au corps qui sont déjà les plus proches des obstacles.
L'idée d'amener plusieurs corps au contact vient du fait qu'un seul corps n'est généralement pas suffisant pour assurer la stabilité. On ajoute donc un à un des corps supplémentaires à amener au contact jusqu'à obtenir une posture stable. On peut à ce moment vouloir rajouter un dernier corps pour obtenir un redondance qui pourra être utile par la suite, dans la planification d'appuis.

6.4.4 Test de stabilité

Pour tester si un robot $\mathcal{R}$ à la configuration $\mathbf{q}$ est statiquement stable ou non, il faut tout d'abord déterminer lesquels de ses corps sont en contact à cette configuration.

$$I(q) = \left\{ i \in \{1, \ldots, m\} \mid 0 < d(r_i(q), \mathcal{O}) < \varepsilon_{\text{contact}} \right\}$$

puis écrire les conditions de stabilité statique définies précédemment. Afin d'avoir un système linéaire, nous discrétisons les cônes de friction $\mathscr{C}_{A_{r_i},\mathbf{n}_i,\theta_i}$ en *cônes polyédriques* avec un nombre

fini de générateurs $\mathbf{u}_{i,1}, \ldots, \mathbf{u}_{i,n_i}$

$$\begin{aligned}\mathscr{C}_{A_{r_i},\mathbf{n}_i,\theta_i} &= \mathscr{C}(\mathbf{u}_{i,1}, \ldots, \mathbf{u}_{i,n_i}) \\ &= \left\{ \sum_{j=1}^{n_i} \lambda_j \mathbf{u}_{i,j} \ / \ \lambda_1, \ldots, \lambda_{n_i} \in \mathbb{R}^+ \right\}\end{aligned}$$

qui est l'ensemble de toutes les combinaisons linéaires non négatives des générateurs. Avec cette modélisation, nous avons

$$\mathbf{f}_i \in \mathscr{C}_{A_{r_i},\mathbf{n}_i,\theta_i} \Longleftrightarrow \exists\, (\lambda_{i,j})_{j=1..n_i} \in \left(\mathbb{R}^+\right)^{n_i},\ \mathbf{f}_i = \sum_{j=1}^{n_i} \lambda_{i,j} \mathbf{u}_{i,j} \tag{6.23}$$

Nous pouvons maintenant réécrire les conditions de stabilité statique en problème linéaire

$$\exists\, (\lambda_{i,j})_{\substack{i\in I(\mathbf{q}) \\ j=1..n_i}} \in \prod_{i\in I(\mathbf{q})} \left(\mathbb{R}^+\right)^{n_i},\ \text{tels que} \begin{cases} \sum_{\substack{i\in I(\mathbf{q}) \\ j=1..n_i}} \lambda_{i,j} \mathbf{u}_{i,j} + m\mathbf{g} = \mathbf{0} \\ \sum_{\substack{i\in I(\mathbf{q}) \\ j=1..n_i}} \mathscr{M}_O(\lambda_{i,j} \mathbf{u}_{i,j}) + \mathscr{M}_O(m\mathbf{g}) = \mathbf{0} \end{cases}$$

En posant

$$\mathbf{a}_{i,j} = \begin{pmatrix} \mathbf{u}_{i,j} \\ \mathscr{M}_O(\mathbf{u}_{i,j}) \end{pmatrix} \quad \text{and} \quad \mathbf{v} = -\begin{pmatrix} m\mathbf{g} \\ \mathscr{M}_O(m\mathbf{g}) \end{pmatrix}$$

le test de stabilité statique devient :

$$\exists\, (\lambda_{i,j})_{\substack{i\in I(\mathbf{q}) \\ j=1..n_i}} \in \prod_{i\in I(\mathbf{q})} \left(\mathbb{R}^+\right)^{n_i},\ \text{tels que} \sum_{\substack{i\in I(\mathbf{q}) \\ j=1..n_i}} \lambda_{i,j} \mathbf{a}_{i,j} = \mathbf{v}$$

qui peut se voir comme l'appartenance de $\mathbf{v}$ à un cône généré par les vecteurs $\mathbf{a}_{i,j}$

$$\mathbf{v} \in \mathscr{C}(\mathbf{a}_{i,j})_{i,j}$$

Ce problème se résout grâce à la théorie des cônes polyédriques convexes, en s'appuyant notamment sur le lemme de Farkas [Farkas 02]. On montre par cette théorie que ce problème est équivalent à

$$\forall i \in \{1, \ldots, l\}\ \mathbf{v}^T \mathbf{b}_i \leq 0$$

où les $\mathbf{b}_i$ sont les générateurs du cône dual de $\mathscr{C}(\mathbf{a}_{i,j})_{i,j}$. Ce dernier test est plus simple à faire, pourvu qu'on puisse déterminer les vecteurs $\mathbf{b}_i$. Cela se fait grâce à l'algorithme de double description de Motzkin [Motzkin 53] [Fukuda 95]. Nous avons implémenté une version modifiée de l'algorithme original proposé dans [Padberg 99].

Grâce aux méthodes présentées dans cette section, nous sommes donc en mesure de générer automatiquement une trajectoire guide qui sera ensuite utilisée pour construire un champ de potentiel à destination du planificateur d'appuis. La trajectoire trouvée dans le cadre du scénario de la table et la chaise, que l'on verra plus en détails au chapitre suivant, est trouvée en 10 min et présentée à la figure 6.10. Elle permet par la suite une planification d'appuis réussie en environ 3h30.

Il reste des améliorations à apporter à ce module. Notamment, il serait souhaitable qu'il trouve plusieurs trajectoires possibles (plutôt que de s'arrêter à la première) et puisse choisir entre elles selon des critères de confort ou de sécurité.

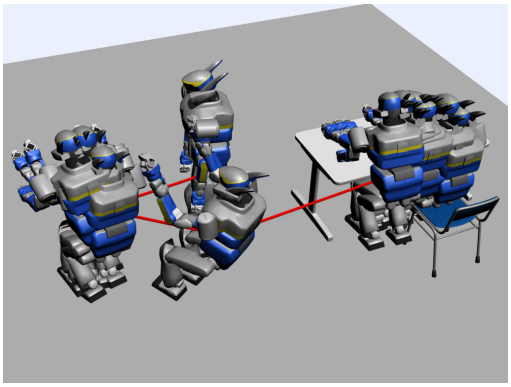


FIG. *6.10 – Trajectoire guide trouvée pour un scénario où le robot doit se lever d'une chaise placée devant une table.*

6.5 Extension : planification avec des tâches supplémentaires

Le générateur de posture est un élément très important et puissant de notre planificateur. Il permet de se restreindre à une sous-variété de l'espace de configuration d'une façon simple et relativement intuitive pour l'utilisateur qui a un contrôle sur la solution à la fois via les contraintes et via le critère.
Nous avons vu au chapitre 3 un certain nombre de possibilités du générateur de posture dont nous n'avons pas tiré parti dans la planification. En particulier, nous ne nous sommes servis des tâches $\mathcal{T}_i$ que sous forme de contacts. Cependant les changements nécessaires pour ajouter des tâches, soit pour toute la durée d'une planification soit pour une partie de celle-ci sur la bases de déclencheurs, sont minimes : il suffit de rajouter, éventuellement sous certaines conditions, la tâche $\mathcal{T}_i$ au problème $\mathcal{Q}_s$ en cours à chaque génération de posture.

Voici quelques exemple de tâches qui peuvent avoir un grand intérêt lors d'une planification :

- maintenir l'orientation d'un objet porté. Nous donnerons un exemple juste après,
- regarder sans discontinuité un objectif, ou maintenir dans son champ de vue des points de repère. Cela peut être utile pour de l'auto-localisation,
- regarder le contact qui se crée, de façon à avoir un retour visuel lorsque le plan est joué sur robot réel, ce qui peut pallier l'absence de capteurs tactiles, etc.

Prenons un exemple : le robot doit porter un verre plein entre deux points. Si il ne commence pas avec le verre dans la main, il doit d'abord l'atteindre et l'attraper. Nous avons vu une telle planification au chapitre précédent. Séparer le problème en sous-missions comme "va jusqu'au verre", "prend le verre", "porte le", ..., serait la tâche d'un autre planificateur et nous ne nous attaquerons pas à de tels problèmes ici. Supposons que le robot commence avec le verre à la main. Afin de ne pas renverser de liquide, il doit garder à tout instant le verre en position verticale, en particulier au moment des postures témoins. Cela se contraint grâce à la tâche suivante :

$$\mathcal{T}_{\text{glass}} = \begin{cases} \mathbf{n}(\mathbf{q}).\mathbf{i} = 0 \\ \mathbf{n}(\mathbf{q}).\mathbf{j} = 0 \\ -\mathbf{n}(\mathbf{q}).\mathbf{k} < 0 \end{cases} \tag{6.24}$$

où $\mathbf{n}(\mathbf{q})$ est l'axe de la main tenant le verre, et $(\mathbf{i}, \mathbf{j}, \mathbf{k})$ est le repère absolu (cf Fig. 6.11).

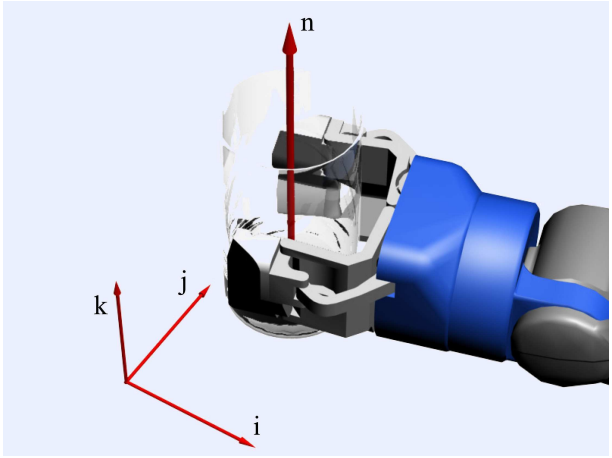


FIG. *6.11 – Vecteurs liés à la définition de la tâche* $\mathcal{T}_{glass}$

Cette tâche est rajoutée à chaque génération de posture. Le résultat d'une planification avec une telle tâche est montré au chapitre suivant.

Ajouter une tâche lors de la planification réduit le nombre de degrés de liberté dont dispose le robot pour évoluer, ce qui signifie que des postures témoins seront trouvées pour moins d'ensembles de contacts. Cela a donc un impact direct sur le plan puisque réduisant les choix du planificateur.

6.6 Conclusion

Ce chapitre a permis de décrire les versions avancées des modules les plus conséquents et importants de notre planificateur. Nous avons d'abord montré comment se faisait le choix des points de contact candidats, par l'introduction d'un espace de recherche à trois dimensions, puis comment réduire l'effort de génération et validation de ces candidats en se basant sur une prédiction partielle de leur score relatif. Nous avons ensuite décrit comment nous construisions le champ de potentiel utilisé par la planification, à partir d'une trajectoire guide, qui peut être donnée par l'utilisateur ou trouvée automatiquement par un algorithme que nous détaillons. Enfin, nous avons montré comment tirer pleinement parti des options supplémentaires qu'ouvre le choix de l'utilisation de la génération de posture au sein de la planification : l'ajout de tâches n'ayant pas de but locomoteur.
Avec les modules de ce chapitre, nous disposons d'un planificateur d'appuis évolué et avons terminé la partie planification de cette thèse. Le chapitre suivant est consacré aux résultats obtenus pour divers scénarios et aux passages de certains d'entre eux sur robot réel.

Chapitre 7

Simulations et expérimentations

Le chapitre précédent nous a fourni des outils efficaces pour réaliser des planifications d'appuis. Dans ce chapitre nous montrons des résultats que ceux-ci ont permis d'obtenir à travers trois scénarios (section 7.2 à 7.4) pour lesquels nous transposons les plans obtenus sur un robot HRP-2 réel [Kaneko 04]. Avant d'entrer dans le détail de ces scénarios, nous expliquons comment nous obtenons un chemin continu à partir de la sortie discrète de notre planificateur (section 7.1). Enfin nous concluons par des remarques et perspectives tirées de ces planifications et expérimentations (section 7.5). Tous les temps de calcul donnés dans ce chapitre ont été obtenus sur un Pentium 4 cadencé à 3.4GHz et doté de 2Go de mémoire vive.

7.1 Trajectoires entre ensembles de contacts

7.1.1 Passage d'un plan de contacts à une exécution

La sortie de notre planificateur est une séquence de couples (ensemble de contacts, posture témoin) $(\mathcal{Q}_s, q)$. On souhaite pouvoir faire exécuter ce plan par un robot. Plusieurs possibilités sont ouvertes pour cela :

- utilisation d'une pile de tâches [Mansard 07] : on traduit les problèmes de la séquence des $\mathcal{Q}_s$ en une succession de tâches. Cette traduction est aisée tant les formulations sont proches, comme évoqué au chapitre 3. Il faut cependant ajouter des tâches intermédiaires pour guider le mouvement des corps allant ou sortant d'un contact. Par exemple, il est préférable que ce mouvement se commence ou termine, selon le cas, de manière perpendiculaire à la surface de contact de l'environnement. Les grands avantages à l'utilisation de la pile de tâches sont d'une part de ne pas avoir besoin d'une trajectoire explicite, et d'autre part de fournir un contrôleur pour l'exécution du plan. En cela, cette exécution est robuste aux erreurs,
- génération de trajectoires par optimisation, soit en utilisant l'extension du générateur de posture (cf 3.5), soit en utilisant des outils prenant en compte la dynamique du ro-

bot [Miossec 06]. Cette seconde solution est celle qui offrira les trajectoires les plus rapides. Les deux cas demandent un travail de réécriture et réorganisation des contraintes de $\mathcal{Q}_s$ pour tenir compte du temps. On fait ensuite jouer la trajectoire obtenue sur le robot,

– planification entre deux ensembles de contacts consécutifs s_i et s_{i+1} du plan : on considère le problème de planification entre $\mathbf{q}^i$ et $\mathbf{q}^{i+1}$ sur une sous-variété de $\mathcal{C}$ définie par $\mathcal{Q}_{s_i}$. En mettant bout à bout les chemins obtenus pour chaque couple (s_i,s_{i+1}), on obtient un chemin total auquel il faut ajouter un profil de vitesse qui tienne compte de l'hypothèse d'équilibre quasi-statique. On joue alors la trajectoire obtenue sur le robot,
– spécification du chemin des corps entrant ou sortant d'un contact, et utilisation de la cinématique inverse. Cela permet l'obtention d'un chemin auquel on rajoute un profil de vitesse, avant passage sur le robot

Par ailleurs, les approches précédemment proposées ne sont pas exclusives et peuvent être combinées de manière à résoudre le problème efficacement.Nous avons commencé à travailler sur les deux premières solutions. Néanmoins les résultats expérimentaux présentés ici ont été obtenus avec la quatrième, que nous détaillons maintenant.

7.1.2 Chemin entre deux postures témoins

L'obtention d'un chemin complet se fait en résolvant les sous-problèmes que sont les passages entre deux postures témoins consécutives $\mathbf{q}^i$ et $\mathbf{q}^{i+1}$ attachées aux problèmes Q_{s_i} et $Q_{s_{i+1}}$. On s'attache donc à chercher un chemin de $\mathbf{q}^i$ à $\mathbf{q}^{i+1}$. Ce chemin doit se trouver dans Q_{s_i}.

Pour passer de $\mathbf{q}^i$ à $\mathbf{q}^{i+1}$, la première idée est d'aller en ligne droite dans C, mais il n'y a aucune raison que ce chemin se trouve sur Q_{s_i}. L'idée est alors de le projeter sur Q_{s_i}. Pour cela, on découpe le segment $\mathbf{q}^i\mathbf{q}^{i+1}$ en k morceaux et on projette les $k-1$ postures intermédiaires $\mathbf{p}^{i,j} = \frac{1}{k}\left((k-j)\,\mathbf{q}^i + k\mathbf{q}^{i+1}\right)$. La projection se fait en utilisant le générateur de posture et en prenant la distance aux $\mathbf{p}^{i,j}$ comme critère : on résout les $k-1$ problèmes

$$\min_{\mathbf{q}\in\mathcal{Q}_{s_i}} \left\|\mathbf{q} - \mathbf{p}^{i,j}\right\|^2 \tag{7.1}$$

pour $j \in [|1, k-1|]$. On obtient ainsi des solutions $\mathbf{q}^{i,j}$. On utilise le résultat d'un problème pour initialiser le suivant si bien que les calculs sont très rapides.

Il reste deux inconvénients à cette approche :

– théoriquement la projection peut ne pas se faire de manière continue au sens où, pour deux points successifs $\mathbf{p}^{i,j}$ et $\mathbf{p}^{i,j+1}$ qu'on peut rendre aussi proche que l'on veut en augmentant k, la distance entre leurs projetés $\mathbf{q}^{i,j}$ et $\mathbf{q}^{i,j+1}$ peut rester supérieure à une constante non nulle. Cela peut être le cas si la courbure de la sous-variété sur laquelle on projette est importante, ou si il y a un "trou" dans cette sous-variété, dû à des contraintes (cf Fig. 7.1). Dans ces deux cas, les points projetés sautent brutalement d'une région à l'autre de la sous-variété. Ce problème n'a pas été observé en pratique mais il faudrait cependant l'étudier. Une première solution pour limiter les possibilités de saut serait de contraindre la projection de $\mathbf{p}^{i,j}$ sur l'hyperplan de normale $\mathbf{q}^i\mathbf{q}^{i+1}$, passant par $\mathbf{p}^{i,j}$ et de forcer $\mathbf{q}^{i,j}$ à rester proche de $\mathbf{q}^{i,j-1}$. Cela ne donne cependant aucune garantie théorique,

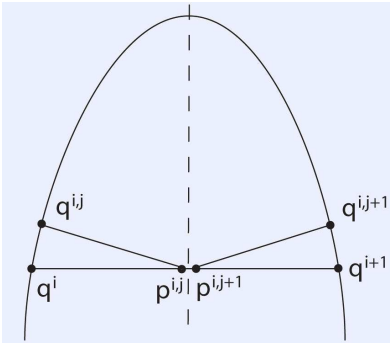


FIG. *7.1 – Les projections $\mathbf{q}^{i,j}$ et $\mathbf{q}^{i,j+1}$ de deux points $\mathbf{p}^{i,j}$ et $\mathbf{p}^{i,j+1}$ aussi proches que l'on veut ne peuvent pas toujours être rendus infiniment proches, comme par exemple dans le cas d'un espace trop courbé.*

– la trajectoire des corps qui entrent en contact, en changent ou en sortent n'est pas forcément idéale : pour une bonne exécution sur robot réel, il est souhaitable que le mouvement d'entrée ou de sortie du contact se fasse selon la normale à la surface de l'environnement, cela pour éviter de frotter.

Pour pallier le problème du second point, l'idée est de se donner des trajectoires en position et orientation avec la forme voulue pour les corps concernés, de les discrétiser et de rajouter les contraintes qu'elles impliquent aux problèmes Q_{s_i}. $\mathbf{q}^{i,j}$ est alors la solution de l'optimisation

$$\min_{\mathbf{q}\in\mathcal{Q}_{s_i}\cap\mathcal{T}_{i,j}} \left\|\mathbf{q}-\mathbf{p}^{i,j}\right\|^2 \tag{7.2}$$

où $\mathcal{T}_{i,j}$ est une tâche qui regroupe les contraintes imposées par les trajectoires. Comme celles-ci reviennent à fixer complètement la position et l'orientation d'un corps, on peut les écrire de la même façon que des contraintes de contact. On utilise alors le générateur de posture dans une optique de cinématique inverse généralisée.

Il existe cinq cas possibles pour passer de $\mathbf{q}^{i,j}$ à $\mathbf{q}^{i,j+1}$ qui dépendent du contexte dans lequel ont été obtenues ces deux configurations. Selon les changements d'espaces qu'ils impliquent, il faudra des trajectoires différentes en nombre et en types. On rappelle qu'on peut diviser un problème $\mathcal{Q}_s$ en $\mathcal{Q}_s^{\text{stab}}$, ses contraintes de stabilité, et $\mathcal{Q}_s^{\text{géom}}$, ses contraintes géométriques (voir 5.3.1.3). Un changement de $\mathcal{Q}_s^{\text{géom}}$ entre $\mathbf{q}^{i,j}$ et $\mathbf{q}^{i,j+1}$, implique qu'il y a eu géométriquement un contact en plus ou en moins et il faut alors une trajectoire pour le corps concerné.

– si $s_{i-1}\subset s_i$ et $s_i\subset s_{i+1}$ alors $\mathbf{q}^i\in\mathcal{Q}_{s_i}^{\text{géom}}\cap\mathcal{Q}_{s_{i-1}}^{\text{stab}}$ et $\mathbf{q}^{i+1}\in\mathcal{Q}_{s_{i+1}}^{\text{géom}}\cap\mathcal{Q}_{s_i}^{\text{stab}}$. Tout en restant dans $\mathcal{Q}_{s_i}$, on doit donc passer de $\mathcal{Q}_{s_i}^{\text{géom}}$ à $\mathcal{Q}_{s_{i+1}}^{\text{géom}}$ (un changement de contact au niveau géométrique), il faut pour cela une trajectoire,
– symétriquement si $s_{i-1}\supset s_i$ et $s_i\supset s_{i+1}$ alors $\mathbf{q}^i\in\mathcal{Q}_{s_{i-1}}^{\text{géom}}\cap\mathcal{Q}_{s_i}^{\text{stab}}$ et $\mathbf{q}^{i+1}\in\mathcal{Q}_{s_i}^{\text{géom}}\cap\mathcal{Q}_{s_{i+1}}^{\text{stab}}$. Tout en restant dans $\mathcal{Q}_{s_i}$, on doit alors passer de $\mathcal{Q}_{s_{i-1}}^{\text{géom}}$ à $\mathcal{Q}_{s_i}^{\text{géom}}$, il faut pour cela une trajectoire,
– si $s_{i-1}\subset s_i$ et $s_i\supset s_{i+1}$ alors $\mathbf{q}^i\in\mathcal{Q}_{s_i}^{\text{géom}}\cap\mathcal{Q}_{s_{i-1}}^{\text{stab}}$ et $\mathbf{q}^{i+1}\in\mathcal{Q}_{s_i}^{\text{géom}}\cap\mathcal{Q}_{s_{i+1}}^{\text{stab}}$. On reste alors dans $\mathcal{Q}_{s_i}$ (pas de changement de contact du point de vu géométrique),
– si $s_{i-1}\supset s_i$ et $s_i\subset s_{i+1}$ alors tout en restant dans $\mathcal{Q}_{s_i}$, on doit donc passer de $\mathcal{Q}_{s_{i-1}}^{\text{géom}}$ à $\mathcal{Q}_{s_{i+1}}^{\text{géom}}$ (2 changements de contact du point de vue géométrique). Deux cas se présentent :
  – s_{i-1} et s_{i+1} font apparaître les mêmes corps en contact : le corps en contact qui a disparu entre s_{i-1} et s_i revient dans s_{i+1}. Il y a alors besoin d'une seule trajectoire,

– s_{i-1} et s_{i+1} ne font pas apparaître les mêmes corps en contact, il faut deux trajectoires.

Pour un corps, nous construisons la trajectoires comme suit : si le corps sort d'un contact, nous commençons par un segment perpendiculaire à la surface. La longueur de ce segment est typiquement 2cm pour le robot HRP-2. Si le corps entre en contact (ce qui n'est pas exclusif avec la sortie d'un autre contact), nous finissons par un segment du même type. Le reste de la trajectoire est une spline qui se raccorde de manière C^1 à l'extrémité des segments évoqués ci-dessus quand ils existent.
Le long d'un segment, l'orientation reste la même, le long de la spline, elle est interpolée en passant par les quaternions et en utilisant l'interpolation "slerp" (spherical linear interpolation) qui donne une rotation à vitesse constante autour d'un axe fixe.

Il se peut que cette trajectoire générée automatiquement force le robot à être en collision ou dans une posture instable. On corrige alors manuellement.
En pratique dans nos scénarios, nous n'avons utilisé de telles trajectoires que pour les mains et les pieds. L'usage de trajectoires pour d'autres corps peut entraîner des problèmes surcontraints qui n'ont donc pas de solution. C'est le cas lorsque deux corps d'un même membre sont contraints sur le robot HRP-2.

Une fois les trajectoires nécessaires obtenues, on peut écrire et résoudre les problèmes (7.2) et on a alors un chemin affine par morceaux défini par $\left(\mathbf{q}^i, \mathbf{q}^{i,1}, \cdots, \mathbf{q}^{i,k-1}, \mathbf{q}^{i+1}\right)$. Ce chemin est d'autant plus proche de $\mathcal{Q}_{s_i}$ que k est grand (en admettant qu'on ne tombe pas dans les problèmes de discontinuité). Par recollement, on obtient le chemin complet

$$\left(\mathbf{q}^0, \mathbf{q}^{0,1}, \cdots, \mathbf{q}^{0,k_0-1}, \mathbf{q}^1, \mathbf{q}^{1,1}, \cdots, \mathbf{q}^{1,k_1-1}, \mathbf{q}^2, \cdots, \mathbf{q}^{l-1}, \mathbf{q}^{l-1,1}, \cdots, \mathbf{q}^{l-1,k_{l-1}-1}, \mathbf{q}^l\right)$$

Il reste à y appliquer un profil de vitesse. Celui-ci doit répondre à plusieurs critères :
– la vitesse angulaire maximale ne doit pas dépasser un seuil, pour des raisons de sécurité,
– la vitesse relative opérationnelle doit être nulle lors de la création ou de la suppression d'un contact,
– la vitesse générale du robot doit rester suffisamment basse pour ne pas induire trop d'effet dynamique, étant donné que le chemin a été planifié en quasi-statique,
– elle doit cependant être suffisamment élevée pour éviter de rester trop longtemps dans des configurations très exigeantes pour le robot, où l'intensité des courants délivrés aux moteurs est importante.

Il serait aussi souhaitable que la vitesse soit continue, néanmoins cela n'est pas possible avec un chemin linéaire par morceaux, sauf à annuler la vitesse à chaque $\mathbf{q}^{i,j}$ ce qui rendrait le mouvement extrêmement lent. Les moteurs cependant filtrent les discontinuités de vitesse, qui peuvent être rendues petites en prenant des k_i grands.

On applique un profil de vitesse sinusoïdale à chaque chemin $\left(\mathbf{q}^i, \mathbf{q}^{i,1}, \cdots, \mathbf{q}^{i,k-1}, \mathbf{q}^{i+1}\right)$: soit L_i la longueur du chemin de $\mathbf{q}^i$ à $\mathbf{q}^{i+1}$, δt le temps d'une boucle de contrôle du robot et T_i la durée totale choisie pour ce mouvement qui doit être un multiple de δt. On définit l'abscisse curviligne le long du chemin :

$$a_i(t) = \frac{L_i}{2}\left(1 - \cos\left(\frac{\pi t}{T_i}\right)\right) \tag{7.3}$$

et pour l'exécution de ce chemin, on a $\mathbf{q}(t)$ qui est le point d'abscisse curviligne $a_i(t)$. La figure 7.2 montre les trajectoires et vitesses angulaires obtenues avec la méthode décrite sur les

25 premières secondes du scénario de la section 7.3 pour des temps T_i choisi automatiquement de façon à ce que la vitesse ne dépasse pas 10deg/s (Ces T_i seront cependant changés pour l'exécution présentée après).

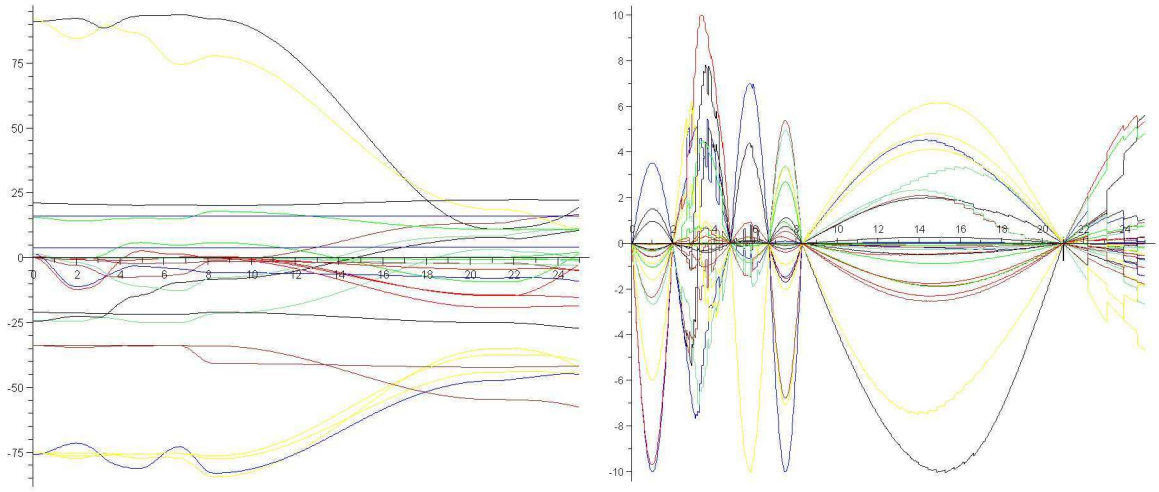


FIG. 7.2 – *Exemple de trajectoires (à gauche) et vitesses (à droite) obtenues.*

7.2 Scénario 1 : attraper une canette sur une table

7.2.1 Scénario

Le premier scénario a déjà été présenté au chapitre 5. Il s'agit pour le robot d'attraper une canette posée sur une table, hors d'atteinte depuis sa posture initiale à un bord de la table. La planification de ce scénario va jusqu'au moment où la canette est atteinte, en passant par un appui de la main sur la table, puis de la jambe contre la table. La canette est alors atteignable. Une fois le chemin obtenu, celui-ci a été renversé, avec la main fermée sur la canette, pour revenir à la position de départ.

7.2.2 Expérimentation

Par rapport à son modèle numérique utilisé pour la planification, l'environnement doit subir quelques modifications. En premier lieu la table est supposée être inamovible lors de la planification. Ce n'est pas le cas, et le robot s'appuyant fortement dessus lorsqu'il se penche pour prendre la canette, il la fait bouger. Pour empêcher cela, nous avons donc dû adosser la table à un des piliers de la salle expérimentale. Parce que l'estrade sur laquelle se trouve le robot n'est aussi pas véritablement fixe, nous devons de même ajouter une matière adhésive pour l'empêcher de glisser sur le sol.

Le second point est le rajout d'une nappe compliante sur la table pour absorber les différences entre le plan et son exécution.

La figure 7.3 montre la réalisation de ce plan. Le mouvement total (aller et retour) est effectué en 45 secondes.

FIG. 7.3 – *Étapes du scénario de la canette joué par le robot HRP-2.*

7.3 Scénario 2 : se lever d'une chaise devant une table

7.3.1 Scénario

Le second scénario que nous présentons est le suivant : HRP-2 est tout d'abord assis à une table (cf Fig. 7.4), sur une chaise. On lui demande de se lever et de s'éloigner de la table (sans bouger la chaise) [Escande 08c]. Pour cela, le scénario est spécifié comme suit :

- l'ensemble de contacts initial s_{init} est constitué de quatre contacts, les deux pieds au sol et les deux cuisses sur la chaise.
- la tâche de condition finale $\mathcal{T}_{\text{final}}$ est donnée sous la forme d'une coordonnée que le bassin doit dépasser : $y_{\text{bassin}} > 2\text{m}$, x étant la direction pointant face à la chaise et y celle pointant vers la gauche.

Douze emplacements de contacts sont prédéfinis sur le robot : les pieds, les genoux, les cuisses, le dessus des mains, un côté des mains et les avant-bras. Nous utilisons trois surfaces

FIG. 7.4 – *Le scénario "assis à une table".*

de l'environnement : le sol, le siège de la chaise et la table. Nous interdisons aux mains d'être en contact avec le sol et aux pieds de l'être avec la table et la chaise.

La difficulté principale de ce scénario est l'espace restreint dans lequel le robot doit évoluer pour sortir de la chaise (cf Fig. 7.4, droite). En particulier, il ne peut pas se tenir pleinement debout entre la chaise et la table.

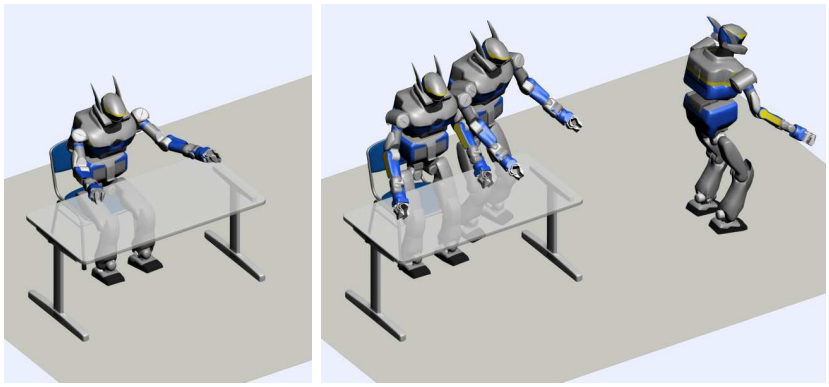


FIG. 7.5 – *Les postures clés définissant la trajectoire globale.*

La figure 7.5 montre les quatre postures clés qui spécifient la trajectoire globale. Elles ont été définies a priori. Au moment de cette expérience, nous ne disposions pas du générateur de trajectoires globales présenté en 6.4. Dans ce scénario, la trajectoire ne sert qu'à la génération d'un potentiel global, il n'y a pas de potentiel individuel pour certains corps.

7.3.2 Planification

Nous avons effectué une première planification qui était résolue en environ 3h30. Cependant, pour des raisons que nous expliquerons plus tard, ce plan ne s'exécutait pas correctement sur le robot, et nous avons dû re-planifier en augmentant grandement les marges de stabilité au niveau des pieds (cela est revenu à considérer les pieds comme des disques de 2cm de diamètre), rendant la planification plus difficile.

Le plan que nous présentons ici est le résultat de 5.2 heures de calculs. Il consiste en une séquence de 81 ensembles de contacts et postures associées. L'arbre final comporte 2411

nœuds ou feuilles.

La séquence se découpe en trois parties :

- un ajustement de la place des pieds avant de se lever, puis des appuis avec la main gauche sur la table pour déplacer le pied gauche à l'extérieur de la chaise. Cette partie est faite en 19 nœuds,
- des petits déplacements des pieds, très proches (souvent à distance de sécurité, soit $1cm$) du pied avant de la chaise, et une aide par deux fois de la main droite sur la chaise pour aider à sortir le pieds droit de la chaise. Cette phase est faite en 22 nœuds,
- une marche quasi-statique.

Ces trois parties se partagent équitablement le temps de planification. Cela peut paraître étonnant pour la troisième partie, mais beaucoup de temps est perdu par le planificateur à créer ou essayer de créer des postures avec des contacts sur la table, quand bien même celle-ci se trouve derrière le robot.

La figure 7.7 montre de plus près l'enchaînement des contacts sur les deux premières parties du mouvement en donnant les 24 premières postures témoins d'indice pair. On y retrouve les différentes étapes évoquées ci-dessus.

7.3.3 Expérimentation

L'obtention des trajectoires est faite grâce à la méthode exposée en 7.1. Les trajectoires des corps sont représentées figure 7.8. Il a fallu retoucher à la main celle où la main droite passe d'au-dessus de la table à la chaise et quelques trajectoires des pieds lors de la marche finale qui forçaient ceux-ci à entrer en collision. Une fois ces ajustements faits, l'ensemble des générations de postures sur les trajectoires discrétisées prend environ 90 secondes, pour des discrétisations avec $k_i = 20$ (soit 1539 générations).

La plus grosse difficulté rencontrée pour faire jouer cette planification sur HRP-2 a été due à la flexibilité passive que celui-ci a au niveau des chevilles. Si cette flexibilité est un avantage pour la marche dynamique, elle s'est révélée être un grande source d'instabilité sur des mouvements acycliques quasi-statiques. Il a donc fallu re-planifier de façon à ce que la projection du CdM se trouve le plus près possible du centre de cette flexibilité lors des appuis sur un seul pied, et proche du segment issu du centre des flexibilités lorsque les deux pieds servaient d'appuis. Cela néanmoins n'a pas suffit à assurer la stabilité et il a fallu utiliser le stabilisateur proposé avec HRP-2. Ce stabilisateur est un contrôleur en boucle fermée qui agit sur la posture de HRP-2 de manière à compenser les perturbations de nature diverses (imprécisions des modèles, flexibilités, forces extérieures appliquées, etc.) et satisfaire la stabilité de la posture au plus près de ce qui a été souhaité. C'est en réalité la technologie clé de HRP-2. L'utilisation de ce stabilisateur requiert deux points :

- qu'on fournisse une trajectoire de référence du Zero Moment Point (ZMP [Vukobratović 04]) dans le repère lié au bassin. Puisque le mouvement est quasi-statique, nous donnons comme ZMP la projection du centre de masse sur le sol, exprimée dans le repère du bassin. Il faut en plus fournir l'orientation du bassin,
- que les genoux soient suffisamment pliés. En effet, le stabilisateur se comporte mal lorsque les jambes sont droites, et donc dans une position singulière qui gêne la cinématique inverse utilisée. Nous avons donc dû générer à nouveau toutes les postures en changeant les bornes sur les angles des genoux.

Le stabilisateur, cependant, a été écrit pour la marche, pour des contacts coplanaires. Il

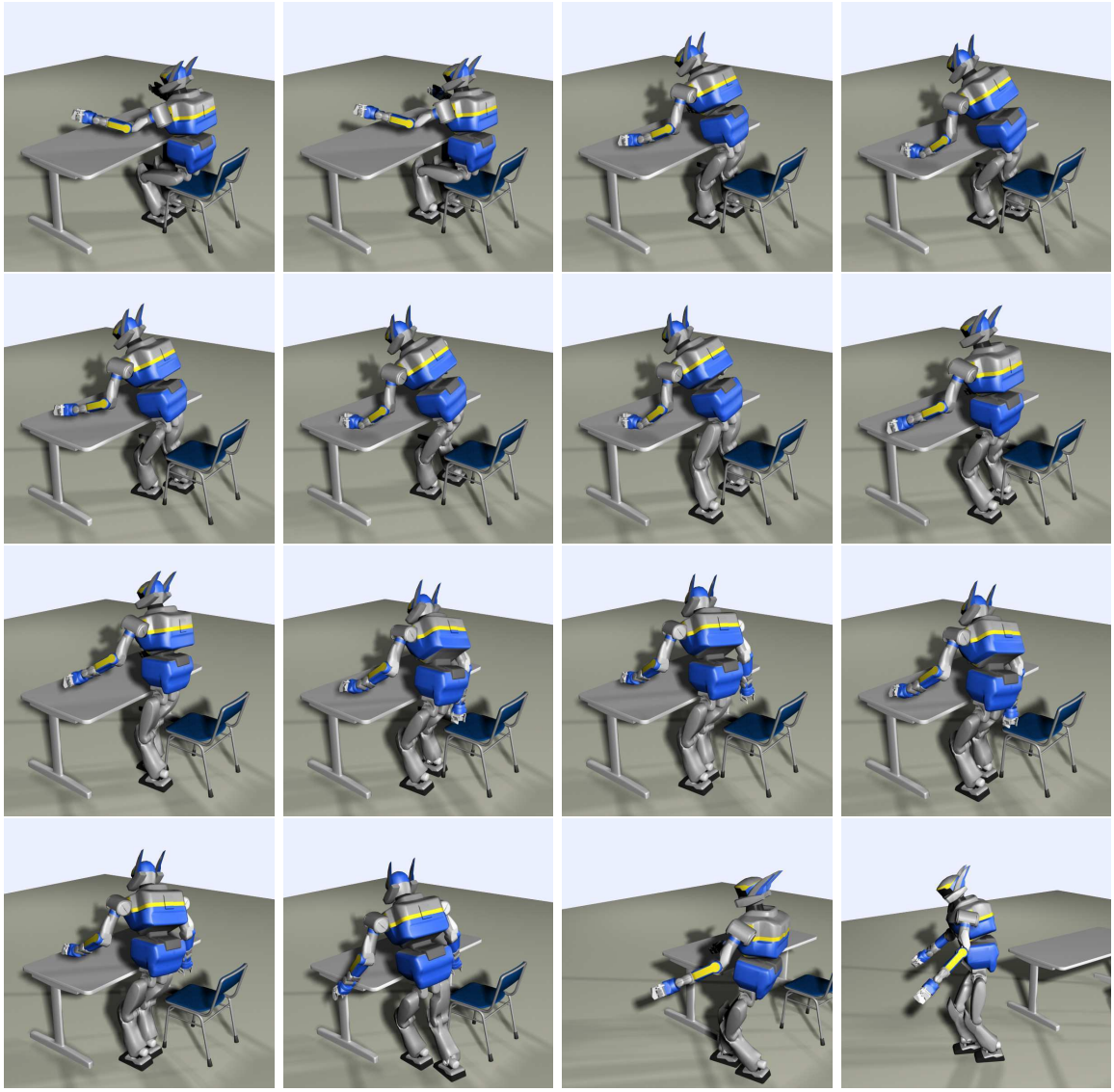


FIG. 7.6 – *Quelques postures témoins de la planification du scénario "assis à une table". Le robot est assis sur une chaise et bouge d'abord légèrement ses pieds avant de se lever, puis de poser sa main gauche sur la table (première ligne, postures 0, 2, 5 et 7). Il décale ensuite sa main en contact, ce qui l'aide à passer son pied gauche à l'extérieur de la chaise (deuxième ligne, postures 16, 17, 20, 23). Il utilise ensuite par deux fois sa main droite pour s'aider à sortir son pied droit (troisième ligne, postures 29, 31, 36, 38). Il peut alors s'éloigner en marchant (quatrième ligne, postures 42, 46, 60, 73).*

faut donc le désactiver lorsque d'autres contacts sont pris, comme c'est le cas lorsqu'une main s'appuie sur la table ou la chaise, et le remettre en route ensuite. La désactivation et la réactivation du stabilisateur posent des problèmes délicats. Lorsque le stabilisateur est désactivé, la posture s'asservit abruptement et exclusivement à la posture désirée. A l'inverse lors d'une activation, la posture d'asservissement du robot est changée par le stabilisateur. Il faut donc faire attention à ce que le ZMP de référence donné ne soit pas trop loin du ZMP mesuré, sous peine d'avoir une correction forte et très rapide de la posture. On éteint et

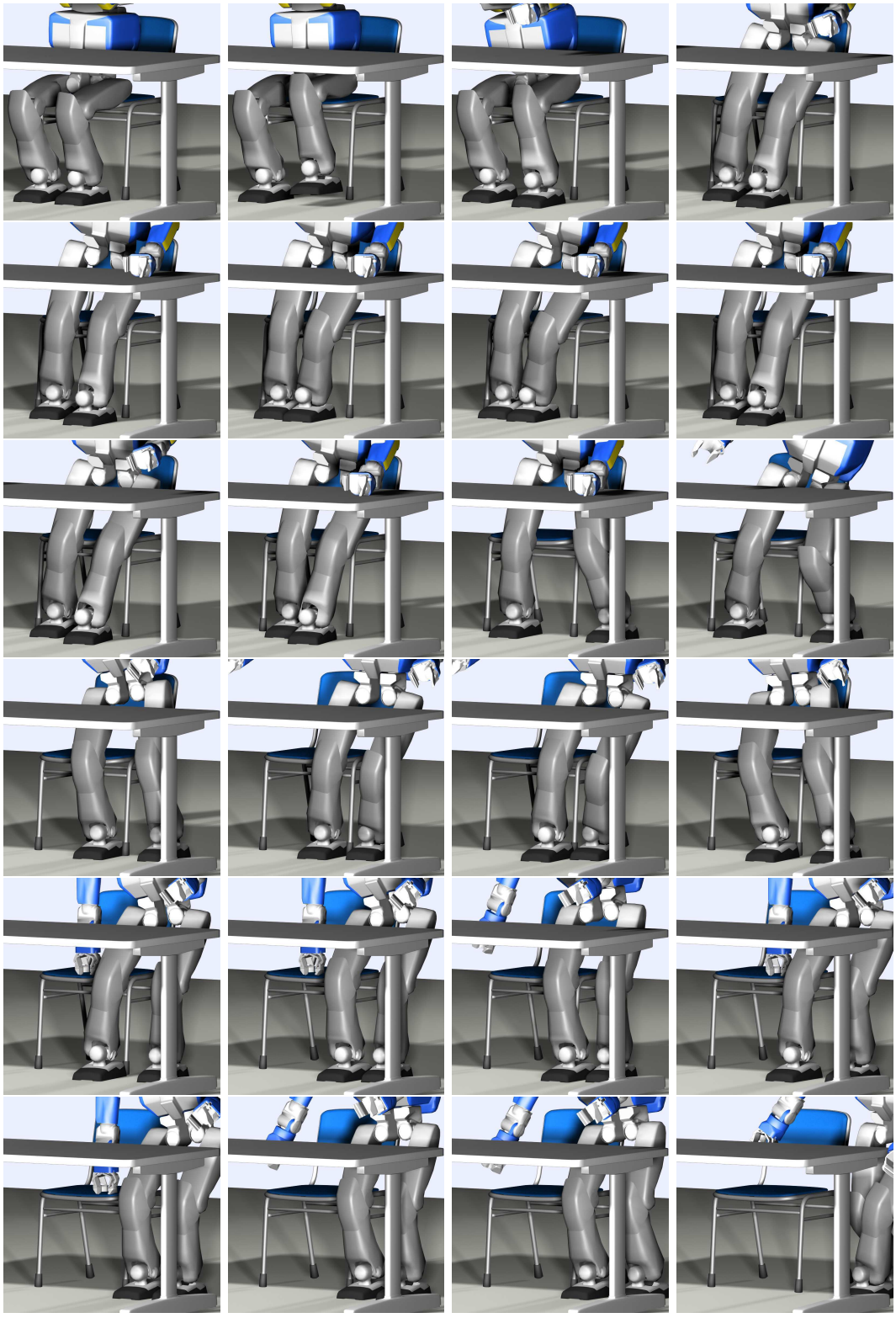


FIG. 7.7 – *Les posture* $\mathbf{q}^0$ *à* $\mathbf{q}^{46}$*, une sur deux.*

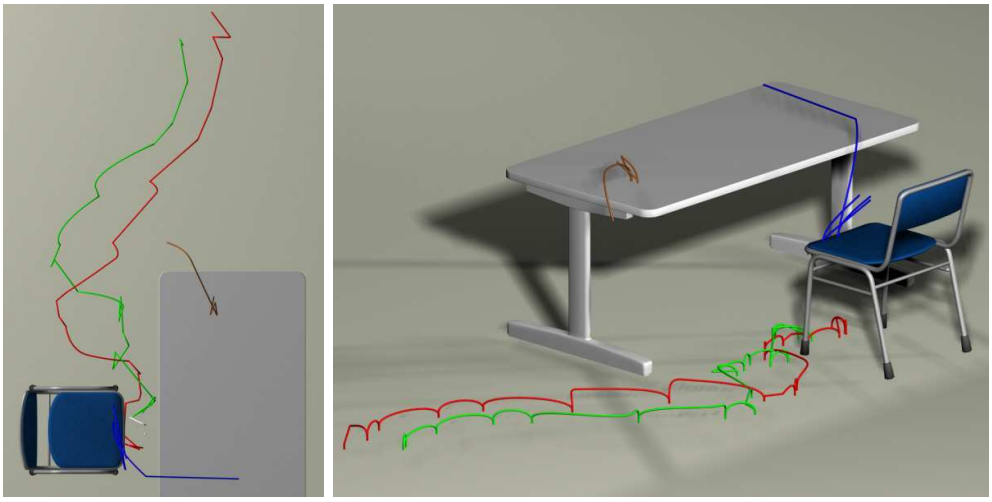


FIG. *7.8 – Les trajectoires des corps entrant, sortant ou changeant de contact. Les splines de la main droite sont en bleu, celles de la main gauche en marron, rouge pour le pied droit et vert pour le pied gauche.*

rallume donc le stabilisateur entre deux mouvements, au niveau d'une posture où le contact autre que les pieds n'est plus sensé être pris en compte dans la stabilité. Résoudre ce problème d'activation/désactivation nécessite la réécriture du stabilisateur en multi-contacts.

La durée des différentes phases du mouvement a fait l'objet d'ajustements manuels importants qu'il s'agira d'automatiser. Certaines phases sont critiques, lorsque le robot ne se soutient qu'avec une jambe et doit faire des mouvements amples. Il faut alors trouver un bon compromis entre une exécution rapide évitant la surchauffe des moteurs et une dynamique du mouvement pas trop importante pour que le stabilisateur puisse corriger. On rajoute une temporisation plus ou moins longue (de 0 à 1 seconde) entre chaque phase, pour permettre de mieux stabiliser. Et finalement le plan entier est prêt à être joué.

Compte tenu des marges prises dans la planification, la position de la table par rapport au robot n'est pas cruciale. En revanche sa hauteur (63.5cm) est importante pour que le robot s'appuie correctement dessus. De même, le robot doit être placé de manière précise par rapport à la chaise, car il y a une suite de mouvements des pieds où un pied de la chaise est très proche du robot.
Cette expérience est complexe de par sa durée (80 mouvements et 8 minutes), son ampleur et le fait que le robot soit coincé entre la table et la chaise au démarrage. Cela interdit entre autre l'utilisation des porteurs habituels (qu'on devine dans l'exemple de la canette) pour assurer le robot. On utilise alors un porteur industriel de déplacement qui se déplace au plafond de l'espace expérimental et dont l'utilisation n'est pas aisée. De plus, la restriction de la projection du centre de gravité à de petites surfaces entraîne des mouvements plus amples du robot, induisant un suivi délicat pour le système d'assurage.

Bien que complexe, l'expérience a une excellente répétabilité. Nous l'avons exécutée trois fois de suite une fois qu'elle a été mise en place, et l'avons rejouée plus tard, en devant replacer tout l'environnement.
Des photos extraites de la vidéo de cette expérience sont montrées par la figure 7.9

La figure 7.10 montre une comparaison entre les valeurs de référence données pour le ZMP et l'orientation du bassin et celles mesurées lors de l'expérience. On peut voir que la trajectoire

FIG. 7.9 – *Scénario "assis à une table" sur robot réel. Chaque ligne reprend une ligne de la figure 7.6.*

ZMP est bien suivie par le stabilisateur. On observe seulement des écarts importants en y lorsque le robot se redresse.

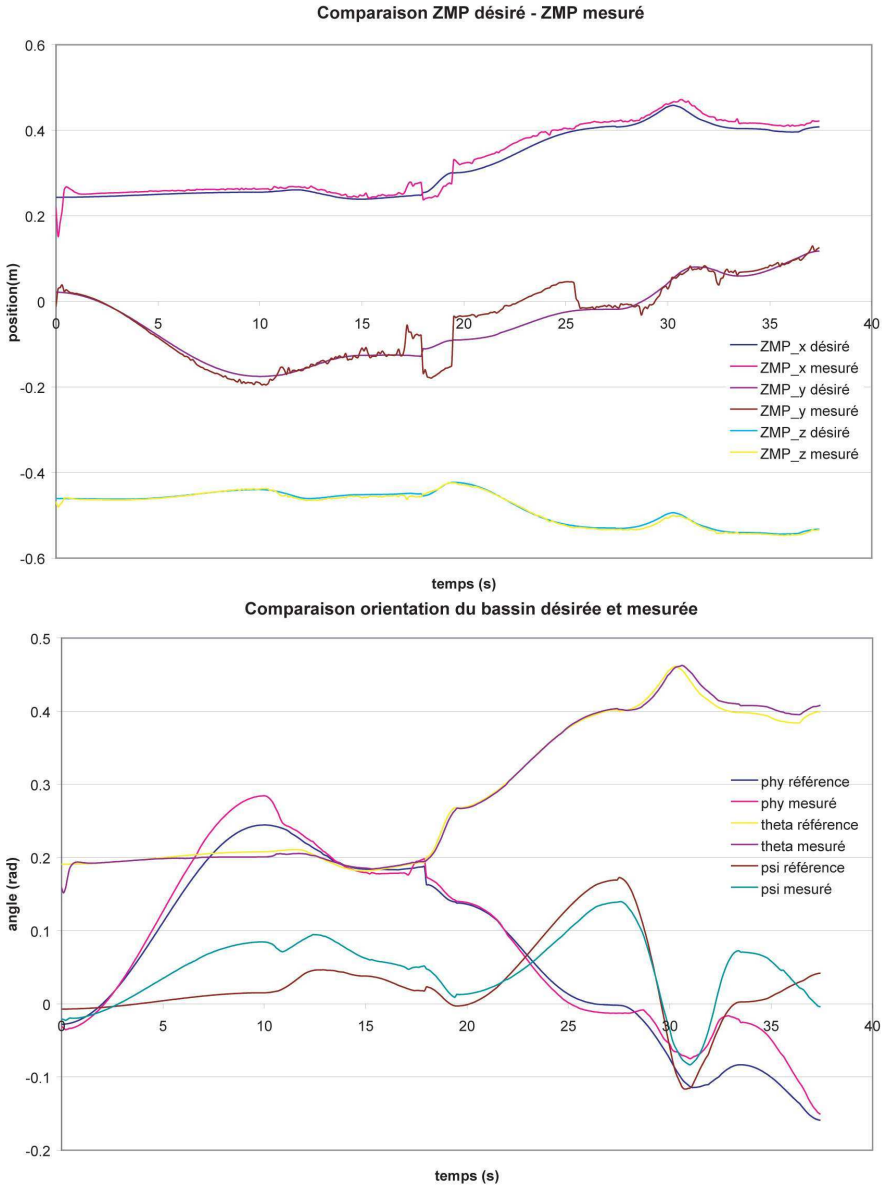


FIG. *7.10 – Comparaisons entre les valeurs désirées et mesurées pour la position du ZMP dans le repère du bassin et entre les orientations désirée et mesurée du bassin, pour les sept premières étapes du mouvement.*

7.4 Scénario 3 : s'asseoir à une table avec un verre à la main

7.4.1 Scénario

Pour ce troisième scénario, on reprend un environnement semblable au précédent, mais la difficulté est accrue de deux façons :

– on rajoute une tâche comme discuté à la section 6.5,
– le robot doit *rentrer* dans un espace étroit, et non pas s'en sortir.

Le but pour le robot est d'aller s'asseoir à une table, avec un verre plein dans la main droite. Le robot est initialement debout, en appuis sur ses deux pieds derrière la chaise et décalé (de 40cm) sur le côté. La tâche de fin s'écrit comme l'appartenance d'un point de chaque cuisse chacun à une zone peu épaisse de l'espace centré sur la surface de la chaise. Nous gardons les mêmes emplacements de contact que dans le scénario précédent, mis à part ceux faisant intervenir la main droite. Les surfaces sont les mêmes ainsi que les couples (corps, surface) interdits.

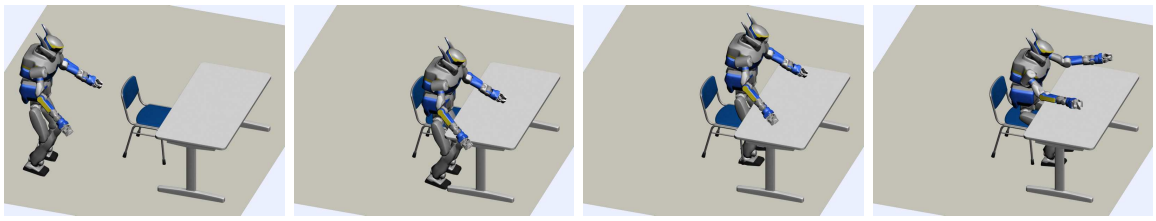


FIG. *7.11 – Postures clés pour le scénario du verre.*

La trajectoire globale est donnée par les postures clés montrées par la figure 7.11.

7.4.2 Planification

Pour ce scénario, nous avons effectué deux planifications, avec et sans l'utilisation de champs individuels de potentiel pour les mains et les pieds.

Sans utilisation des champs individuels de potentiel, la planification prend environ 4 heures et génère un arbre de 5400 nœuds. Avec utilisation des champs de potentiel, on tombe à 3 heures pour 3800 nœuds.
Dans ce second cas, la sortie est une séquence de 69 nœuds dont certains sont montrés par la figure 7.12.

Le robot commence par marcher, puis en s'aidant de sa main gauche sur la chaise, réussit après des tâtonnements à placer un pied en face de la chaise (au bout d'environ 50 nœuds). Il tient alors son point d'entrée dans l'espace étroit qui fait la difficulté de cette planification. Il peut alors commencer à s'asseoir sur la chaise et bouger dessus en s'aidant de ses cuisses.

7.4.3 Expérimentation

Afin de ne pas avoir une expérimentation aussi longue que la précédente, nous éliminons manuellement quelques contacts trop proches d'autres et re-générons les postures nécessaires aux raccords. De plus, nous nous arrêtons à la première fois où le robot s'assied sur la chaise. On se réduit ainsi à un plan d'une trentaine de nœuds.

Le schéma de l'expérience précédente ayant été bien éprouvé, nous sommes repassés par les mêmes outils, et l'expérience a pu être montée rapidement (en quelques heures, le temps de régler les vitesses par la simulation et de vérifier qu'il n'y a pas de problème). Seules certaines splines mettant en jeu la main gauche ont eu besoin d'être retouchées, ainsi que la spline où le pied gauche passe devant la chaise pour la première fois. La seule différence se situe dans

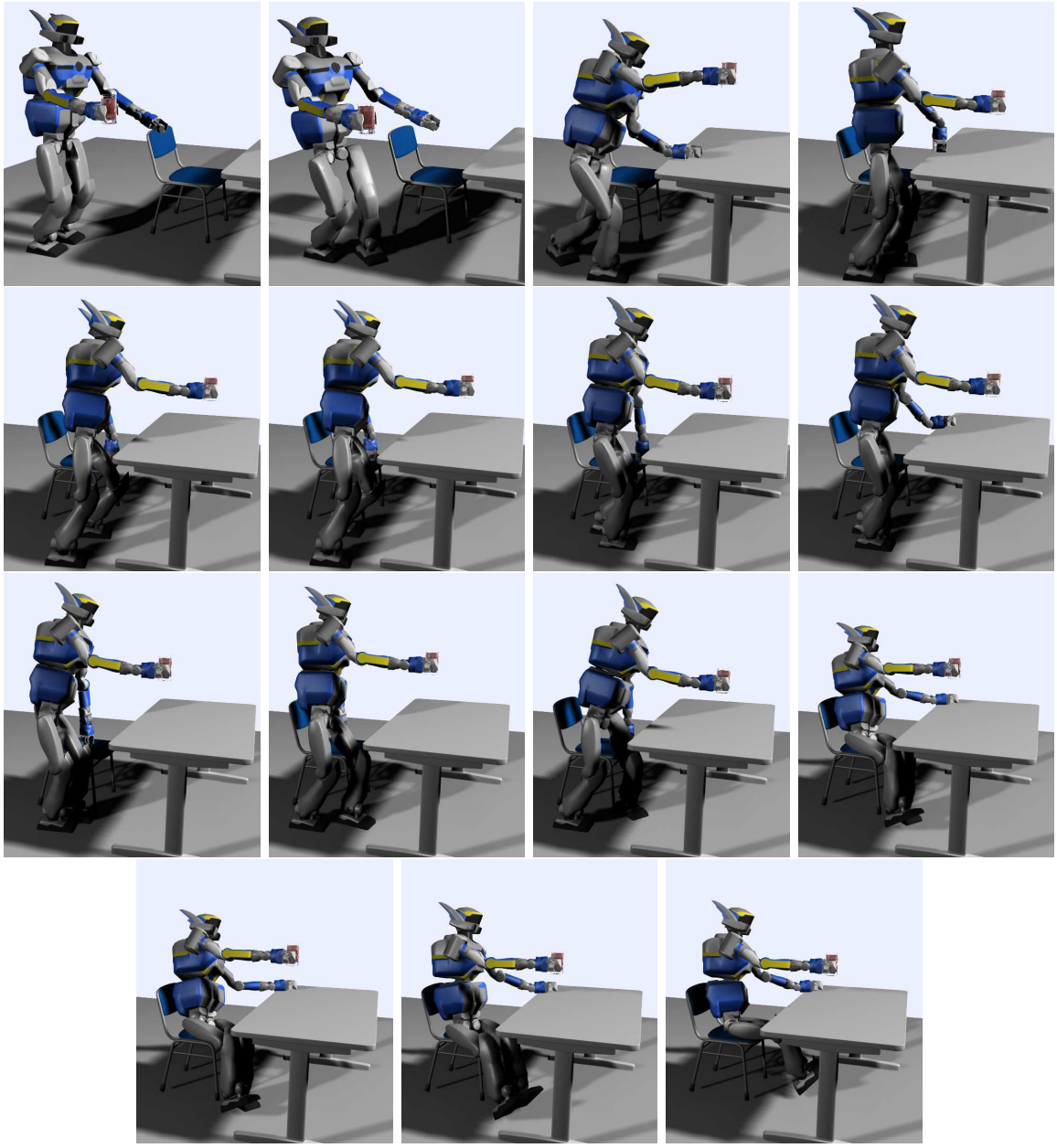


FIG. *7.12 – Les 69 nœuds du plan trouvé, donnés tous les cinq nœuds.*

le fait qu'il faut rajouter la tâche supplémentaire de tenue du verre dans la génération des postures intermédiaires pour la détermination du chemin.

Le robot demande à être très précisément placé vis-à-vis de la chaise. La table en revanche n'intervenant qu'au niveau des collisions, peut être placée de manière plus grossière. L'expérience a été réalisée une première fois à vide, puis avec un verre effectivement rempli de thé japonais. Le déplacement quasi-statique permet au liquide de ne bouger que très peu ; l'expérience peut être menée avec un verre bien plein sans problème. Des photos de celle-ci sont montrées par les figures 7.13 et 7.14. Pour la deuxième figure, le dessus de la table a été

enlevé afin de mieux voir les contacts autour de la chaise.

FIG. *7.13 – Le remplissage du verre et images clés.*

7.5 Données générales et discussion

7.5.1 Planification

Le premier point soulevé par les différents exemples de planifications donnés dans ce chapitre (ou cette thèse) est le petit nombre de nœud générés lors d'une résolution : quelques milliers dans des scénarios complexes avec un environnement large. Même en comptant les de nœuds qu'on a tenté de générer (environ 3 fois plus, voir section 7.5.2), on a un nombre de l'ordre de la dizaine de milliers. Notre planificateur a donc une approche différente des algorithmes basés sur des tirages aléatoires (la seule planification de la trajectoire guide demande un nombre similaire de nœuds pour un problème de planification bien moins contraint). Il passe beaucoup de temps pour créer des nœuds de qualité (on approche d'une moyenne de 3 secondes par nœud), mais en crée peu et de manière très peu dispersée, quand les autres algorithmes cherchent à avoir une bonne couverture de l'espace par un grand nombre de nœuds générés très rapidement. En ce sens notre algorithme construit un arbre beaucoup plus guidé, d'une part par la trajectoire globale, mais surtout par l'utilisation du générateur de posture qui, outre la résolution d'un ensemble de contraintes, permet de lisser la posture obtenue par l'optimisation.

Un second point de notre planificateur est d'avoir des temps de calcul qui ne soient pas influencés par la taille physique de l'espace. Le volume de l'espace agit bien souvent comme un facteur multiplicateur dans le temps de calcul. Ici, que le volume soit de quelques mètres cubes, ou de plusieurs kilomètres cubes, le comportement du planificateur est le même. Nous n'avons donc pas mentionné la taille dans les nombres relatifs à nos exemples.

Un mot enfin sur la séquence de sortie. Celle-ci est en général assez longue et non optimale : le nombre de mouvements est important, du fait de contacts parfois trop proches, ou de la recherche du bon emplacement pour pouvoir passer un endroit difficile. Un travail à faire est donc de filtrer la sortie pour diminuer la longueur de la séquence. Si nous l'avons déjà fait manuellement, il faut cependant se pencher sur une méthode automatique. Ce travail de filtrage peut aller de pair avec un travail de lissage pour essayer d'avoir une séquence plus naturelle. De tels problèmes ont déjà été rencontrés et traités dans le cadre de la marche [Choi 03], où les empreintes des pieds sont entre autres ajustées. L'adaptation d'un tel travail à notre cas n'est pas trivial.
Pour le filtrage et le lissage, la génération de posture pourrait encore une fois être un outil

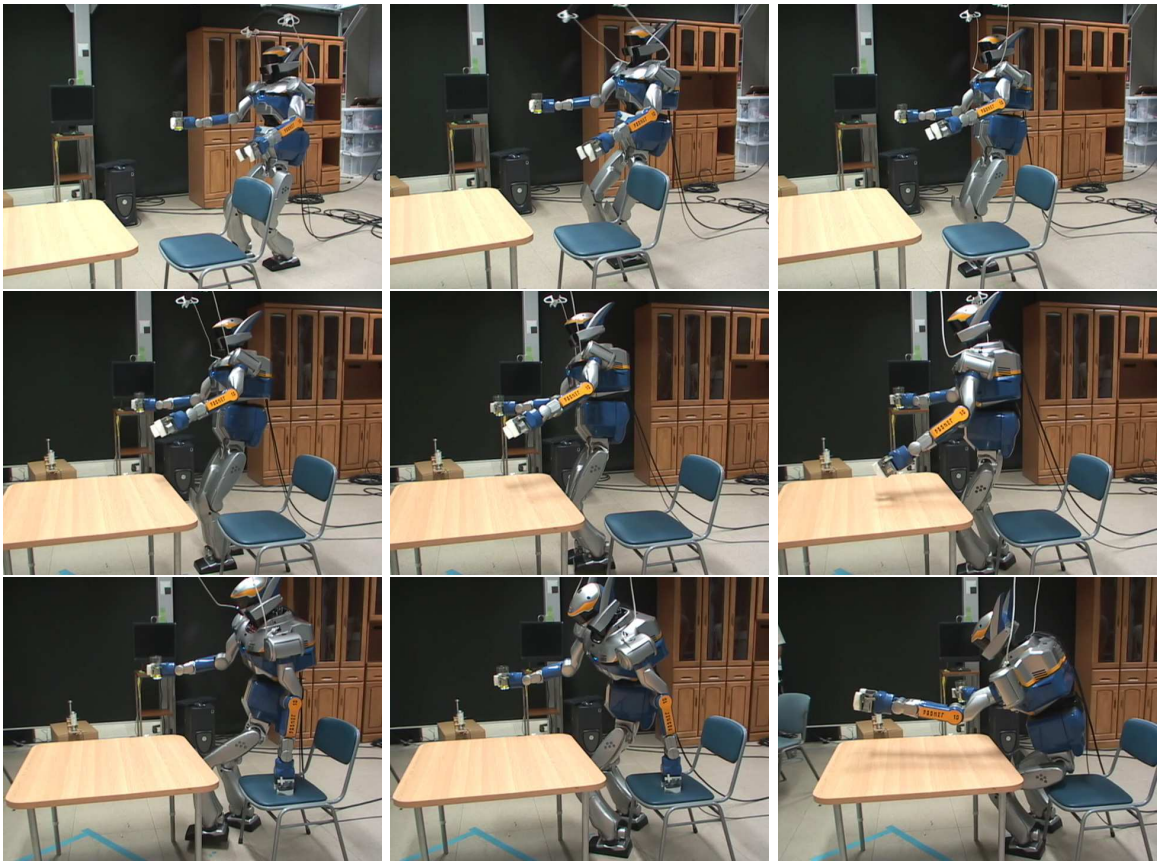


FIG. *7.14 – S'asseoir à une table avec un verre à la main : expérience sur HRP-2.*

puissant. Elle permet en effet de tester la possibilité de sauter des contacts, ce qui constitue une première étape vers un filtrage, et en utilisant la génération de postures multiples, on peut modifier l'emplacement d'un contact c selon un critère, tout en gardant constant les autres contacts des ensembles de contacts contenant c.

7.5.2 Générateur de posture

Nous présentons ici quelques chiffres relatifs à l'utilisation du générateur de posture dans la planification (et uniquement la planification, par exemple l'utilisation comme cinématique inverse n'est pas comptée). Ces données sont tirées d'environ 10Go provenant des logs de près de $2,5$ millions d'appels au générateur, collectés sur environ deux ans.
Pour ne pas multiplier les résultats, et parce qu'il manque à ces données une information importante qui est de savoir si les calculs ont été faits avec du code optimisé ou non (ce qui a changé au gré du développement du planificateur et des tests), nous regroupons toutes ces données en un seul tableau et un seul graphique, sans séparer en fonction des options de FSQP choisies (notamment le nombre maximal d'itérations, qui vaut dans ces données 100, 150 ou 200, mais aussi la marge de tolérance sur les égalités ou encore l'utilisation d'un affichage de debogage).
Nous obtenons cependant un bon aperçu de la façon dont est utilisée la génération de posture,

et du temps passé selon la sortie (succès ou différentes erreurs) de FSQP.

FSQP peut soit réussir son optimisation, soit essuyer un échec pour différentes raisons (cf chap. 3) qui sont signalées par un code. Certains de ces codes ne sont pas présents ici car ne correspondant pas à notre problème. Ainsi en est-il de la non-résolution des contraintes linéaires (code 1), puisque nous n'en avons pas.
La table 7.1 présente le nombre de générations ayant abouti à chaque code et les temps moyens et totaux selon ces codes. On peut souligner plusieurs points :

- une génération prend en moyenne 0,74 secondes, mais une génération réussie ou ayant échoué en phase de recherche d'un point satisfaisant les inégalités linéaires est plus rapide en moyenne. L'exécution la plus rapide est d'ailleurs lorsque la recherche de ce point échoue, ce qui justifie notre choix d'utiliser des pré-contacts pour tester rapidement la non faisabilité d'un nouveau contact entre un corps et une surface donnés,
- les temps sont tirés vers le haut principalement par les générations atteignant le maximum d'itérations, bien que ne représentant qu'environ un huitième des cas, elles comptent pour plus de la moitié dans les temps de calculs totaux. Elles sont environ dix fois plus longues en moyenne qu'une génération réussie,
- les générations réussies représentent un tiers environ du total des générations,
- il y a un net partage entre les deux premiers cas, qui représentent les trois quarts des générations mais 22% du temps de calcul, et les autres,

code FSQP	signification	nombre	tps moyen(s)	tps total(s)
0	succès	763 037	0.335	255 282
2	inégalités non satisfaites	1 134 141	0.138	156 981
3	maximum itérations atteint	310 502	3.417	1 061 100
4	échec de la recherche linéaire	142 217	0.987	140 332
5	construction de la direction échouée	50 357	2.24	112 800
8	$\mathbf{x}^k$ et $\mathbf{x}^{k+1}$ numériquement équivalents	83 767	0.895	75010
9	égalités non-linéaires non satisfaites	14 238	3.35	47634
totaux		2 498 259	0.740	1 849 138

TAB. *7.1 – Temps passé dans la génération de posture selon la raison de l'arrêt de l'optimisation.*

La figure 7.15 donne un aperçu de la façon dont se répartissent les différents codes en fonction du temps de calcul. Les courbes sont certainement déformées par l'utilisation de différents paramètres. On observe cependant bien que certains cas ont une variance assez grande.

7.5.3 Expérimentations

Les différentes expériences réalisées ont soulevé plusieurs points tant au niveau de la conception des robots que des logiciels pour les piloter.

En premier lieu, il faut pouvoir jouer de manière robuste la planification. Les différences entre le robot et l'environnement d'une part, leurs modèles numériques d'autre part, conduisent à des erreurs dans l'exécution qui peuvent poser problème. Si un contact ne se fait pas à l'endroit voulu par exemple, cela peut induire une déformation de la posture qui conduira soit à un déséquilibre, soit à un contact suivant encore plus mal placé. Si par exemple la table est

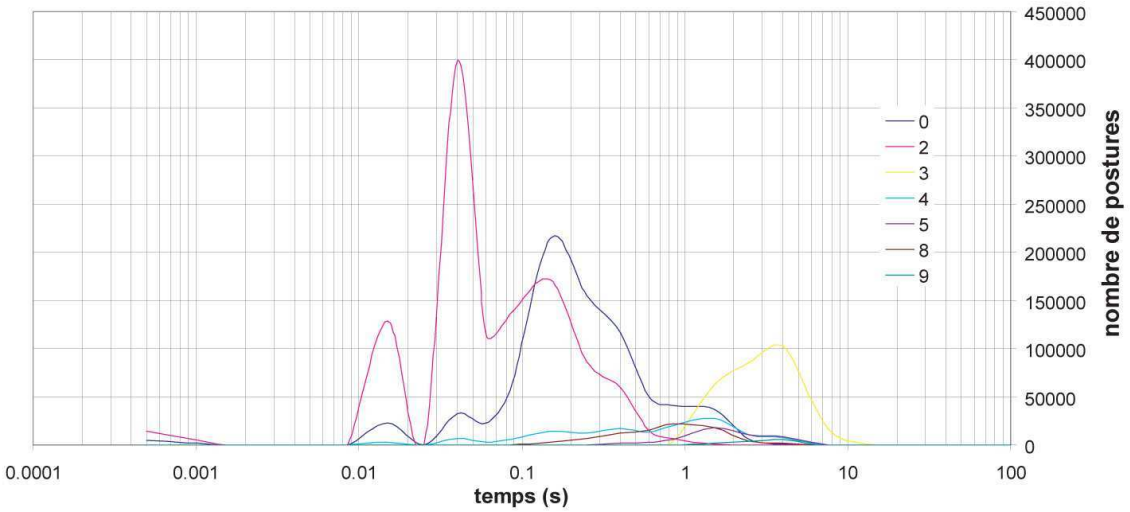


FIG. *7.15 – Nombre de tentatives de génération en fonction du temps passé à générer, pour chaque type de sortie de FSQP.*

plus basse que prévu, le contact avec la main ne se fera pas au moment voulu mais lorsque le robot voudra s'appuyer sur sa main. Le robot penchera donc, et posera trop tôt le pied qu'il déplaçait. Si au contraire la table est trop haute, le robot va trop appuyer dessus, au risque de se déstabiliser. De tels exemples se résoudraient par l'utilisation de commandes gardées couplées à un schéma de contrôle comme une pile de tâches, mais cette utilisation soulève un autre problème.

Pour effectuer une commande gardée, il faut en effet mesurer une force, or les robots n'ont dans la plupart des cas que peu ou pas de capteur de forces. Plus généralement, il manque aux robots la capacité de détecter des contacts sur toute la surface de leur corps, ou au moins sur une surface importante. Un robot tel que HRP-2 ne dispose de capteurs que dans les bases des préhenseurs ou les liaisons pieds-jambes et ne peut donc pas détecter de contact sur ses bras, ses jambes ou plus généralement sur sa carapace.

A défaut de capteurs supplémentaires, et de manière non exclusive, nous avons aussi pu observer les avantages d'une compliance passive. Dans le cadre du scénario de la canette tout d'abord, dans les deux autres ensuite, où la surface de la chaise permet des transitions plus douces et plus sûres, lorsque le robot met la main dessus. Il serait souhaitable que les robots aient des revêtements compliants au dessus de leur structure rigide. Il nous est apparu que la flexibilité de la peau du robot est plus importante et plus sûre que toute autre méthode de compliance (grâce à des articulations actives ou passives). En effet, non seulement une peau compliante absorbe les chocs, mais en se déformant, elle agrandit la surface de contact et donc augmente la stabilité.

7.5.4 Travaux futurs

Plusieurs axes de recherche peuvent être envisagés en vue d'étendre les possibilités du planificateur [Kheddar 08]. En premier lieu, l'ajout dans l'environnement d'objets qui peuvent être déplacés. Un tel problème a déjà été étudié dans [Stilman 06], mais la solution se restreint aux cas où le robot est capable de marcher. Nous voudrions résoudre des problèmes tels que

celui présenté par la figure 7.16 : le robot doit monter au grenier par l'échelle. Pour cela, le planificateur ne doit pas seulement trouver des appuis sur l'échelle, mais aussi une séquence permettant l'ouverture de la trappe. Le contact sur la trappe en ce cas, ne peut pas servir d'appui.

FIG. *7.16 – Monter au grenier. Exemple de scénario requérant de bouger un objet de l'environnement.*

Une seconde piste serait la prise en compte d'objets déformables dans la planification. Cela permettrait d'appréhender correctement les surfaces qui se déforment, qu'elles soient sur le robot ou dans l'environnement. On pourrait aussi s'attaquer à des environnements tels que celui donné par la figure 7.17 : un avatar doit atteindre une étagère au dessus d'un lit. Pour cela il doit s'appuyer sur le lit, qui se déforme. Ce type de contact demande d'inclure un modèle de déformation dans la génération de posture.

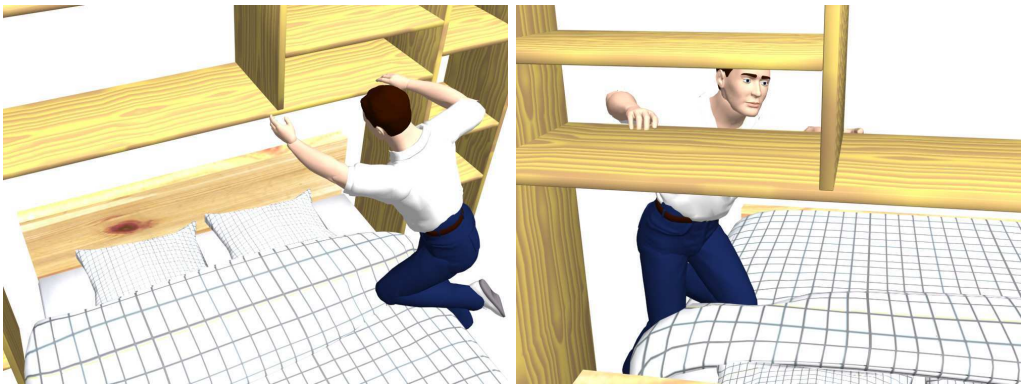


FIG. *7.17 – Atteindre une étagère. Exemple de scénario avec objet déformable.*

Enfin, un dernier axe serait l'utilisation de ce planificateur de manière locale pour suivre une trajectoire fournie de manière interactive par un utilisateur, via une manette de jeu par exemple. Cette extension demande le développement d'heuristiques pour accélérer les calculs, et de manière plus générale, des efforts de recherche et logiciels pour tendre vers le temps réel. On pourra entre autres profiter du fait que la sortie de minimums locaux importants est laissée au soin de l'utilisateur, qui s'occupe de la partie globale.

Conclusion

Cette thèse propose un planificateur d'appuis, nécessaire pour des robots dont la mobilité repose sur une interaction avec l'environnement, afin de résoudre des problèmes requérant des mouvements fortement acycliques que la majorité des planificateurs ne sont pas en mesure de fournir, en grande partie du fait des simplifications ou des restrictions qu'ils font sur les mouvements possibles.

Nous avons pour cela étudié la structure de l'espace de configuration induit 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.
A cette fin, nous avons développé 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 qu'il résout alors. Ce générateur de posture est fortement générique et permet la résolution de nombreux problèmes autres que la satisfaction de contacts, ce qui a permis d'en faire usage dans des situations bien différentes de la planification, comme l'observation d'un objet en vue de sa reconstruction.

Ce générateur a par ailleurs nécessité, afin d'inclure efficacement les contraintes d'évitement de collision, de s'intéresser à la cause de discontinuités dans certains cas du gradient de la distance entre deux objets. Il s'est avéré que le problème était dû à la non-stricte convexité des deux objets. Nous avons alors proposé un nouveau type de volume englobant strictement convexe, le STP-BV, qui approxime très bien les polyèdres convexes et permet des calculs rapides de distance. Nous avons implémenté une méthode pour faire ces calculs sous forme d'une étape à temps constant ajoutée au-dessus d'un des algorithmes de calcul de distance les plus rapides et fiables.
Outre la génération de posture, ces volumes ont aussi démontré leur intérêt pour l'ajout de contraintes de distance dans une loi de commande en garantissant la continuité de la vitesse.

Grâce à ces outils, nous avons développé 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 un potentiel, un arbre dont les nœuds sont des ensembles de contacts. L'arbre croît 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.
La construction efficace de cet arbre est un défi, car à chaque pas le nombre potentiel de nœuds est immense, à cause du nombre possible de sous-variétés et de la nature continue

des emplacements de contact. Pour résoudre ce problème, il faut à la fois choisir au mieux le nœud de l'arbre à étendre à chaque itération, et réduire le nombre de fils à construire pour ce nœud tout en gardant ouverte toute possibilité d'évolution. Pour le premier point, nous avons proposé de baser le champ de potentiel sur une trajectoire guide qui peut être soit spécifiée par l'utilisateur, soit générée automatiquement. La résolution du second point passe par une combinaison astucieuse du champ de potentiel et de la génération de posture qui permet de prédire quels fils sont nécessaires.

Grâce à ce planificateur, nous avons été en mesure de résoudre des problèmes complexes du fait d'un environnement contraint. Les plans générés pour ces problèmes ont alors été portés sur un robot réel avec succès et ont permis de mettre en évidence les points techniques et technologiques qu'il faudra résoudre pour disposer de robots avec toutes les capacités nécessaires pour des interactions avancées avec l'environnement.

Nous avons aussi abordé les travaux futurs sur ce planificateur. Il vise majoritairement à l'extension de celui-ci à des environnements plus généraux comportant des objets déformables (y compris le robot même) et mobiles.
Avec plus de recul, il y a aussi un travail à faire sur l'utilisation de ce planificateur. En effet, de part sa généralité, il n'est pas aussi efficace dans un cas particulier (par exemple la marche) qu'un planificateur dédié. Il est cependant nécessaire dès que l'on sort du cadre de ces planificateurs spécifiques. Dans un environnement humain, cette sortie de cadre est souvent locale, bien que fréquente. Il s'agit donc d'étudier comment associer différents planificateurs pour tirer parti du meilleur de chacun selon le contexte. Cela passerait par la construction d'un méta-planificateur.

Annexes

Annexe **A**

Calcul de géométrie directe sur un robot

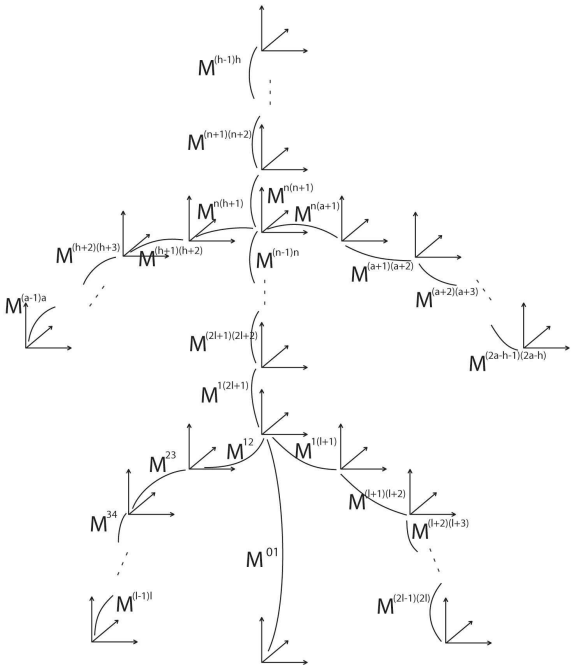


FIG. *A.1 – Architecture cinématique classique d'un robot humanoïde.*

Pour exprimer des contraintes telles que les contacts ou les collisions, il est nécessaire de pouvoir calculer la position d'un corps du robot, ou d'un point de celui-ci pour une configuration donnée, c'est-à-dire, être capable de faire des calculs de géométrie directe sur le robot.
Nous considérons des robots dont la structure cinématique est un arbre, avec les caractéris-

tiques suivantes : (i) la base est totalement libre de se mouvoir en rotation et translation, (ii)les liaisons entre deux corps consécutifs sont uniquement des liaisons pivot ou glissière (un degré de liberté). Nous adoptons les matrices de transformation homogènes pour représenter les relations entre deux corps :

$$M = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{A.1}$$

avec R matrice 3×3 de rotation, $\mathbf{t}$ un vecteur de translation et $\mathbf{0}$ le vecteur nul. Puisqu'il n'y a qu'un seul degré de liberté entre deux corps i et j consécutifs, la matrice de transformation correspondante est

$$M^{ij}(q_{j-1}) = \begin{pmatrix} R^{ij}(q_{j-1}) & \mathbf{t}_0^{ij} \\ \mathbf{0}^T & 1 \end{pmatrix} \quad \text{ou} \quad M^{ij}(q_{j-1}) = \begin{pmatrix} I & \mathbf{t}^{ij}(q_{j-1}) + \mathbf{t}_0^{ij} \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{A.2}$$

où $\mathbf{t}_0^{ij}$ est une translation constante entre les deux objets, et I est la matrice identité 3x3. Généralement, la translation ou la rotation entre un corps et le suivant se fait selon un des axes du repère de l'un d'eux. On a alors $\mathbf{t}^{ij}(q_{j-1}) = q_{j-1} \cdot \mathbf{t}_u^{ij}$ avec $u = x$, $u = y$, ou $u = z$ et

$$\mathbf{t}_x^{ij} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{t}_y^{ij} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \text{et} \quad \mathbf{t}_z^{ij} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{A.3}$$

De même, $R^{ij}(\theta)$ prend une des formes suivantes :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, \quad \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \quad \text{ou} \quad \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.4}$$

selon que la rotation se fait autour de $\mathbf{t}_x^{ij}$, $\mathbf{t}_y^{ij}$ ou $\mathbf{t}_z^{ij}$.

Un cas particulier est la matrice de transformation entre le repère absolu et le corps racine du robot (on numérote 0 et 1 les repères respectifs) . Celle-ci représente en effet une transformation libre à 6 degrés de liberté. On adopte la convention suivante pour les 6 paramètres correspondants, et pour la construction de la matrice : on effectue dans cet ordre une rotation autour de l'axe x, puis y puis z. Ces rotations sont paramétrisées par q_{n+4}, q_{n+5} et q_{n+6}. Puis on effectue une translation de vecteur $(q_{n+1}, q_{n+2}, q_{n+3})^T$. On obtient alors la matrice $M^{01}(q_{n+1}, q_{n+2}, q_{n+3}, q_{n+4}, q_{n+5}, q_{n+6})$ suivante où q_k^e est un raccourci pour q_{n+k} :

$$\begin{pmatrix} \cos q_5^e \cos q_6^e & \sin q_4^e \sin q_5^e \cos q_6^e - \cos q_4^e \sin q_6^e & \cos q_4^e \sin q_5^e \cos q_6^e + \sin q_4^e \sin q_6^e & t_x^{01} + q_1^e \\ \cos q_5^e \sin q_6^e & \sin q_4^e \sin q_5^e \sin q_6^e + \cos q_4^e \cos q_6^e & \cos q_4^e \sin q_5^e \sin q_6^e - \sin q_4^e \cos q_6^e & t_y^{01} + q_2^e \\ -\sin q_5^e & \sin q_4^e \cos q_5^e & \cos q_4^e \cos q_5^e & t_z^{01} + q_3^e \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.5}$$

De manière très classique, la position dans le repère absolu d'un point de coordonnées $(x, y, z)^T$ dans le corps i du robot est obtenu par $M^i(x, y, z, 1)^T$ où M^i est la matrice de transformation du $i^{ème}$ corps par rapport au repère du monde. Un vecteur $(v_x, v_y, v_z)^T$ dans le corps i a pour expression dans le repère absolu $R^i(v_x, v_y, v_z)^T$ où R^i est la matrice de rotation extraite de M^i.

Soit $\{0, 1, i_2, i_3, ..., i_k, i\}$ la chaîne cinématique du repère absolu au $i^{ème}$ corps. On a $M^i =$

$M^{01} \cdot M^{1i_2} \cdot M^{i_2i_3} \cdot ... \cdot M^{i_ki}$
Effectuer ce calcul à chaque nouveau $\mathbf{q}$ pour chaque corps nécessaire est loin d'être optimal car plusieurs chaînes $\{0, 1, i_2, ..., i\}$ partagent des sous-chaînes communes, et donc des calculs communs. D'autre part les matrices M^{ij}, M^{01} mise à part, comptent un grand nombre de 0. Il y en a au moins 7 si la transformation correspond à une rotation, 9 au minimum pour une translation. Faire intervenir ces 0 dans les multiplications de matrices est une perte de temps en terme de multiplications scalaire, additions et assignations.
Pour les deux points cités ci-avant, la connaissance a priori du modèle du robot permet d'optimiser les calculs à faire en générant un code spécifique pour les effectuer. Comme on change peu de modèle de robot par comparaison avec les changements d'environnement ou de problème à réaliser dans ces environnements, ces pré-calculs sont tout à fait acceptables.

Une première approche consiste à générer une routine calculant toutes les matrices de transformations M^i ensemble. Des programmes comme Maple™ sont capables de générer du code efficace et optimisé pour des calculs donnés. L'INRIA fournit l'ensemble d'outils logiciel HuMAnS [Wieber 06] qui comprend des routines en Maple™ générant des codes relatifs au calcul de la géométrie et la dynamique d'un robot. Nous avons adapté ces codes pour générer une fonction qui prend $\mathbf{q}$ en entrée et retourne les matrices de transformation M^i de tous les corps du robot ainsi que la position du CdM dans le repère absolu. Cette fonction ne calcule que les coefficients non nuls ou non constants, ne comprend aucune multiplication inutile (par 0 ou 1), et factorise les calculs de façon à ne pas faire de répétition. Au final, le gain de temps est conséquent. Par rapport à une implémentation classique, il est de l'ordre de 50% pour le calcul du CdM, qui fait intervenir le calcul de toutes les matrices.

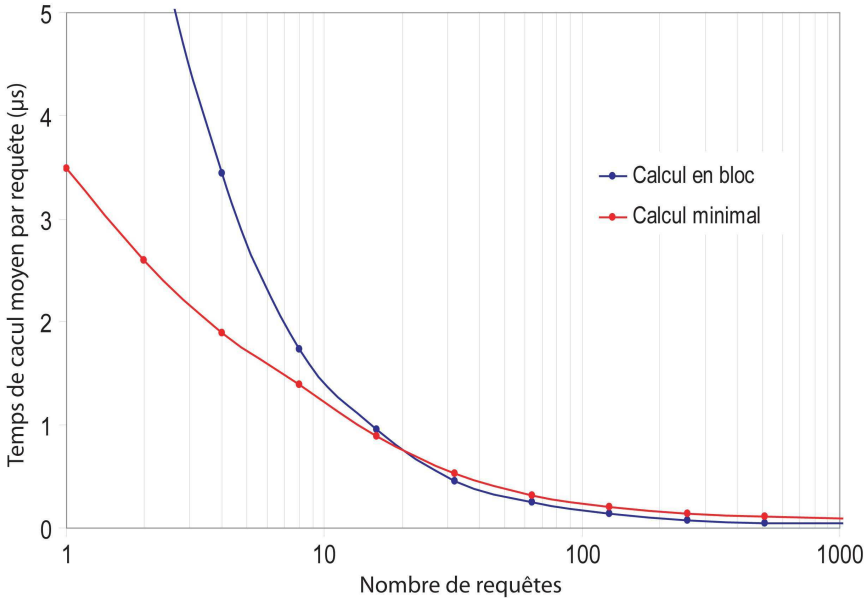


FIG. *A.2 – Comparaison du temps de calcul pour deux méthodes de calcul des matrices $M^i(\mathbf{q})$ en fonction du nombre de requêtes pour un $\mathbf{q}$ fixé. Quand le nombre de requête est faible (ici inférieur à 20 pour un robot à 31 corps), il est plus intéressant de calculer les matrices au cas par cas. Quand le nombre devient plus grand, il est plus rapide de faire tous les calculs en bloc.*

Cette approche est cependant susceptible de faire des calculs inutiles si on ne se sert

pas de toutes les matrices, en particulier si on ne s'intéresse pas à une branche de l'arbre (correspondant à un membre). On voudrait pouvoir ne calculer que les matrices nécessaires, sans pour autant perdre les optimisations évoquées avant. Cela se fait en fragmentant la fonction précédente de la façon suivante : si i et j sont les indices de deux corps consécutifs, la sous-fonction en charge de retourner M^j effectue uniquement les opérations nécessaires à son calcul qui n'ont pas déjà été faites pour M^i. A chaque fois, tous les résultats de calculs intermédiaires sont stockés et accessibles par les autres sous-fonctions. A l'appel d'une sous-fonction f_j, on vérifie par le biais d'un drapeau s'il est nécessaire d'effectuer les calculs qu'elle contient (i.e. on regarde si la fonction n'a pas encore été appelée pour ce $\mathbf{q}$). Si tel est le cas alors, on appelle la fonction correspondant au corps précédent f_i, puis on effectue les calculs de cette fonction f_j. On remonte ainsi récursivement la chaîne de calculs à effectuer jusqu'à arriver à des calculs déjà faits ou à la racine du robot. Cette méthode revient à calculer les matrices M^{ij} quand on en a besoin et à les multiplier entre elles : globalement, la fonction f_j calcule M^{ij} et la multiplie avec M^i. La différence vient de ce que f_j a connaissance de la places des 0 dans M^{ij} et M^i et en tire parti pour accélérer les calculs.
Si elle limite les calculs, cette seconde méthode a cependant un inconvénient par rapport à la première, celui de devoir faire des tests de drapeaux. Quand on a besoin de toutes les matrices, comme c'est le cas lors du calcul du CdM, ces tests sont en surcoût par rapport à la première méthode. Intuitivement, si il y a besoin de peu de matrices et que celles-ci sont sur de mêmes branches cinématiques, alors la seconde méthode sera plus rapide. Si au contraire on a besoin de la plupart des matrices, et en particulier sur des branches différentes, alors la première méthode sera meilleure. C'est ce que confirme le graphique A.2 dans le cas du robot HRP-2.
Dans le cas de FSQP, il y a souvent évaluation d'un petit nombre de contraintes (ce qui équivaut souvent à un petit nombre de matrices de transformation nécessaires) en de nombreux points $\mathbf{q}$. Cela arrive lors de la recherche d'un pas selon la direction fournie par le QP. La seconde méthode est alors plus avantageuse. Elle permet de gagner environ 5% de temps de calcul sur la génération de posture totale dans le cas du robot HRP-2.

Annexe B

Compléments sur les STP-BV

B.1 Preuve du théorème 4.3.2

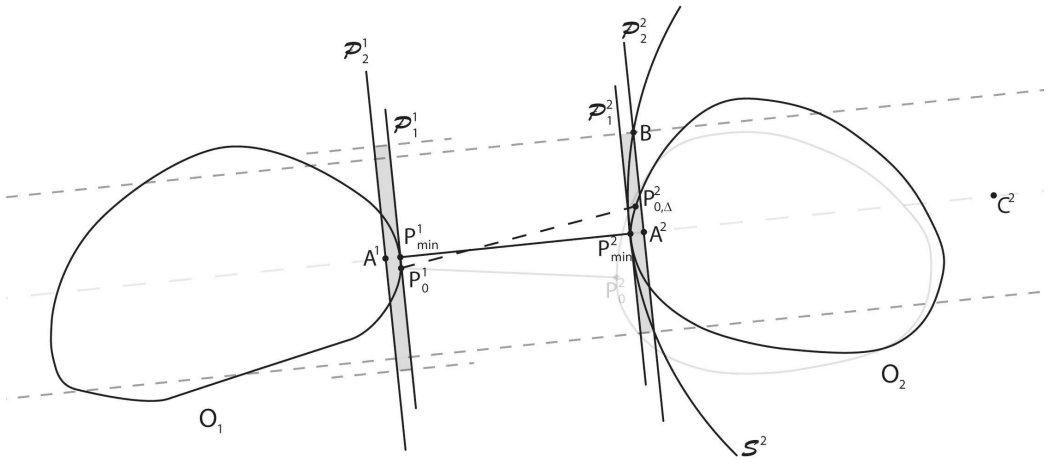


FIG. *B.1 – Démonstration de la continuité de* $\mathbf{p}_{\min}$.

Theorem 4.3.2 : Les points témoins de la distance entre deux objets convexes sont des fonctions continues de $\mathbf{q}$ si au moins l'un d'eux est strictement convexe ($\mathbf{u}_{\min}$ et $\mathbf{p}_{\min}$ sont des fonctions de $\mathbf{q}$).

Preuve: Tout d'abord, si aucun corps n'est strictement convexe, les points témoins ne sont pas nécessairement uniques et il n'y a donc pas continuité. Il nous faut donc démontrer que la stricte convexité implique la continuité. L'idée de la démonstration est de construire des volumes qui contiennent les points témoins pour deux configurations différentes et de montrer que lorsque ces deux configurations tendent l'une vers l'autre, ces volumes tendent vers des points.

Considérons deux objets convexes O_1 et O_2, le second étant strictement convexe. Du fait de

la stricte convexité, il existe une sphère incluant O_2 tout en lui étant tangent, quelque soit le point de tangence. Soit R le rayon de cette sphère.
On considère les objets pour une position relative décrite par $\mathbf{q}_0$. P_0^1 et P_0^2, respectivement sur O_1 et O_2 sont les uniques points témoins pour cette position. $\mathbf{p}_{\min}(\mathbf{q}_0) = (P_0^1, P_0^2)^T$
On bouge à la position $\mathbf{q}_0 + \Delta\mathbf{q}$. On note $P_{0,\Delta}^2$ la nouvelle position du point P_0^2, et $P_{\min}^1$ et $P_{\min}^2$ les nouveaux points témoins. $\mathbf{p}_{\min}(\mathbf{q}_0 + \Delta\mathbf{q}) = (P_{\min}^1, P_{\min}^2)^T$.
Comme δ est lipschitzienne, il existe K tel que $|\delta(\mathbf{q}_0 + \Delta\mathbf{q}) - \delta(\mathbf{q}_0)| \leq K\Delta\mathbf{q}$ (cf [Rusaw 01]).
Soient $\mathcal{P}_1^1$ et $\mathcal{P}_1^2$ les plans tangents à O_1 et O_2 en $P_{\min}^1$ et $P_{\min}^2$. $\mathcal{P}_2^2$ (resp. $\mathcal{P}_2^1$) est le plan parallèle à $\mathcal{P}_1^2$ (resp. $\mathcal{P}_1^1$) distant de $\delta(\mathbf{q}_0) + K\Delta\mathbf{q}$ de $P_{\min}^1$ (resp. $P_{\min}^2$). $\mathcal{S}^2$ est la sphère tangente à O_2 en $P_{\min}^2$ et de rayon R. Son centre C^2 est aligné avec $P_{\min}^1$ et $P_{\min}^2$. A^1 et A^2 sont les points de $\mathcal{P}_2^1$ et $\mathcal{P}_2^2$ sur cet alignement et B est un point sur l'intersection de $\mathcal{P}_2^2$ avec $\mathcal{S}^2$ (cette intersection n'est pas vide dès que $\Delta\mathbf{q}$ est suffisamment petit).

Soit $\sigma = \sqrt{|\delta^2(\mathbf{q}_0 + \Delta\mathbf{q}) - \delta^2(\mathbf{q}_0)|}$. σ tend vers 0 quand $\Delta\mathbf{q}$ fait de même.
On définit $\mathcal{C}_{\Delta\mathbf{q}}^1$ (resp. $\mathcal{C}_{\Delta\mathbf{q}}^2$) comme étant le cylindre d'axe $(P_{\min}^1 P_{\min}^2)$ et de rayon $A^2B + \sigma$ (resp. A^2B), délimité par $\mathcal{P}_1^1$ et $\mathcal{P}_2^1$ (resp. $\mathcal{P}_1^2$ et $\mathcal{P}_2^2$). $A^2B + \sigma$ est la distance maximale possible entre $P_{\min}^1$ et P_0^1, obtenue pour le cas extrême où $P_{0,\Delta}^2$ est sur la frontière de $\mathcal{C}_{\Delta\mathbf{q}}^2$.
Soit $E_{\Delta\mathbf{q}} = \mathcal{C}_{\Delta\mathbf{q}}^1 \times \mathcal{C}_{\Delta\mathbf{q}}^2$.
Par construction, $(P_{\min}^1, P_{\min}^2)$ et $\left(P_0^1, P_{0,\Delta\mathbf{q}}^2\right)$ sont dans $E_{\Delta\mathbf{q}}$.
$A^1P_{\min}^1 = A^2P_{\min}^2 = \delta(\mathbf{q}_0) + K\Delta\mathbf{q} - \delta(\mathbf{q}_0 + \Delta\mathbf{q})$,
$A^2C^2 = R - A^2P_{\min}^2$,
et $A^2B = \sqrt{(R^2 - A^2C^{2^2})}$.
Comme δ est une fonction continue, $A^2P_{\min}^2$ et $A^1P_{\min}^1$ tendent vers 0 quand $\Delta\mathbf{q}$ tend vers 0.
Ainsi, A^2C^2 tend vers R et A^2B tend vers 0.
Les deux cylindres tendent vers un singleton : $E_{\Delta\mathbf{q}}$ tend vers $\left\{\left(P_0^1, P_{0,\Delta\mathbf{q}=0}^2\right)\right\} = \{(P_0^1, P_0^2)\}$.
On a alors $\mathbf{p}_{\min}(\mathbf{q}_0 + \Delta\mathbf{q}) = (P_{\min}^1, P_{\min}^2)^T$ tend vers $\mathbf{p}_{\min}(\mathbf{q}_0) = (P_0^1, P_0^2)$ ce qui démontre la continuité en $\mathbf{q}_0$. □

B.2 Avoir une surface C^1 n'a pas d'impact sur la continuité de la distance

(a) cas C^0 (b) cas C^1

FIG. *B.2 – Distance entre un plan et un objet C^0 ou C^1.*

Considérons, dans un espace 2D, deux arcs de cercles distincts de même rayon R, se rejoi-

gnant à l'origine du repère O, délimitant un objet strictement convexe O_1 qui est seulement C^0. θ est l'angle que fait l'axe de symétrie avec l'axe vertical, θ_0 l'angle non orienté entre le diamètre passant par O d'un cercle et l'axe de symétrie. On considère la distance $\delta(\theta)$ entre cet objet et la droite $z = -z_0$.

$$\delta(\theta) = \begin{cases} R\left(\cos(\theta_0 - \theta) - 1\right) + z_0 & \text{if } \theta < -\theta_0, \\ z_0 & \text{if } -\theta_0 \leq \theta \leq \theta_0, \\ R\left(\cos(\theta - \theta_0) - 1\right) + z_0 & \text{if } \theta > \theta_0 \end{cases} \tag{B.1}$$

Cette fonction est C^1, mais pas C^2 :

$$\delta''(\theta) = \begin{cases} -R\cos(\theta_0 - \theta) & \text{if } \theta < -\theta_0, \\ 0 & \text{if } -\theta_0 \leq \theta \leq \theta_0, \\ -R\cos(\theta - \theta_0) & \text{if } \theta > \theta_0 \end{cases} \tag{B.2}$$

On considère maintenant deux arcs de cercles de rayons r et R ($R > r$), dont les centres sont alignés avec l'origine O (cf Fig. B.2). La surface est C^1, et

$$\delta(\theta) = \begin{cases} R\left(\cos(\theta) - 1\right) + z_0 & \text{if } \theta < 0, \\ r\left(\cos(\theta) - 1\right) + z_0 & \text{if } \theta \geq 0 \end{cases} \tag{B.3}$$

Et nous avons toujours la discontinuité de la dérivé seconde, due au changement de rayon :

$$\delta''(\theta) = \begin{cases} -R\cos(\theta) & \text{if } \theta < 0, \\ -r\cos(\theta) & \text{if } \theta \geq 0 \end{cases} \tag{B.4}$$

Cette discontinuité est cependant moins importante dans ce cas $R - r$ au lieu de R dans le premier cas.

B.3 Construction des STP-BV

La construction d'un STP-BV est détaillée par l'algorithme 12. Elle ne se base pas sur les faces du polyèdre, mais travaille directement avec le nuage de points 3D des sommets du polyèdre.
L'idée de base est similaire à celle de l'algorithme dit de *marche de Javis* : on trouve en premier lieu une face dont la grande sphère correspondant contient tous les points du nuage et leurs petites sphères associés. On fait alors tourner cette sphère autour des arêtes de la faces jusqu'à rencontrer la petite sphère d'un point. L'arête et le point forment une nouvelle face. Sa sphère est la seule contenant tous les points du nuage et étant basée sur l'arête, exception faite de la sphère précédente. On tourne alors autour des arêtes de cette nouvelle face, et ainsi de suite, jusqu'à atteindre une face déjà calculée.

La tâche de trouver la première face revient à la fonction *init* (algorithme 13). L'algorithme principal est basé sur la remarque suivante : puisqu'on ne peut tourner qu'autour d'une seule arête à chaque instant, il faut maintenir une liste des arêtes autour desquelles on n'a pas encore tourné. Bien que nous puissions choisir n'importe laquelle des arêtes de cette liste pour aller au pas suivant, nous sélectionnons celle autour de laquelle il faudra faire la plus

Algorithm:construction de STP-BV : pseudo-code

Data: un nuage de points, les valeurs de r et R
Result: un ensemble de faces avec leur sphères
-e, e_1 et e_2 : *sont des arêtes accompagnées d'un sommet supplémentaire*
-v,v_1,v_2 et v_3 : *sont des sommets*
-s et s' : *sont des données relatives à des sphères*
-*vertices* : *l'ensemble de sommets en entrée*
-edgeStack : *une liste d'arêtes accompagnées de deux sommets, et triée selon un angle.*
-output : *une liste de triangles et leurs sphères tangentes*

```
BuildVolume()
begin
    init(edgeStack, output)
    while ( !empty (edgeStack)) do
        (e, v) ← first(edgeStack)
        (e1, e2) ← newEdges(e,v)
        push(output, face(e,v))
        if contains(edgeStack, e1) then
            delete(edgeStack, e1)
        else
            insert(edgeStack, e1, angleMin(e1))
        end
        if contains(edgeStack, e2) then
            delete(edgeStack, e2)
        else
            insert(edgeStack, e2, angleMin(e2))
        end
    end
    return output
end
```

Algorithm 12: Construction du STP-BV

petite rotation (l'explication de ce choix est donnée plus loin). De ce fait, les arêtes doivent être conservées avec des données supplémentaires afin de pouvoir calculer l'angle de cette rotation. La structure d'arête contient alors trois sommets : les deux extrémités de l'arête et un additionnel correspondant au troisième sommet de la face à laquelle l'arête appartient déjà. De cette façon, on peut savoir d'où est fait la rotation autour de l'arête, et dans quelle direction.
La fonction *angleMin* prend en argument une telle structure d'arête et retourne l'angle de la rotation qu'il faut effectuer autour de cette arête pour atteindre le sommet suivant, ainsi que le sommet en question. La liste d'arêtes, appelée *edgeStack* dans le pseudo-code de l'algorithme, contient donc des arêtes appariées chacune à un sommet et rangées dans l'ordre croissant de la valeur d'angle qui leur est associée.

La fonction principale de *buildVolume* est de mettre à jour cette liste, grâce aux fonctions classiques *first*, *insert*, *delete* et *contains*. La fonction *first* dépile l'élément qu'elle vient de lire.

Algorithm: Fonctions auxiliaires pour la construction de STP-BV : Pseudo-code

```
init(edgeStack, faceList)
begin
    for each (v1,v2,v3) ∈ vertices^3 with v_i ≠ v_j do
        s ← sphere(v1,v2,v3)
        if allPointsInSphere(s) then
            for i ∈ {1,2,3} do
                e_i ← edge(v_i, v_(i+1)[3] ,v_(i+2)[3])
                insert(edgeStack, e_i, angleMin(e_i))
            end
            push(faceList, face(e,v))
            return SUCCESS
        end
        return FAIL "le rayon de la grande sphère est trop petit"
    end
end

angleMin(e)
begin
    (v1,v2,v3) ← e
    s ← sphere(v1,v2,v3)
    for each (v) ∈ vertices − {v1, v2, v3} do
        s' ← sphere(e,v)
        if allPointsInSphere(s') then
            a = angle(s, s', e)
            return (a,v)
        end
    end
end
```

Algorithm 13: Principales fonctions auxiliaires pour la construction de STP-BV.

Partant d'une arête et d'un sommet, définissant une face, *newEdges* construit simplement deux nouvelles structures d'arête correspondant aux deux arêtes de cette face qui contiennent ce sommet. Si l'une de ces arêtes est déjà présente dans *edgeStack*, cela signifie que l'algorithme est revenu à une face déjà construite, puisque l'arête a déjà été crée. Dans ce cas, les deux faces adjacentes à cette arête ont déjà été trouvées, et l'arête doit donc être retirée de la liste. Dans le cas contraire, elle doit y être ajoutée. Une arête apparaît exactement deux fois dans cet algorithme (une fois par face adjacente) : elle est construite une première fois, puis est soit sélectionnée pour tourner autour, soit créée une seconde fois.
L'algorithme se termine quand il n'y a plus d'arêtes dans *edgeStack*.
sphere retourne la grande sphère d'une face, décrite soit par trois sommets, soit par une arête et un sommet. *face* renvoie une face accompagnée de sa sphère pour les mêmes données, et *angle* calcule l'angle entre deux sphères pour une arête donnée, c'est à dire l'angle fait par les centres des deux sphères et le point milieu de l'arête.

allPointsInSphere vérifie si tous les points du nuage sont à l'intérieur de la sphère donnée en entrée. Le fait de garder des sphères qui ne contiennent pas tous les points amènerait des

problèmes de non-convexité.

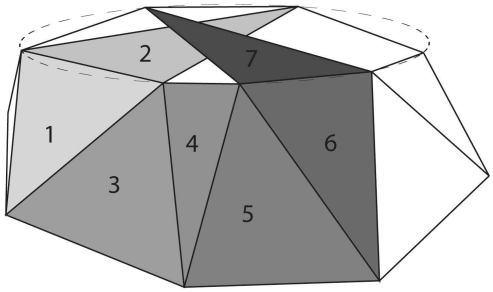


FIG. *B.3 – Mauvais comportement de l'algorithme lorsqu'il est confronté à des points cocycliques. L'algorithme construit les faces dans l'ordre indiqué par les nombres. Quand il tourne autour de l'arête supérieure de la face 6, le point qu'il choisit n'est pas le même que celui choisi lors de la construction de la face 2 à partir de la face 1. Il en résulte un chevauchement de faces.*

Les arêtes sont choisies dans l'ordre présenté ci-dessus pour des raisons de robustesse de l'algorithme vis-à-vis des erreurs numériques dans le cas d'un polygone dont les sommets des faces sont sur la même sphère. Dans ce cas, tous les sommets devraient être atteint en même temps lorsqu'on tourne autour d'une arête de ce polygone. Le sommet de plus petit indice serait alors sélectionné. Cependant, ce n'est parfois pas le cas, à cause d'erreurs d'arrondi. Il se peut alors que des sommets différents soient choisis selon l'arête par laquelle on arrive sur le polygone, cela ayant pour résultat, dans la majorité des cas, de construire des faces qui se recouvrent, ce qui fait échouer l'algorithme (cf Fig. B.3). Afin d'éviter cela, nous forçons l'algorithme à finir de couvrir un polygone qu'il vient de commencer, en choisissant l'arête pour laquelle la plus petite rotation est nécessaire. De cette manière, il n'y a pas besoin de définir un seuil à partir duquel des points sont considérés comme co-sphériques, ni d'avoir à chercher à chaque fois qu'on tourne autour d'une arête si il y a des points co-sphériques.

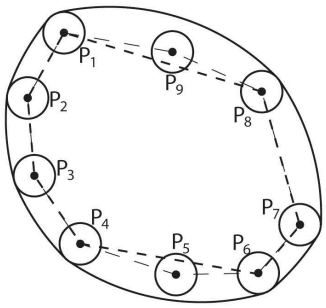


FIG. *B.4 – Différence entre l'enveloppe obtenue est l'enveloppe convexe originale dans le cas 2D. Du fait de la courbure des cercles, les points P_5 et P_9 ainsi que les petites sphères qui leur sont associées sont strictement à l'intérieur du STP-BV, et de ce fait ne font pas partie de l'enveloppe sous-jacente (ligne pointillée épaisse) alors qu'ils sont sur l'enveloppe convexe (ligne pointillée fine).*

Notre algorithme génère une espèce d'enveloppe convexe pour un objet, ainsi que les grandes sphères correspondantes à chacune des faces de cette enveloppe. A partir des sphères et des faces, il est aisé de déduire le STP-BV à proprement parler, ainsi que ses régions de voronoi.
L'enveloppe sous-jacente obtenue n'est généralement pas l'enveloppe du nuage de points ini-

tial ; en effet, certain points ont pu être ignorés pendant le processus de construction à cause de la courbure des sphères, comme illustré figure B.4 dans un exemple en 2D.

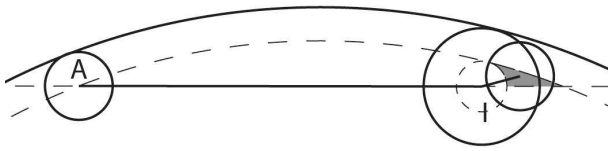


FIG. *B.5 – Un cas où l'enveloppe sous-jacente obtenue n'est pas convexe : afin de simplifier, on considère un triangle isocèle ABC vu dans son plan de symétrie (B et C sont donc hors du plan et se projette en I). Le cercle centré en A est la petite sphère de ce sommet, le cercle en trait plein centré en I est le tore lié à BC et le grand cercle en trait plein est la sphère associée à ABC. Les cercles en pointillés correspondent aux mêmes sphères et tore dont le rayon a été diminué de r. Tout point de la zone grisée peut être accepté par l'algorithme de construction quand il tourne autour de BC alors que ce point se trouve au dessus de la face ABC.*

Remarque 1 : L'enveloppe obtenue peut ne même pas être convexe dans de rares cas dégénéré mettant en jeu des triangles très fins, comme illustré sur la figure B.5. Cela peut être un problème car nous voulons utiliser par la suite des algorithmes de calcul de distance qui ne travaillent que sur des polyèdres convexes. Ce problème peut se résoudre de différentes manières, soit en bougeant ou supprimant le sommet en cause, soit en découpant l'enveloppe en parties convexes, par exemple. Cependant les essais menés avec une grande variété d'objets provenant d'applications usuelles ne nous ont jamais confrontés à ce cas. Nous n'avons donc rien programmé pour prendre en charge ce problème ; nous vérifions simplement de manière automatique (et rapide) la convexité de l'enveloppe obtenue, une fois la construction finie.
Remarque 2 (temps de calcul) : pour les polyèdres que nous considérons habituellement, avec quelques centaines de sommets, le temps de construction est de l'ordre de la seconde. Le modèle entier du robot HRP-2 (31 corps) est par exemple obtenu en environ 50 secondes.

Bibliographie

[AbdelMalek 06] Karim Abdel-Malek, Jingzhou Yang, Timothy Marler, Steven Beck, Anith Mathai, Xianlian Zhou, Amos Patrick, Jasbir Arora. – Towards a new generation of virtual humans. *International Journal of Human Factors Modelling and Simulation*, 1(1) :2–39, 2006.

[AbdelMalek 07] Karim Abdel-Malek, Jingzhou Yang, Joo H. Kim, Timothy Marler, Steven Beck, Colby Swan, Laura Frey-Law, Anith Mathai, Chris Murphy, Salam Rahmatallah, Jasbir Arora. – Development of the virtual-human santos$^{\text{tm}}$. – Vincent G. Duffy (édité par), *HCI (12)*, vol. 4561 of *Lecture Notes in Computer Science*, pp. 490–499. Springer, 2007.

[Alami 95] R. Alami, J. P. Laumond, T. Siméon. – Two manipulation planning algorithms. – *WAFR : Proceedings of the workshop on Algorithmic foundations of robotics*, pp. 109–125, Natick, MA, USA, 1995. A. K. Peters, Ltd.

[Barraquand 91] Jérôme Barraquand, Jean-Claude Latombe. – Robot motion planning : a distributed representation approach. *International Journal of Robotic Research*, 10(6) :628–649, 1991.

[Benallegue 09] Mehdi Benallegue, Adrien Escande, Sylvain Miossec, Abderrahmane Kheddar. – Fast $\mathcal{C}^1$ proximity queries using support mapping of sphere-torus-patches bounding volumes. – *IEEE International Conference on Robotics and Automation*, 2009.

[Biswas 04] Arpan Biswas, Vadim Shapiro. – Approximate distance fields with non-vanishing gradients. *Graph. Models*, 66(3) :133–159, 2004.

[Boissonnat 00] Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Lazard. – Motion planning of legged robots. *SIAM J. Computing*, 30 :2000, 2000.

[Bonnans 02] Frédéric Bonnans, Charles Gilbert, Claude Lemaréchal, Claudia A. Sagastizábal. – *Numerical optimization- Theoretical and Practical Aspects.* – Springer, September 2002.

[Bonnans 03] J. Frédéric Bonnans, J. Charles Gilbert, Claude Lemaréchal, Claudia A.Sagastizábal. – *Numerical Optimization.* – Springer, 2003.

[Bourgeot 02] Jean-Matthieu Bourgeot, Nathalie Cislo, Bernard Espiau. – Path-planning and tracking in a 3d complex environment for an anthropo-

mophic bibed robot. – *IEEE/RSJ International Conference on Robots and Intelligent Systems*, 2002.

[Bouyarmane 09] Karim Bouyarmane, Adrien Escande, Florent Lamiraux, Abderrahmane Kheddar. – Collision-free contacts guide planning prior to non-gaited motion planning for humanoid robots. – *IEEE International Conference on Robotics and Automation*, 2009.

[Bretl 05] Tim Bretl. – *Multi-Step Motion Planning : Application to Free-Climbing Robots*. – PhD. Thesis, Stanford University, 2005.

[Broyden 70] C. G. Broyden. – The convergence of a class of double-rank minimization algorithms. *IMA Journal of Applied Mathematics*, 6 :222–231, 1970.

[Chakraborty 06] N. Chakraborty, J. Peng, S. Akella, J. Mitchell. – Proximity queries between convex objects : An interior point approach for implicit surfaces. – *icra*, pp. 1910–1916, Orlando, FL, mai 2006.

[Chestnutt 03] Joel Chestnutt, James Kuffner, Koichi Nishiwaki, Satoshi Kagami. – Planning biped navigation strategies in complex environments. – 2003.

[Chestnutt 05] Joel Chestnutt, Manfred Lau, Kong Man Cheung, James Kuffner, Jessica K Hodgins, Takeo Kanade. – Footstep planning for the honda asimo humanoid. – *IEEE International Conference on Robotics and Automation*, April 2005.

[Chestnutt 07] Joel Chestnutt, Koichi Nishiwaki, James Kuffner, Satoshi Kagami. – An adaptative action model for legged navigation planning. – *IEEE/RSJ International Conference on Humanoid Robots*, 2007.

[Choi 03] Min Gyu Choi, Jehee Lee, Sung Yong Shin. – Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 22(2) :182–203, 2003.

[Choset 97] Howie Choset, Brian Mirtich, Joel Burdick. – Sensor based planning for a planar rod robot : Incremental construction of the planar Rod-HGVG. – *IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3427 – 3434, April 1997.

[Cortés 02] Juan Cortés, Thierry Siméon, Jean-Paul Laumond. – A random loop generator for planning the motions of closed kinematic chains using prm methods. – *IEEE International Conference on Robotics and Automation*, 2002.

[Cortés 03] Juan Cortés. – *Motion Planning Algorithms for General Closed-Chain Mechanisms*. – PhD. Thesis, 2003.

[deBoor 78] C. de Boor. – *A practical guide to splines*. – Applied Mathematical Sciences, New York : Springer, 1978, 1978.

[Ericson 05] Christer Ericson. – *Real-time collision detection*. – Morgan Kaufmann Publishers, 2005.

[Escande 06a] Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec. – Planning support contact-points for humanoid robots and experiments on HRP-2. – *IEEE/RSJ International Conference on Robots and Intelligent Systems*, pp. 2974–2979, Beijing, China, 9-15 October 2006.

[Escande 06b] Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec, Kazuhito Yokoi. – Planning support contact-points for humanoid robots. – *RSJ National Conference*, 2006. – en japonais.

[Escande 08a] Adrien Escande, Paul Evrard, Abderrahmane Kheddar, Nicolas Mansard, Sylvain Miossec, Olivier Stasse, Kazuhito Yokoi. – On-going works around collision avoidance for humanoid robots at the JRL-Japan. – *RSJ National Conference*, Kobe, 2008.

[Escande 08b] Adrien Escande, Paul Evrard, Abderrahmane Kheddar, Nicolas Mansard, Sylvain Miossec, Olivier Stasse, Kazuhito Yokoi. – Recherche en détection de collision en humanoïde au JRL-Japon. – *Journées Nationales de la Robotique Humanoïde*, 2008.

[Escande 08c] Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec, Sylvain Garsault. – Planning support contact-points for acyclic motions and experiments on HRP-2. – *International Symposium on Experimental Robotics*, Athens, Greece, 14-17 July 2008.

[Evrard 08] Paul Evrard, François Keith, Jean-Rémy Chardonnet, Abderrahmane Kheddar. – Framework for haptic interaction with virtual avatars. – *17th IEEE International Symposium on Robot and Human Interactive Communication*, München, Germany, August 1-3 2008.

[Farkas 02] J. Farkas. – Uber der einfachen ungleichungen. *Journal fur die Reine und Angewandte Mathematik*, 124 :1–27, 1902.

[Farrell 07] Kimberly Farrell. – *Kinematic Human Modeling and Simulation Using Optimization-Based Posture Prediction*. – PhD. Thesis, The University of Iowa, 2007.

[Faverjon 87] Bernard Faverjon, Pierre Tournassoud. – A local based approach for path planning of manipulators with a high number of degrees of freedom. – *IEEE International Conference on Robotics and Automation*, 1987.

[Fletcher 70] R. Fletcher. – A new approach to variable metric algorithms. *The Computer Journal*, 13 :317–322, 1970.

[Foissotte 08] Toréa Foissotte, Olivier Stasse, Adrien Escande, Abderrahmane Kheddar. – A next-best-view algorithm for autonomous 3d object modeling by a humanoid robot. – *IEEE/RSJ International Conference on Humanoid Robots*, 2008.

[Fuhrmann 03] A. Fuhrmann, G. Sobottka, C. Gross. – Distance fields for rapid collision detection in physically based modeling. – *International Conference on Computer Graphics and Vision*, 2003.

[Fukuda 95] Komei Fukuda, Alain Prodon. – Double description method revisited. – *Lecture Notes in Computer Science*, vol. 1120, pp. 91–111, 1995.

[Gilbert 85] Elmer G. Gilbert, Daniel W. Johnson. – Distance functions and their application to robot path planning in the presence of obstacles. *IEEE journal of robotics and automation*, 1 :21–30, 1985.

[Goldfarb 70] D. Goldfarb. – A family of variable metric updates derived by variational means. *Mathematics of Computing*, 24 :23–26, 1970.

[Gottschalk 96] S. Gottschalk, M. C. Lin, D. Manocha. – OBBTree : A hierarchical structure for rapid interference detection. – vol. 30, pp. 171–180, 1996.

[Guan 05] Yisheng Guan, Kazuhito Yokoi, Kazuo Tanie. – Feasibility : Can humanoid robots overcome given obstacles ? – *IEEE International Conference on Robotics and Automation*, 2005.

[Harada 06] Kensuke Harada, Kris Hauser, Tim Bretl, Jean-Claude Latombe. – Natural motion generation for humanoid robots. – *IEEE/RSJ International Conference on Robots and Intelligent Systems*, 2006.

[Hauser 05] Kris Hauser, Tim Bretl, Jean-Claude Latombe. – Non-gaited humanoid locomotion planning. – *IEEE/RSJ International Conference on Humanoid Robots*, pp. 7–12, December 5-7 2005.

[Hauser 06] Kris Hauser, Tim Bretl, Kensuke Harada, Jean-Claude Latombe. – Using motion primitives in probabilistic sample-based planning for humanoid robots. – *Workshop on the Algorithmic Foundations of Robotics*, 2006.

[Hauser 07] Kris Hauser, V. Ng-Thow-Hing, H. Gonzalez-Banos. – Multi-modal motion planning for a humanoid robot manipulation task. – *International Symposium on Robotics Research*, 2007.

[Johnson 01] David E. Johnson, Elaine Cohen. – Spatialized normal come hierarchies. – *Symposium on Interactive 3D graphics*, pp. 129–134, New York, NY, USA, 2001. ACM.

[Kalisiak 01] Maciej Kalisiak, Michiel van de Panne. – A grasp-based motion planning algorithm for character animation. *The Journal of Visualization and Computer Animation*, 12(3) :117–129, 2001.

[Kan 89] A. H. G. Rinnooy Kan, G. T. Timmer. – *Global optimization*. – Elsevier North-Holland, Inc., vol. 1, 631–662p., New York, NY, USA, 1989.

[Kanehiro 08] Fumio Kanehiro, Florent Lamiraux, Oussama Kanoun, Eiichi Yoshida, Jean-Paul Laumond. – A local collision avoidance method for non-strictly convex polyhedra. – *Proceedings of Robotics : Science and Systems IV*, Zurich, Switzerland, June 2008.

[Kaneko 04] Kenji Kaneko, Fumio Kanehiro, Shuuji Kajita, Hirohisa Hirukawa, Toshikazu Kawasaki, Masaru Hirata, Kazuhiko Akachi, Takakatsu Isozumi. – Humanoid robot HRP-2. – *IEEE International Conference on Robotics and Automation*, pp. 1083–1090, New Orleans, LA, April 2004.

[Kavraki 96] 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.

[Khalil 02] Wisama Khalil, Etienne Dombre. – *Modeling identification & control of robots*. – Taylor & Francis, 2002.

[Khatib 78] Oussama Khatib, Jean-François Le Maître. – Dynamic control of manipulators operating in a complex environment. – *3rd CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, pp. 267–282, Udine, Italy, September 1978.

[Khatib 80] Oussama Khatib. – *Commande Dynamique dans l'espace opérationnel des robots manipulateurs en présence d'obstacles.* – PhD. Thesis, Ecole Nationale Supérieure de l'aéronautique et de l'espace, 1980.

[Khatib 86] Oussama Khatib. – Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotic Research*, 5 :90–98, Spring 1986.

[Kheddar 08] Abderrahmanne Kheddar, Adrien Escande. – Planning of contact supports for acyclic motion of humanoids and androids : challenges and futue perspectives. – *International Symposium on Robotics*, 2008.

[Klosowski 98] James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, Karel Zikan. – Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1) :21–36, 1998.

[Koga 94] Yoshihito Koga, Koichi Kondo, James Kuffner, Jean-Claude Latombe. – Planning motions with intentions. – *SIGGRAPH '94 : Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 395–408, New York, NY, USA, 1994. ACM.

[Körbel 07] Stefan Körbel. – Optimality conditions and constraint qualifications for nonlinear optimization problems. – Seminar "Convergence Theory for NLP Solvers", October 24 2007.

[Kuffner 98] James J. Kuffner. – Goal-directed navigation for animated characters using real-time path planning and control. – pp. 171–186. Springer-Verlag, 1998.

[Kuffner 02] James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Yasuo Kuniyoshi, Masayuki Inaba, Hirochika Inoue. – Self-collision detection and prevention for humanoid robots. – *IEEE International Conference on Robotics and Automation*, pp. 2265–2270, Washington DC, May 2002.

[Kuffner 03] James Kuffner, K. Nishiwaki, Satoshi Kagami, Y. Kuniyoshi, M. Inaba, H. Inoue. – Online footstep planning for humanoid robots. – *IEEE International Conference on Robotics and Automation*. IEEE, September 2003.

[Kutulakos 99] Kiriakos N. Kutulakos, Steven M. Seitz. – A theory of shape by space carving. – *7th International Conference on Computer Vision*, 1999.

[Latombe 91] Jean-Claude Latombe. – *Robot motion planning.* – Kluwer Academic Publishers, Boston-Dordrecht-London, 1991.

[LaValle 98] Steven M. LaValle. – *Rapidly-exploring random trees : A new tool for path planning.* – Rapport de recherche, 1998.

[LaValle 06] Steven M. LaValle. – *Planning Algorithms.* – Cambrige University Press, 2006.

[Lawrence 97] Craig Lawrence, Jian L. Zhou, André L. Tits. – User's guide for CFSQP version 2.5 : A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints, 1997.

[Lawson 74] H. Blaine Lawson. – Foliations. *Bulletin of the American Mathematical Society*, 80 :369–418, 1974.

[Lee 01] Ji Yeong Lee, Howie Choset. – Sensor-based construction of a retract-like structure for a planar rod robot. *IEEE Transactions on Robotics and Automation*, 17(4) :435–449, August 2001.

[Lefebvre 05] O. Lefebvre, F. Lamiraux, D. Bonnafous. – Fast computation of robot-obstacle interactions in nonholonomic trajectory deformation. – *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 2005.

[Lengagne 07] Sebastien Lengagne, Nacim Ramdani, Philippe Fraisse. – Guaranteed computation of constraints for safe path planning. – *IEEE/RSJ International Conference on Humanoid Robots*, 2007.

[Li 03] Tsai-Yen Li. – Motion planning for humanoid walking in a layered environment. – *IEEE International Conference on Robotics and Automation*, pp. 3421–3427, 2003.

[Lin 91] Ming Chieh Lin, John F. Canny. – Efficient algorithms for incremental distance computation. – *IEEE International Conference on Robotics and Automation*, 1991.

[Mansard 06] Nicolas Mansard. – *Enchaînement de tâches robotiques*. – PhD. Thesis, Université de Rennes 1, mention informatique, December 2006.

[Mansard 07] Nicolas Mansard, François Chaumette. – Task sequencing for sensor-based control. *IEEE Transactions on Robotics*, 23(1) :60–72, February 2007.

[Mardia 00] Kanti V. Mardia, Peter E. Jupp. – *Directional Statistics*. – Wiley, 2000.

[Miossec 06] Sylvain Miossec, Kazuhito Yokoi, Abderrahmane Kheddar. – Development of a software for motion optimization of robots - application to the kick motion of the hrp-2 robot. – 2006.

[Mirtich 98] Brian Mirtich. – V-Clip : fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3) :177–208, 1998.

[Motzkin 53] T. S. Motzkin, H. Raiffa, G. L. Thomson, R. M. Thrall. – The double description method. *Contributions to theory of games*, 2 :51–73, 1953.

[Neff 90] C. A. Neff. – Finding the distance between two circles in three-dimensional space. *IBM J. Res. Dev.*, 34(5) :770–775, 1990.

[Okada 06] Kei Okada, Masayuki Inaba. – A hybrid approach to practical self collision detection system of humanoid robot. – *IEEE/RSJ International Conference on Robots and Intelligent Systems*, pp. 3952–3957, 2006.

[Padberg 99] M. W. Padberg. – *Linear Optimization and Extensions*. – Springer, 2 édition, 165–187p., 1999.

[Patel 05] R.V. Patel, F. Shadpey, F. Ranjbaran, J. Angeles. – A collision-avoidance scheme for redundant manipulators : theory and experiments. *Journal of Robotic Systems*, 22(12) :737–757, 2005.

[Peinado 05] Manuel Peinado, Ronan Boulic, Benoî Le Callennec, Daniel Méziat. – Progressive cartesian inequality constraints for the inverse kinematic control of articulated chains. – *EuroGraphics*, 2005.

[Pettré 03] Julien Pettré, Jean-Paul Laumond, Thierry Siméon. – A 2-stages locomotion planner for digital actors. – *ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 258–264, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Rusaw 01] Shawn Rusaw. – *Sensor-based motion planning in SE(2) and SE(3) via nonsmooth analysis.* – Rapport de recherche, Oxford University Computing Laboratory, 27 September 2001.

[Saidi 06] F. Saidi, O. Stasse, K. Yokoi. – A visual attention framework for search behavior by a humanoid robot. – *IEEE RAS/RSJ Conference on Humanoids Robots*, pp. 346–351, Genova, Italy, December 4-6 2006.

[Samson 91] Claude Samson, Michel Le Borgne, Bernard Espiau. – *Robot Control : the Task Function Approach.* – Clarendon Press, Oxford, United Kingdom, 1991.

[Sentis 05] L. Sentis, O. Khatib. – Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(4) :505–518, 2005.

[Shanno 70] D. F. Shanno. – Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24 :647–656, 1970.

[Shapiro 91] Vadim Shapiro. – *Theory of R-functions and Applications : A Primer.* – Rapport de recherche, Departement of Computer Science, Cornell University, 1991.

[Shapiro 99] Vadim Shapiro, Igor Tsukanov. – Implicit functions with guaranteed differential properties. – pp. 258–269, 1999.

[Siméon 04] Thierry Siméon, Juan Cortès, Jean-Paul Laumond, Anis Sahbani. – Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8) :729–746, July-August 2004.

[Stasse 07] Olivier Stasse, Diane Larlus, Baptiste Lagarde, Adrien Escande, Francois Saidi, Abderrahmane Kheddar, Kazuhito Yokoi, Frederic Jurie. – Towards autonomous object reconstruction for visual search by the humanoid robot hrp-2. – *IEEE RAS/RSJ Conference on Humanoids Robots, Pittsburg, USA, 30 Nov. - 2 Dec.*, p. Oral presentation, 2007.

[Stasse 08] Olivier Stasse, Adrien Escande, Nicolas Mansard, Sylvain Miossec, Paul Evrard, Abderrahmane Kheddar. – Real-time (self)-collision avoidance task on a HRP-2 humanoid robot. – *IEEE International Conference on Robotics and Automation,ICRA-08, Pasadena, California, on May 19-23, 2008*, p. accepted, 2008.

[Stilman 06] Mike Stilman, Koichi Nishiwaki, Satoshi Kagami, James J. Kuffner. – Planning and executing navigation among movable obstacles. – *IEEE/RSJ International Conference on Robots and Intelligent Systems*, 2006.

[Stilman 07] Mike Stilman. – Task constrained motion planning in robot joint space. – *IEEE/RSJ International Conference on Robots and Intelligent Systems*, 2007.

[van den Bergen 04] Gino van den Bergen. – *Collision detection in interactive 3D environments*. – Morgan Kaufmann Publishers, 2004.

[Verrelst 06] Bjorn Verrelst, Olivier Stasse, Kazuhito Yokoi, Bram Vanderborght. – Dynamically stepping over obstacles by the humanoid robot hrp-2. – *IEEE RAS/RSJ Conference on Humanoids Robots, Genova, Italy*, pp. 117–123, December 4-6 2006. – Oral Presentation.

[Vranek 02] David Vranek. – Fast and accurate circle-circle and circle-line 3d distance computation. *J. Graph. Tools*, 7(1) :23–32, 2002.

[Vukobratović 04] Miomir Vukobratović, Branislav Borovac. – Zero-moment point - thirty five years of its life. *International Journal of Humanoid Robotics*, 1 :157–173, 2004.

[Wächter 06] Andreas Wächter, Lorenz T. Biegler. – On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106 :25–57, 2006.

[Wieber 02] Pierre-Brice Wieber. – On the stability of walking systems. – *The third IARP International Workshop on Humanoid and Human Friendly Robotics*, pp. 53–59, Tsukuba, Japan, 11-12 december 2002. AIST.

[Wieber 06] Pierre-Brice Wieber, Florence Billet, Laurence Boissieux, Roger Pissard-Gibollet. – The humans toolbox, a homogeneous framework for motion capture, analysis and simulation. – *9th Symposium on 3D Analysis of Human Movement*, 2006.

[Wood 94] A. T. A. Wood. – Simulation of the von mises fisher distribution. *Communications in statistics. Simulation and computation*, 23(1) :157–164, 1994.

[Yang 04] Jingzhou Yang, Tim Marler, HyungJoo Kim, J.S. Arora, Karim Abdel-Malek. – Multi-objective optimization for upper body posture prediction. – *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, New York, August 30 - September 1 2004.

[Yoshida 05] E. Yoshida, I. Belousov, C. Esteves, J-P. Laumond. – Humanoid motion planning for dynamic tasks. – *IEEE/RSJ International Conference on Humanoid Robots*, 2005.

[Yoshida 06] E. Yoshida, C. Esteves, T. Sakaguchi, J-P. Laumond, K. Yokoi. – Smooth collision avoidance : Practical issues in dynamic humanoid motion. – *IEEE/RSJ International Conference on Robots and Intelligent Systems*, 2006.

[Zhu 04] Xiangyang Zhu, Han Ding, S. K. Tso. – A pseudodistance function and its applications. *IEEE Transactions on Robotics and Automation*, 20(2) :344–352, April 2004.

Publications

Conférences internationales

[1] Mehdi Benallegue, Adrien Escande, Sylvain Miossec, Abderrahmane Kheddar. – Fast $\mathcal{C}^1$ proximity queries using support mapping of sphere-torus-patches bounding volumes. – *ICRA'09*, Kobe, Japon Mai 2009.

[2] Karim Bouyarmane, Adrien Escande, Florent Lamiraux, Abderrahmane Kheddar. – Collision-free contacts guide planning prior to non-gaited motion planning for humanoid robots. – *ICRA'09*, Kobe, Japon Mai 2009.

[3] Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec. – Planning support contact-points for humanoid robots and experiments on HRP-2. – *IROS'06*, Pékin, Chine, Octobre 2006.

[4] Adrien Escande, Sylvain Miossec, Abderrahmane Kheddar. – Continuous gradient proximity distance for humanoids collision-free optimized postures. – *Humanoids 07*, Pittsburg, USA, Décembre 2007.

[5] Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec, Sylvain Garsault. – Planning support contact-points for acyclic motions and experiments on HRP-2. – *ISER'08*, Athènes, Grèce, Juillet 2008.

[6] Adrien Escande, Abderrahmane Kheddar. – Planning contact supports for acyclic motion with task constraints and experiment on HRP-2. – *SYROCO'09*, Gifu, Japon, Septembre 2009. à paraître.

[7] Adrien Escande, Abderrahmane Kheddar. – Contact Planning for Acyclic Motion with Tasks Constraints. – *IROS'09*, St-Louis, USA, October 2009. à paraître.

[8] Adrien Escande, Abderrahmane Kheddar. – Contact Planning for Acyclic Motion with Task Constraints and Experiment on HRP-2 Humanoid. – *IROS'09*, St-Louis, USA, October 2009. vidéo, à paraître.

[9] Toréa Foissotte, Olivier Stasse, Adrien Escande, Abderrahmane Kheddar. – A next-best-view algorithm for autonomous 3d object modeling by a humanoid robot. – *Humanoid 08*, Daejeon, Korée, Décembre 2008.

[10] Toréa Foissotte, Olivier Stasse, Adrien Escande, Pierre-Brice Wieber, Abderrahmane Kheddar. – A Two-Steps Next-Best-View Algorithm for Autonomous 3D Object Modeling by a Humanoid Robot. – *ICRA'09*, Kobe, Japon Mai 2009.

[11] Abderrahmanne Kheddar, Adrien Escande. – Planning of contact supports for acyclic motion of humanoids and androids : challenges and futue perspectives. – *ISR'08*, Korée, 2008.

[12] Olivier Stasse, Diane Larlus, Baptiste Lagarde, Adrien Escande, Francois Saidi, Abderrahmane Kheddar, Kazuhito Yokoi, Fréderic Jurie. – Towards autonomous object reconstruction for visual search by the humanoid robot HRP-2. – *Humanoids 07*, Pittsburg, USA, Décembre 2007.

[13] Olivier Stasse, Adrien Escande, Nicolas Mansard, Sylvain Miossec, Paul Evrard, Abderrahmane Kheddar. – Real-time (self)-collision avoidance task on a HRP-2 humanoid robot. – *ICRA'08*, Pasadena, Californie, May 2008.

Conférences nationales

[14] Adrien Escande, Abderrahmane Kheddar, Sylvain Miossec, Kazuhito Yokoi. – Planning support contact-points for humanoid robots. – *RSJ'06*, Japon, Septembre 2006. – en japonais.

[15] Adrien Escande, Paul Evrard, Abderrahmane Kheddar, Nicolas Mansard, Sylvain Miossec, Olivier Stasse, Kazuhito Yokoi. – On-going works around collision avoidance for humanoid robots at the JRL-Japan. – *RSJ'08*, Kobe, Japon, Septembre 2008

[16] Adrien Escande, Paul Evrard, Abderrahmane Kheddar, Nicolas Mansard, Sylvain Miossec, Olivier Stasse, Kazuhito Yokoi. – Recherche en détection de collision en humanoïde au JRL-Japon. – *JNRH'08*, Paris, France, Mai 2008.

[17] Adrien Escande, Abderrahmane Kheddar. – Plannification de points d'appui pour humanoïdes. – *JNRH'09*, Nantes, France, Mai 2009.

[18] Toréa Foissotte, Olivier Stasse, Adrien Escande, Abderrahmane Kheddar. – Towards a next-best-view algorithm for autonomous 3d object modeling by a humanoid robot. – *RSJ'08*, Kobe, Japon, Septembre 2008

Planification de points d'appui pour la génération de mouvements acycliques : application aux humanoïdes

Un système tel qu'un humanoïde est sous-actionné et hyper-redondant. Il ne peut se mouvoir qu'à travers les interactions qu'il a avec son environnement, c'est-à-dire des contacts, et se déplace dans un espace de configuration de très grande dimension.

Le besoin de rechercher des contacts pour assurer la locomotion implique que tout objet de l'environnement peut être considéré comme un support potentiel. En cela, on s'éloigne des hypothèses classiques de la planification de mouvement où les objets sont des obstacles à éviter. Ces objets prennent, dans le cadre d'un humanoïde, le double statu d'obstacle et support.

L'objectif de nos travaux est de construire un planificateur de mouvement qui sache prendre en compte cette dualité. Nous proposons une approche qui travaille directement dans l'espace des contacts. Pour cela nous développons tout d'abord des outils qui permettent le passage entre l'espace contacts et celui des configurations : un *générateur de posture* qui permet la projection sur des sous-variétés de l'espace de configuration, et un nouveau type de volumes englobants, nommé *STP-BV*, donnant la possibilité d'inclure l'évitement de collision dans la génération de posture. Nous détaillons ensuite l'architecture générale et les divers modules de notre *planificateur de conctacts* avant d'en montrer les résultats en simulation puis sur un robot HRP-2.

Mots-clefs : Robotique, Robot humanoïde, Planification de mouvement, Génération de posture, Contact, Distance de proximité à gradient continu

Contact points planning for acyclic motions generation : application to humanoid robots

Systems like humanoid robots are both underactuated and hyper-redundant. They can move only through their interactions with the environment, i.e. contacts, and have high-dimensional configuration spaces.

The need to look for contacts in order to ensure the locomotion implies that any object in the environment can be considered as a possible support. This conflicts with the classical hypothesis in motion planning that considers the objects as obstacles the robot needs to avoid. In the humanoid robot context, objects have a double status : obstacle and support.

In this document, we aim at building a motion planner that can handle this double property. We propose a method which works directly in the contact space.
We first introduce tools to link the contact space and the configuration space : a *posture generator*, allowing to project onto submanifolds of the configuration space, and a new kind of bounding volume, called *STP-BV*, that can be used to introduce collision avoidance constraints within the posture generation. We then describes the global architecture and the various modules of our *contact planner*, and conclude by showing the result obtained in simulation and on a real robot HRP-2.

Keywords : Robotics, Humanoid robot, Motion planning, Posture Generation, Contact, Proximity distance with continuous gradient