Planification de tâches de manipulation pour robots parallèles à câbles

Diane Bury

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *17/12/2020* **par :**

Diane BURY

Planification de tâches de manipulation pour robots parallèles à câbles

---

JURY

Philippe SOUÈRES	Directeur de Recherche	Président du Jury
Hélène CHANAL	Maître de Conférences	Rapporteure
Jean-Pierre MERLET	Directeur de Recherche	Rapporteur
Delphine KELLER	Ingénieure R&D	Examinatrice
Florent LAMIRAUX	Directeur de Recherche	Directeur de Thèse
Marc GOUTTEFARDE	Directeur de Recherche	Co-Directeur de Thèse

---

École doctorale et spécialité :
EDSYS : Robotique 4200046
Unité de Recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes (UPR 8001)
Directeur(s) de Thèse :
Florent LAMIRAUX et *Marc GOUTTEFARDE*
Rapporteurs :
Hélène CHANAL et *Jean-Pierre MERLET*

To learn which questions
are unanswerable,
and not to answer them:
this skill is most needful
in times of stress and darkness.

Ursula K. Le Guin
(The Left Hand of Darkness)

Remerciements

Je tiens tout d'abord à remercier chaleureusement Florent Lamiraux, mon Directeur de Thèse, qui a d'abord été mon directeur de stage et qui m'a permis de poursuivre ma passion de la robotique avec cette thèse. J'ai beaucoup apprécié travailler avec lui et je tiens à saluer sa disponibilité, sa bienveillance et la confiance qu'il m'a accordée dans mes travaux.

Merci également à Marc Gouttefarde, mon co-Directeur, qui a toujours été disponible quand j'en avais besoin et qui a su me guider dans le domaine des robots à câbles que je découvrais au début de ma thèse.

Merci à Hélène Chanal et Jean-Pierre Merlet d'avoir accepté d'être rapporteur.e.s et membres de Jury de soutenance de ma thèse. Leurs remarques judicieuses m'ont apporté le recul nécessaire pour finaliser ma thèse. Je remercie aussi Philippe Souères et Delphine Keller pour avoir fait partie du Jury de soutenance.

Je tiens à remercier du fond de mon coeur toute l'équipe Gepetto, et tou.t.e.s les étudiant.e.s qui ont fait du LAAS un lieu de vie, d'amitié et de partage (de gâteaux). Merci à tout le monde pour les échanges constructifs comme pour les pauses goûter. Nous avons réussi à forger un esprit d'équipe exceptionnel et je sais que les futur.e.s doctorant.e.s seront toujours entre de bonnes mains. Un grand merci tout particulier au B-181 étendu, le meilleur bureau du monde.

Merci à mes parents, à toute ma famille et merci à tou.te.s mes ami.e.s. Enfin, merci à Paillasson pour sa présence et ses ronrons, et merci à mon partenaire Valentin d'avoir été là avec moi tout du long.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction

Sommaire

La robotique a révolutionné le monde industriel dès 1954 avec le robot Unimate, mis en ligne de production en 1961 afin de souder des pièces sur des automobiles. Depuis, la robotique n'a cessé d'évoluer et de se diversifier. Aujourd'hui, en 2020, plus de 1.6 millions de robots industriels sont utilisés dans le monde, et les innovations continuent à affluer, parallèlement à l'explosion des capacités de calculs informatiques. Les robots deviennent de plus en plus puissants, performants, et autonomes. À côté des bras robotiques, des robots cartésiens ou des robots delta bien connus, les robots parallèles à câbles font preuve d'un fort potentiel émergent en industrie.

Robots parallèles à câbles

Les robots parallèles à câbles (RPCs), sont typiquement constitués d'une base — généralement fixe —, d'une plateforme mobile, et de câbles reliant la plateforme à la base. Les câbles sont enroulés sur des tambours grâce à des treuils et des moteurs attachés la base. Les RPCs utilisent les câbles flexibles comme actionneurs pour faire bouger la plateforme mobile.

Les RPCs présentent l'intérêt d'un grand espace de travail comparativement à des robots manipulateurs classiques. Cela mène à de nombreuses applications pour cette technologie en plein essor, que ce soit dans la logistique, la construction ou même les loisirs. En contrepartie, l'actionnement par câbles induit des difficultés spécifiques, notamment le maintien en tension des câbles et la redondance d'actionnement, la gestion des collisions entre les câbles, la gestion des collisions entre les câbles et l'environnement, ou encore la localisation de la charge utile dans son environnement.

Logiciel Humanoid Path Planner (HPP)

Le plateforme logicielle Humanoid Path Planner (HPP) est un logiciel open-source, diffusé sous licence LGPL, développé par l'équipe Gepetto au Laboratoire LAAS-CNRS. Le LAAS-CNRS a une longue expertise dans le domaine de la génération de mouvements pour divers types de systèmes, des robots à roues aux robots humanoïdes. Le logiciel HPP permet de planifier des mouvements pour des systèmes articulés dans des environnements encombrés par des obstacles. Une extension de cette plateforme logicielle permet de définir et de résoudre des problèmes de planification de mouvements de manipulation, c'est-à-dire des problèmes dans lesquels un robot manipule des objets.

Le logiciel HPP est principalement développé en C++. Des bindings python existent pour l'écriture de scripts, ce qui permet une utilisation simplifiée ainsi que le prototypage facilité de nouveaux algorithmes. Une interface graphique (figure 1) accompagne le logiciel et permet la visualisation des robots. Plusieurs plugins permettent d'utiliser graphiquement les fonctionnalités les plus utiles. La planification d'un mouvement peut ainsi se faire exclusivement via l'interface graphique.

La plateforme HPP contient pour l'heure les planificateur de mouvements suivants :

- RRT [8]
- Bi-RRT et Bi-RRT* [10]
- Visibility-PRM [11]
- k-PRM* [12]

HPP comporte aussi différentes méthodes d'optimisation telles que la méthode random shortcut [13] et la méthode basée gradient de [14]

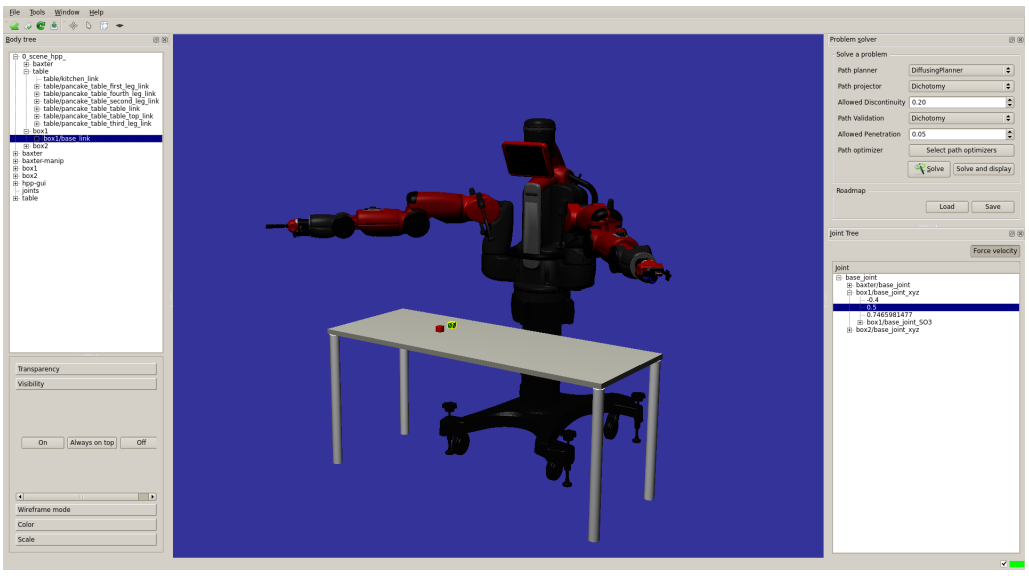


Figure 1 – L'interface graphique d'HPP avec le robot Baxter

Robot CoGiRo

Le robot CoGiRo est un robot parallèle à câbles construit par Tecnalia et le LIRMM dans le cadre du projet ANR CoGiRo, « Control of Giant Robots » [15], débuté en 2010.

Figure 2 – Le robot CoGiRo (LIRMM-Tecnalia)

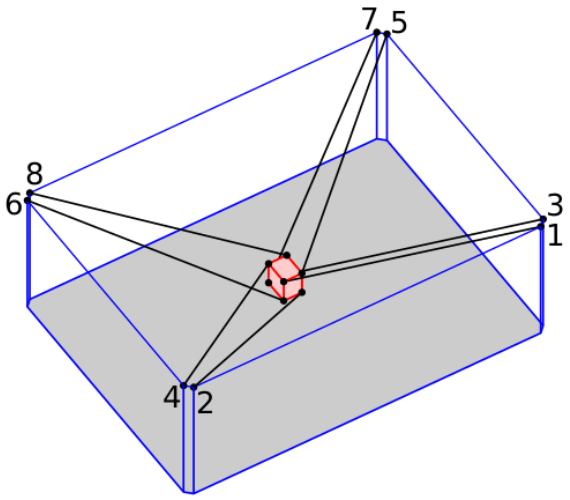


Figure 3 – Schéma du robot CoGiRo (LIRMM-Tecnalia) [1]

CoGiRo est un robot de dimensions 15.24 m × 11.24 m × 5.93 m (L × l × h). Le robot possède une plateforme à 6 degrés de libertés reliée par 8 câbles à sa structure fixe. Il possède donc 2 degrés de redondance d'actionnement. La plateforme peut porter jusqu'à 500 kg de charge utile [16]. Le robot CoGiRo a été construit pour étudier les robots parallèles à câbles de très grande taille. De tels robots ont des applications nombreuses et utiles mais sont rarement construits et étudiés. CoGiRo a été construit de manière à optimiser son espace de travail. Il sert de démonstrateur expérimental pour le développement de nouvelles méthodes de commande, de planification et d'études des grands robots parallèles à câbles. Les

secteurs de la construction, du génie civil et de l'aéronautique pourraient notamment bénéficier des avancées dans le domaine de la robotisation de tâches de grandes dimensions.

Figure 4 – La plateforme mobile de CoGiRo dotée d'une fourche de grue (LIRMM-Tecnalia)

Objectifs et contributions de la thèse

Cette thèse est une thèse CIFRE réalisée au sein de l'entreprise Tecnalia, et considère la problématique de planification de mouvements de manipulation pour RPCs. Notamment, un des objectifs de cette thèse est d'étendre la plateforme logicielle HPP afin d'y inclure les RPCs, qui ne s'assimilent pas aux robots classiques à cause de leurs câbles déformables. La plateforme HPP contient des structures de données et des algorithmes qui permettent de planifier des mouvements de manipulation. Ces algorithmes de planification nécessitent des méthodes de validation de chemins. Deux types de méthodes de validation de chemins sont différenciées : les méthodes de validations discrétisées, qui reposent sur un échantillonnage des trajectoires afin de les valider, et les méthodes de validations continues, qui garantissent la validité de manière continue le long du chemin. Les RPCs présentent des spécificités par rapport aux robots classiques. Premièrement, la présence des câbles entraîne des collisions potentielles supplémentaires : les paires câble-câble, câble-plateforme et câble-environnement doivent être prises en compte pour l'évitement de collisions. Deuxièmement, chaque câble est soumis à des contraintes de tension. En effet, chaque câble peut *tirer* la plateforme mobile mais ne peut pas la *pousser*, ce qui entraîne une contrainte de positivité de la tension. De plus, si la tension d'un câble dépasse une certaine limite, le câble risque de se déformer ou de casser. Ces deux

conditions font que les méthodes de validation de chemins classiques ne fonctionne pas pour les RPCs.

Cette thèse s'est attachée à développer une méthode de validation continue globale pour RPCs. Dans les travaux de cette thèse, les câbles sont supposées *idéaux*, c'est-à-dire sans masse et sans élasticité. Cette hypothèse permet de simplifier grandement les calculs tout en restant relativement réaliste [17].

Dans un premier temps, une méthode de validation continue générale, non spécifique aux RPCs, est formulée. Cette méthode permet de valider un chemin continûment vis-à-vis d'une liste de critères. Elle repose sur la validation successives d'intervalles du chemin. Les critères de validation peuvent être variés mais doivent respecter certaines conditions pour permettre à l'algorithme de fonctionner. Chaque critère donne lieu à un ou plusieurs *éléments de validation* qui doivent permettre, pour un paramètre donné le long d'un chemin, de déterminer un intervalle valide pour ce critère autour de ce paramètre.

Dans un deuxième temps, les éléments de validation pour les collisions impliquant les câbles sont détaillés. Chaque paire câble-câble, câble-plateforme mobile, câble-environnement donne lieu à un élément de validation. Pour chaque paire, calculer un intervalle valide autour d'un paramètre revient à calculer un minorant de la distance entre les paires de corps et un majorant de la vitesse relative d'un corps par rapport à l'autre.

Figure 5 – Scénario de peinture de fuselage par CoGiRo dans le cadre du projet ANR DexterWide (LIRMM-Tecnalia) [2]

Dans un troisième temps, les contraintes des tensions des câbles sont étudiées. Les éléments de validation des tensions sont créés à partir de la méthode de décalage d'hyperplans [18], qui prend en compte les limites minimales et maximales sur les tensions des câbles afin de déterminer la validité d'une configuration. Chaque élément ne représente pas un câble comme on pourrait le penser intuitivement, mais représente une condition de validité du torseur des efforts requis à la configuration donnée. En considérant cette condition de validité comme une condition de positivité d'une distance à un obstacle, on peut calculer un intervalle valide avec un minorant de la distance et un majorant de la vitesse relative comme pour le cas des collisions.

Enfin, la méthode de validation continue globale pour RPC est utilisée pour planifier des mouvements de manipulation pour le robot CoGiRo. Ces mouvements de manipulation représentent les tâches industrielles souhaitables d'un RPC comme

CoGiRo, notamment le déplacement d'objets comme des palettes. Pour chaque scénario, des graphes de contraintes sont créés afin de représenter les différents états et transitions de manipulation. On montre également comment le principe de manipulation et le graphe des contraintes peuvent être utilisés pour générer des mouvements complexes (Fig. 5) qui ne relèvent pas de la manipulation.

Présentation des chapitres

Le Chapitre 1 de cette thèse présente l'état de l'art dans le domaine des RPCs et de la planification de mouvements et de manipulation.

Le Chapitre 2 formule la méthode de validation continue générale. Il détaille l'algorithme de validation continue et introduit le concept d'élément de validation. L'architecture logicielle impliquée pour la validation continue dans la plateforme logicielle HPP est également brièvement présentée.

Le Chapitre 3 présente la détection de collision statique pour RPCs ainsi que les éléments de validation liés aux collisions impliquant des câbles pour la validation continue.

Le Chapitre 4 détaille les contraintes des tensions des câbles d'un RPC, et explique la création des éléments de validation vis-à-vis des tensions des câbles, à partir de la formulation de l'ensemble des torseurs disponibles donnée par la méthode de décalage d'hyperplans.

Le Chapitre 5 décrit les algorithmes de planification de mouvements utilisés, ainsi que des concepts nécessaires à la planification de manipulation comme le graphe des contraintes. Ce chapitre met en avant le problème de composantes connexes qui survient lors de la planification de mouvements pour RPC.

Le Chapitre 6 expose les scénarios de tests considérés et les résultats expérimentaux obtenus sur le robot CoGiRo.

Enfin, le dernier chapitre présente les conclusions de ces travaux et des pistes de travaux futurs.

Publications

Diane Bury, Jean-Baptiste Izard, Marc Gouttefarde and Florent Lamiraux. Continuous Collision Detection for a Robotic Arm Mounted on a Cable-Driven Parallel Robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2019.

Diane Bury, Jean-Baptiste Izard, Marc Gouttefarde and Florent Lamiraux. Continuous Tension Validation for Cable-Driven Parallel Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2020.

Chapitre 1

État de l'art

Sommaire

1.1 Robots parallèles à câbles

Les premiers robots parallèles à câbles (RPCs) ont fait leur apparition dans les années 1980. Parmi les plus connus, le robot SkyCam [19] (Figure 1.1), utilisé pour la première fois en 1984, est un robot de grande envergure, suspendu au-dessus des stades pour filmer les matchs de sport. Grâce à ses câbles longs, SkyCam peut couvrir de grands espaces tout en gardant un poids et un encombrement réduit, en plus de pouvoir atteindre des vitesses importantes et de fortes accélérations. En 1989, le National Institute of Standards and Technology (NIST) lance le projet RoboCrane [20, 21] et conçoit un modèle de RPC simple et versatile (Figure 1.2). L'intérêt et les avantages des RPCs sont évidents, mais les progrès restent lents. Parmi les causes, les capacités de calculs limitées des ordinateurs de l'époque et le coût de construction et d'installation de tels robots. Dans le milieu de années 1990, les progrès s'accélèrent et les robots à câbles deviennent populaires. Aujourd'hui, les RPCs sont activement étudiés et leurs champs d'applications sont nombreux et variés.

Figure 1.1 – Le RPC SkyCam

1.1.1 Principe des RPCs

Un RPC est un robot constitué d'une plateforme mobile, de câbles, et d'une base. La plateforme mobile évolue dans un certain espace, dont la dimension définit le nombre de degrés de liberté du robot. Un RPC à 6 degrés de liberté a une plateforme mobile pouvant se déplacer en translation et en rotation dans l'espace (Figure 1.3). Un RPC planaire possède 2 ou 3 degrés de liberté, sa plateforme évolue dans un plan sans ou avec rotation (Figure 1.4).

Les câbles reliant la plateforme mobile à la base d'un RPC ne travaillent qu'en tension, c'est-à-dire qu'ils ne peuvent que tirer la plateforme et pas la pousser. Cela induit une contrainte de positivité de la tension de chaque câble. De plus, la tension de chaque câble doit rester inférieure à un valeur limite afin d'éviter au câble de

(a) Modèle de NIST RoboCrane (b) NIST RoboCrane, modèle de 2m

Figure 1.2 – Robot NIST RoboCrane

s'allonger excessivement ou de rompre. Ces contraintes sur la tension des câbles constituent un des sujets majeurs dans l'étude des RPCs.

Différents espaces de travail peuvent être définis, comme l'espace de travail de fermeture d'effort [22, 23].

Espace de travail de fermeture d'effort (WCW)

L'espace de travail de fermeture d'effort (WCW, de l'anglais *Wrench-Closure Workspace*) d'un RPC est l'ensemble des poses de la plateforme mobile pour lesquelles n'importe quel torseur d'efforts peut être généré à la plateforme en tendant les câbles. Le WCW est l'ensemble des poses pour lesquelles la plateforme mobile est entièrement contrainte par les câbles.

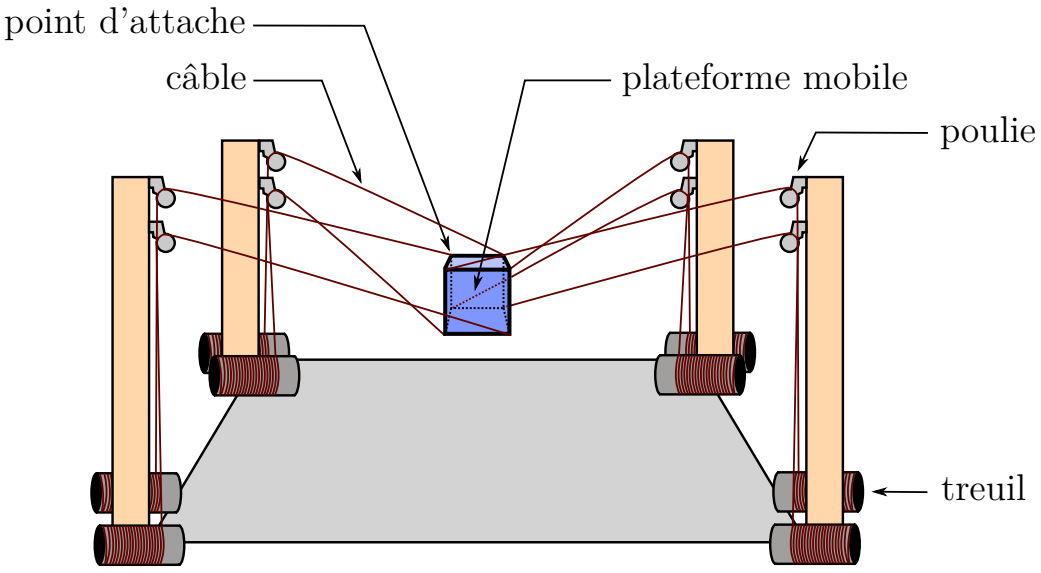


Figure 1.3 – Schéma d'un RPC à 6 degrés de liberté et à 8 câbles

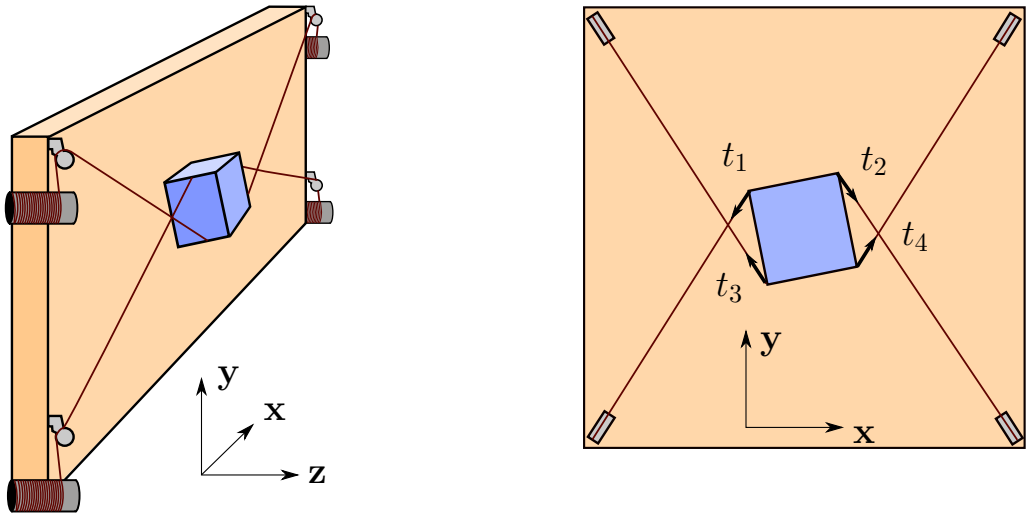


Figure 1.4 – Schéma d'un RPC à 3 degrés de liberté (x,y, rotation) et 4 câbles, vue en perspective (à gauche) et de face (à droite).

L'espace de travail de fermeture d'effort dépend uniquement de la géométrie du RPC, c'est-à-dire des positions des points de sortie des câbles sur la base et des positions des points d'attache sur la plateforme mobile. Une condition nécessaire (mais non suffisante) pour l'existence du WCW est que le nombre de câbles soit strictement supérieur au nombre de degrés de liberté de la plateforme. Quand le WCW d'un RPC existe et recouvre une grande portion de l'espace de travail désiré, le RPC est dit **complètement contraint**. Lorsque que le nombre de câbles d'un RPC est supérieur à son nombre de dégrés de liberté, le RPC est dit **à actionnement redondant**. Les RPCs à actionnement redondants, comme le robot CoGiRo, sont l'objet d'étude de cette thèse.

Généralement, les positions des points de référence des câbles — les points d'attache des câbles sur la plateforme et les points de sortie des câbles de la base — sont supposés fixes. Ces positions sont déterminées lors de la conception du RPC, de manière à optimiser l'espace de travail. Cependant, dans certaines situations, il peut être nécessaire de reconfigurer ces points de référence des câbles. Les robots parallèles à câbles reconfigurables peuvent être reconfigurés selon un ensemble discret de reconfigurations, en se donnant un ensemble de points de sortie et de points d'attache sur la plateforme possibles [24, 25]. Pour chaque tâche, selon l'environnement, une configuration optimale est calculée et choisie. D'autres RPCs reconfigurables possèdent un ensemble continu de reconfigurations possibles, par exemple grâce à un élément mobile de la base qui permet de varier la position du point de sortie d'un câble. Cet élément mobile peut être par exemple un rail [26, 27] ou un drone volant [28, 29, 30, 31]. La reconfigurabilité de ces RPCs apporte notamment une solution au problème de collisions entre les câbles [32].

1.1.2 Applications des robots à câbles

Les robots à câbles ont de nombreuses applications dans des domaines d'activités très variés. La liste suivante présente quelques exemples.

- Télescopes de grande envergure : construire un télescope avec des câbles permet d'atteindre des dimensions record. Le radio-télescope à câbles d'Arecibo [33] (Figure 1.5) est une structure à câbles et était jusqu'en 2016 le plus grand radiotélescope du monde avec un diamètre de 305m pour son antenne principal. Son record a été dépassé par le radiotélescope chinois FAST, qui utilise des câbles pour maintenir ainsi que déplacer sa cabine focale.

Figure 1.5 – Le radio-télescope géant d'Arecibo

- Rééducation neurologique : la robotique joue un rôle crucial dans la médecine et notamment la rééducation et l'assistance pour patients accidentés, et les robots à câbles ne font pas exception. Des exosquelettes actionnés par câbles ont été étudiés pour aider à la rédadaptation neurologique des membres supérieurs [3, 34] (Figure 1.6) et des membres inférieurs [35]. Des RPCs ont été conçus pour interagir via leur organe terminal avec les humains, qui doivent attraper et appliquer un effort sur l'organe terminal du robot [36, 37].

Figure 1.6 – L'exosquelette à câbles Carex [3]

- Simulation de mouvements : les RPCs peuvent être utilisés en tant que simulateurs de mouvements. Les mouvements de la plateforme mobile sont utilisés pour donner à l'utilisateur la sensation d'être dans un environnement différent. Un dispositif de réalité virtuelle peut être ajouté pour compléter l'expérience. Le CableRobot Simulator [38] (Figure 1.7) développé au sein du Max Planck Institute possède une plateforme assez grande (2.6 m de diamètre) pour contenir le matériel embarqué et deux sièges. Ce robot permet de simuler des scénario de conduite de voiture ou de pilotage d'avion, et permet aussi d'effectuer des travaux de recherches en perception.

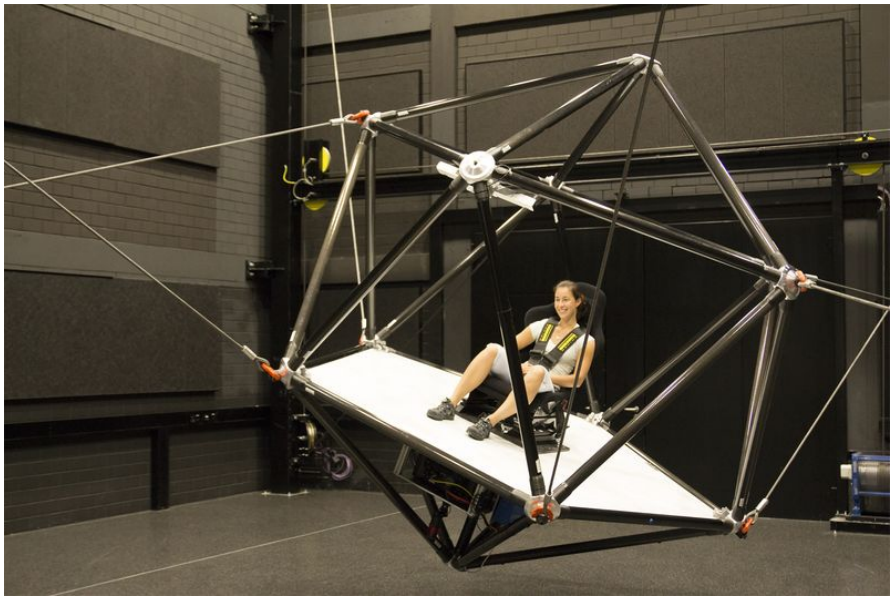


Figure 1.7 – La plateforme du CableRobot Simulator développé par la Max Plack Society

- Robotique industrielle : les RPCs commencent à être étudiés pour être utilisés dans l'industrie, que ce soit pour des tâches de « pick-and-place » [39, 40, 41] — consistant à attraper et déplacer rapidement des objets —, les tâches de manipulation et d'assemblage [42] ou les tâches de logistique [43, 44]. Les RPCs peuvent également être utilisés afin de calibrer d'autres robots, par exemple des bras manipulateurs [4] (Figure 1.8).

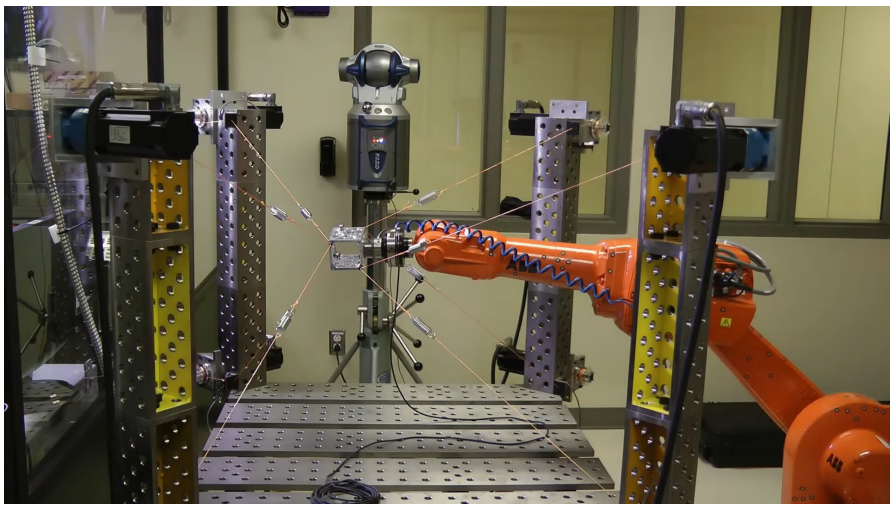
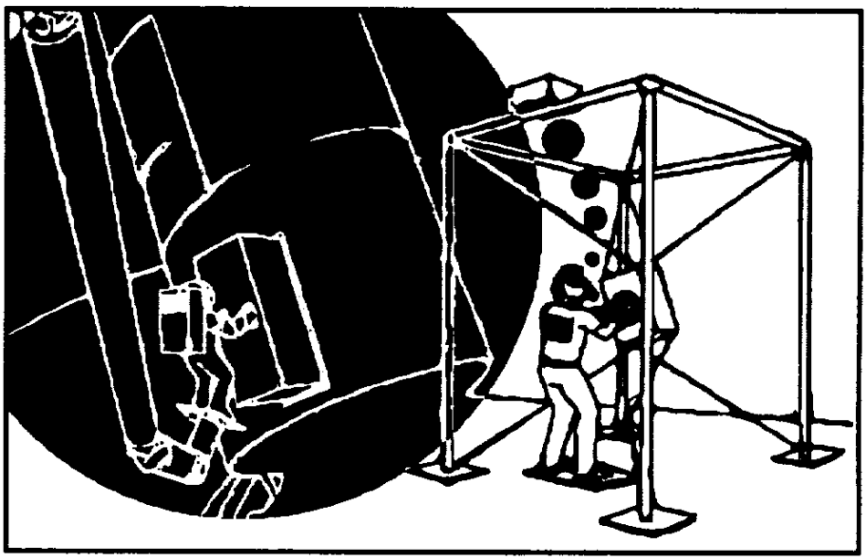


Figure 1.8 – Le RPC CabOLS calibrant automatiquement un bras robotique [4]

- L'impression 3D : l'essort de l'impression 3D dans les années 2010 a mené à des nombreuses avancées et innovations dans le domaine, notamment l'utilisation de RPCs pour actionner l'extrudeuse d'une imprimante 3D [45]. Les RPCs, grâce à leurs larges espaces de travail, permettent également de construire des imprimantes 3D de grandes tailles [46]. Par exemple, le robot CoGiRo [15] doté d'une extrudeuse devient une imprimante 3D capable d'imprimer des murs et des parties de bâtiments [47] (Figure 1.9), laissant envisager que dans le futur, des maisons entières pourront être construites avec un tel robot.

Figure 1.9 – Le RPC CoGiRo imprimant un mur [5]

- Applications spatiales : le RPC Charlotte [48] a été développé par la NASA avec de nombreux objectifs. Entre autre, une des applications envisagée de Charlotte est d'assister les astronautes lors de leurs sorties extra-véhiculaires et de servir au déplacement et à l'installation de matériel sur une base lunaire. Actuellement, Charlotte est utilisé en tant qu'interface à retour d'efforts pour simuler les entraînement des sorties extra-véhiculaires (Figure 1.10). D'autres RPCs peuvent également jouer le rôle d'interface à retour d'efforts, comme le RPC Inca développé par Haption [49].



(a) Dessin de 1994 d'un astronaute s'entraînant avec Charlotte [48]

(b) Un astronaute de la NASA s'entraînant avec Charlotte en 2014 [50]

Figure 1.10 – Le RPC Charlotte de la NASA

- Déplacement et positionnement de lourdes charges : l'utilisation des câbles rend les RPCs particulièrement adaptés pour déplacer des objets lourds. Plusieurs RPCs ont été conçus dans cette optique, notamment le RPC CoGiRo étudié dans cette thèse, le RPC Marionet Crane [51] conçu pour les opérations de sauvetage, ou le RPC Tensile Truss du projet NIST [6] (Fig-

ure 1.11) prévu pour être utilisé à la centrale de Tchernobyl pour nettoyer les débris de la catastrophe nucléaire.

Figure 1.11 – Le RPC Tensile Truss du NIST dans la centrale nucléaire de Tchernobyl [6]

1.1.3 Collision au sein des RPCs

Les RPCs bénéficient de grands espaces de travail, mais souffrent de restrictions dues aux interférences causées par les câbles. Les câbles peuvent entrer en collision entre eux, avec la plateforme mobile et sa charge, ou avec l'environnement. Il est nécessaire de pouvoir générer des trajectoires sans collision, que ce soit en déterminant un espace de travail sans collision ou en détectant les collisions le long d'une trajectoire.

Une détermination approximative du volume balayé par un câble lorsque la plateforme mobile d'un RPC se déplace dans un espace de travail prescrit est examinée dans [52] et la détermination géométrique des lieux de collision câble-cylindre dans l'espace de travail d'un RPC est traitée dans [53]. Dans ce dernier, le cylindre est un objet fixe situé à l'intérieur de l'espace de travail du RPC. En outre, dans [54], l'espace de travail d'impression sans collision est calculé pour les RPC complètement contraints destinés à imprimer des objets de grande dimension dans une séquence de couches horizontales. Alors que toutes ces méthodes déterminent divers types de lieux de collision de câbles dans un espace de travail prescrit, d'autres travaux antérieurs abordent la question de la vérification des collisions de câbles le long d'une trajectoire prescrite de plateforme mobile de RPC [55, 56, 57], ce qui est également l'objet du présent document. Dans [57], un planificateur de trajectoires sans collision dans un espace discrétisé appliqué aux RPC est présenté. Des méthodes plus avancées fondées sur l'analyse des intervalles, qui peuvent tenir compte des

incertitudes des paramètres et des erreurs dans le calcul numérique, sont présentées dans [55, 56].

La détermination des lieux de collisions de câbles dans un espace de travail prescrit est présentée dans [58, 59] dans le cas d'un espace de travail à orientation constante et dans [55, 60, 61] dans le cas d'un espace de travail 6D. Des approches heuristiques rapides sont proposées dans [61] tandis que des calculs certifiés basés sur l'analyse des intervalles sont introduits dans [55, 60].

1.1.4 Analyse de l'espace de travail

La question de l'analyse de l'espace de travail des RPCs a été étudiée souvent avec l'hypothèse d'une orientation constante de la plateforme et un modèle simple de câble en ligne droite [62, 63, 64], parfois avec des modèles de câble plus complexes tenant compte de l'élasticité ou de l'affaissement [65]. L'espace de travail peut être défini par les équations de ses limites [66, 23] ou en échantillonnant l'espace de travail global et en testant un grand nombre de positions et d'orientations de la plateforme mobile [67].

Lors de la conception d'un RPC, faire varier la forme du robot, le nombre de câble et les positions de leurs points d'attache et de sortie permet d'optimiser l'espace de travail en fonction de l'utilisation désirée. La conception d'un RPC peut être optimisée de manière à ce que l'espace de travail soit adapté à une tâche donnée. Dans [68], la norme du vecteur des tensions dans les câbles est minimisée le long d'une trajectoire souhaitée pour sélectionner une configuration de câble adaptée à la tâche. Les collisions entre câbles et entre des câbles et la plateforme mobile peuvent également être minimisées. Dans [1], une analyse est menée afin de sélectionner la géométrie optimale d'un RPC suspendu à redondance d'actionnement, ce qui a mené à la conception du robot CoGiRo.

1.1.5 Commande des RPCs

L'utilisation de câbles entraîne des difficultés de commande spécifiques aux RPCs par rapport aux systèmes articulés classiques. À cause de la flexibilité et de l'affaissement des câbles, la position et l'orientation de la plateforme n'est pas facilement calculable en fonction de la longueur des câbles. De plus, la commande des RPCs doit prendre en compte les contraintes de tensions des câbles. Cela rend nécessaire d'adapter et de modifier les méthodes de commandes classiques pour les RPCs.

Citons quelques exemples de commande pour RPCs parmi les nombreux travaux existants. Dans [69], un contrôleur robuste avec commande en mode glissant et adaptation selon la limite supérieure des incertitudes est proposé pour un RPC complétement contraint. Ce contrôleur ne requiert pas de mesurer la position de l'organe terminal et permet de garder tous les câbles du RPC sous tension. Un contrôleur hybride position/force est présenté dans [70] et appliqué à une réplique minitature du radio-télescope géant FAST. Certaines méthodes abordent le problème des frottements [71] et d'autres celui de la réduction des vibrations [72] tout en garantissant les contraintes de positivité des tensions des câbles. Dans [73], une approche pour la commande de RPC à câbles très flexibles est présentée et testée sur un RPC planaire. La commande en temps réel a été étudié pour les RPCs non-complétement contraints [74] ainsi que pour les RPCs complétement contraints [75,

76]. [77] propose une méthode de commande capable de calculer les distributions des tensions des câbles le long d'une trajectoire pour le cas particulier des RPCs à n degrés de liberté et $n+2$ câbles. Dans [78], les performances d'un RPC à 6 degrés de liberté et 8 câbles sont analysées de manière statique et dynamique, afin d'évaluer la qualité du suivi de trajectoire selon les caractéristiques des câbles, notamment leur matière (nylon, acier).

Les travaux de cette thèse ne s'attachent pas au problème de commande des RPCs, que l'on considère comme une étape postérieure à l'étape de planification de mouvement.

1.2 Planification de mouvements et manipulation

1.2.1 Planification de mouvements

La planification de mouvement consiste à trouver une trajectoire sans collision d'une pose initiale du robot jusqu'à une pose finale donnée. La notion d'espace des configuration (*configuration space*) [79] est essentielle en planification de mouvement.

Espace des configurations

Noté $\mathcal{CS}$, l'espace des configurations est le produit cartésien des intervalles de définition des paramètres des degrés de liberté du robot.

L'espace des configurations peut se diviser en deux domaines :

- $\mathcal{CS}_{\text{obs}}$: le sous-ensemble de $\mathcal{CS}$ des configurations pour lesquelles au moins un corps du robot est en collision avec un autre corps.
- $\mathcal{CS}_{\text{free}}$: le sous-ensemble de $\mathcal{CS}$ des configurations sans collision.

Notons que $\mathcal{CS}_{\text{free}} = \mathcal{CS} \setminus \mathcal{CS}_{\text{obs}}$. Le sous-ensemble $\mathcal{CS}_{\text{free}}$ est un ensemble ouvert.

Une configuration de $\mathcal{CS}$ donne donc la position et l'orientation de chaque corps du robot. Le problème de planification de mouvement revient donc à trouver une courbe continue pour un point dans $\mathcal{CS}$.

Formulation du problème de planification de mouvement

Le problème de planification de mouvement revient, pour une configuration initiale $\mathbf{q}_{\text{init}}$ donnée et une configuration finale $\mathbf{q}_{\text{goal}}$, à trouver un chemin continu $\mathbf{p}(t)$ tel que :

- $\forall t \in [0,1], \mathbf{p}(t) \in \mathcal{CS}_{\text{free}}$
- $\mathbf{p}(0) = \mathbf{q}_{\text{init}}$
- $\mathbf{p}(1) = \mathbf{q}_{\text{goal}}$

Si le problème comporte plusieurs robots et objets, l'espace des configurations est le produit cartésien des espaces des configurations de chaque robot ou objet. On

considère donc l'ensemble des robots et de l'environnement d'un problème comme un système unique.

Pour un robot articulé classique, une configuration comporte les paramètres de chaque articulation. Pour un RPC, il faut prendre en compte les différents degrés de liberté en translation et en rotation de la plateforme mobile. En robotique, les quaternions unitaires sont souvent utilisés afin de décrire l'espace des rotations en dimension trois $SO(3)$ [80]. Cette représentation est plus compacte que les matrices de rotations (9 paramètres) et permet d'éviter le problème singularité de représentation [81] qui peut survenir en utilisant les angles d'Euler. Dans la suite de cette thèse, nous utilisons les quaternions pour décrire les rotations en dimension trois. Pour un corps en translation et rotation libre dans l'espace, 7 valeurs scalaires permettent donc de définir la position de l'orientation du corps : 3 valeurs pour la position selon les axes (x,y,z) et 4 valeurs pour un quaternion unitaire définissant la rotation. Ces 4 valeurs sont liés par la contrainte de norme unitaire du quaternion. Un RPC à 6 degrés de liberté est donc représenté par une configuration à 7 éléments, quel que soit le nombre de câbles. Si le RPC est doté d'un bras robotique, les paramètres articulaires sont notés dans $\mathbf{q}$ à la suite du quaternion décrivant l'orientation de la plateformes. Le vecteur de configuration $\mathbf{q}$ d'un RPC doté d'un bras robotique s'écrit donc de la manière suivante :

$$\mathbf{q} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ \alpha_0 \\ \alpha_1 \\ \vdots \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ b \\ c \end{matrix}}\right\} \text{position de la plateforme} \\ \left.\vphantom{\begin{matrix} a \\ b \\ c \\ d \end{matrix}}\right\} \text{orientation de la plateforme (quaternion)} \\ \left.\vphantom{\begin{matrix} a \\ b \\ c \end{matrix}}\right\} \text{paramètres articulaires du bras robotique} \end{matrix} \tag{1.1}$$

Il faut noter que cette définition de l'espace de configuration pour les RPC est une simplification. En effet, pour chaque pose de la plateforme mobile, il n'existe pas une seule et unique distribution de tensions au sein des câbles. Afin de caractériser entièrement un état du robot, il faudrait inclure cette distribution de tension dans la définition du vecteur de configuration. Dans la suite de cette thèse, nous utilisons cette définition simplifiée de l'espace de configurations.

La vitesse du robot est représentée par un vecteur noté $\mathbf{v}$, composé des paramètres de vitesse linéaire de la plateforme mobile — c'est-à-dire les dérivées des paramètres de position de la plateforme —, des paramètres de vitesse angulaires de la plateforme, et des paramètres de vitesses articulaires des différentes articulations du bras robotique. Il est important de noter que $\mathbf{v}$ n'est pas la dérivée de $\mathbf{q}$, et que les deux vecteurs n'ont pas la même taille. En effet, 4 paramètres servent à décrire l'orientation de la plateforme (un quaternion), tandis que 3 paramètres décrivent la vitesse angulaire de la plateforme (vecteur de vitesse angulaire). Le vecteur de vitesse d'un RPC doté d'un bras robotique s'écrit donc de la manière suivante :

$$\mathbf{v} = \begin{bmatrix} \dot{x_p} \\ \dot{y_p} \\ \dot{z_p} \\ \omega_x \\ \omega_y \\ \omega_z \\ \dot{\alpha_0} \\ \dot{\alpha_1} \\ \vdots \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{vitesse linéaire de la plateforme} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{vitesse angulaire} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{vitesses articulaires du bras robotique} \end{matrix} \tag{1.2}$$

Le concept d'espace de configuration est très utilisé en planification de mouvement. Il existe trois grandes familles de méthodes de planification de mouvement : les méthodes déterministes, les approches probabilistes, et les approches par optimisation.

1.2.1.a Méthodes déterministes

Pour un problème donné et des conditions initiales données, une méthode déterministe calcule toujours la même solution — ce qui n'est pas le cas pour les méthodes probabilistes. Parmi les méthodes les plus connues peuvent être nommées les diagrammes de Voronoi [82], la décomposition cellulaire [83, 84], les champs de potentiel [85, 86], l'algorithme de Canny [87], et les méthodes utilisant des grilles telles A* ou Dijkstra [88].

Les méthodes déterministes garantissent de trouver une solution ou d'indiquer s'il n'en existe aucune. On dit donc que ces méthodes sont *complètes en résolution*. Cependant, elles subissent ce qui est appelé la *malédiction de la dimension*, c'est-à-dire que quand la dimension de l'espace des configurations augmente, les temps de calcul augmentent — souvent de manière exponentielle — et il n'est plus toujours possible d'obtenir une solution dans un temps raisonnable.

1.2.1.b Méthodes probabilistes

Les méthodes probabilistes reposent sur le tirage de configurations aléatoires, et ne permettent ainsi pas d'obtenir la même solution à chaque résolution d'un même problème. Ces méthodes tirent des configurations aléatoires dans l'espace des configurations pour construire un graphe de configurations reliées par des chemins valides. Ce graphe est appelé carte (*roadmap*). Ces méthodes ont la garantie de trouver une solution s'il en existe une, si on leur laisse un temps de calcul infini. En effet, la probabilité de trouver une solution s'il en existe une s'approche de un au fur et à mesure que le nombre de configurations tirées augmente. Les méthodes probabilistes sont donc dites *complètes en probabilité*. Il existe deux types de méthodes probabilistes :

- Les méthodes d'échantillonnage cherchent à construire une carte qui approxime la connectivité de $\mathcal{CS}_{\text{free}}$ autant que possible. Cette carte, une fois construite, est ensuite utilisée pour résoudre différents problèmes de planification dans le même espace des configurations. Pour chaque problème, la configuration initiale et la configuration finale sont ajoutées à la carte déjà calculée et une

trajectoire les reliant est calculée. Le temps de calcul dépend de la carte : plus la carte comporte de configurations et plus la recherche sera lente ; plus la carte représente fidèlement la connectivité de $\mathcal{CS}_{\text{free}}$ et plus les temps de calculs seront courts. La méthode d'échantillonnage la plus connue est la méthode PRM (Probabilistic Roadmaps) [89]. D'autres méthodes améliorent la méthode PRM, comme V-PRM (Visibility PRM) [90] qui identifie les configurations apportant le plus d'information par rapport à la connectivité de $\mathcal{CS}_{\text{free}}$, et ainsi permet d'avoir une carte plus petite. De nombreuses variations de PRM sont étudiées dans [91].

- Les méthodes de diffusion tirent des configurations aléatoires dans $\mathcal{CS}$, sans essayer d'obtenir une carte particulièrement représentative de $\mathcal{CS}_{\text{free}}$, jusqu'à résoudre le problème donné. La carte des configuration est un arbre, constitué initialement de la configuration initiale et éventuellement de la configuration finale, qui est agrandi à chaque itération vers une configuration aléatoire en testant le chemin obtenu vis-à-vis des collisions. La méthode de diffusion la plus connue est la méthode RRT (Rapildly-exploring Random Tree) [8], qui possède de nombreuses variantes comme le BiRRT, RRT* [12] ou RRT-Connect [92].

Les algorithmes probabilistes présentent quelques inconvénients :

- Ils génèrent des trajectoires aléatoires qui peuvent contenir des mouvements inutiles et sont loin d'être optimales. Plusieurs solutions existent : utiliser des méthodes d'optimisation après un algorithme probabiliste pour améliorer la trajectoire ; utiliser une variante d'un algorithme probabiliste qui tend vers la solution optimale comme RRT* ou PRM*[12].
- Quand l'espace des configurations contient des *passages étroits*, peu de configurations aléatoires sont tirées dans ces passages, et de ce fait les temps de calcul des algorithmes probabilistes augmentent.
- Les algorithmes probabilistes sont *complets en probabilité* uniquement et non *complets en résolution*. Si aucune solution n'existe, un tel algorithme n'est pas capable de le détecter et n'arrêtera jamais son exécution.
- Il est impossible d'échantillonner aléatoirement des sous-variétés de volume nul. Cela complique la planification de mouvement des robots humanoïdes par exemple. Une variante de RRT, appelée Constrained-RRT [93, 94], permet de résoudre ce problème.

Les méthodes RRT et PRM sont présentées plus en détail dans la section 5.2.

1.2.1.c Méthodes par optimisation

Les méthodes basées sur l'optimisation formulent le problème sous la forme d'une optimisation de trajectoire. Elles ont besoin en entrée d'une trajectoire initiale, qui n'a pas besoin d'être valide. Elles itèrent ensuite sur cette trajectoire pour la rendre valide tout en optimisant une fonction de coût prédéfinie. Cette fonction de coût effectue généralement un compromis entre l'évitement d'obstacle et l'obtention d'une trajectoire lisse, mais peut également prendre en compte d'autres critères comme la longueur de la trajectoire ou l'énergie nécessaire au mouvement. Ces méthodes ont également besoin de méthodes pour quantifier les distances aux obstacles pour

la détection de collision. La distance entre deux corps qui ne sont pas en collision est positive, et sa valeur est la plus courte distance entre les deux corps. Pour deux corps en collision, la distance entre ces corps est négative et sa valeur correspond à la distance de pénétration des deux corps. La distance de pénétration entre deux corps est la distance de translation minimum requise afin de séparer deux objets en collision.

Zucker et al. [95] ont proposé l'algorithme CHOMP (Covariant Hamiltonian Optimization for Motion Planning). CHOMP itère sur une trajectoire initiale en utilisant des techniques de gradient fonctionnel pour faire diminuer le coût, et utilise l'algorithme de Monte Carlo Hamiltonien afin de diminuer le risque de converger vers un minima local à coût élevé. D'autres méthodes proposent des approches stochastiques. STOMP (Stochastic Trajectory Optimization for Motion Planning) [96] par exemple utilise un paramètre de bruit aléatoire afin de générer de nouvelles trajectoires autour de la trajectoires initiale, puis combine ces trajectoires pour obtenir une trajectoire finale valide et avec un coût plus bas. De cette façon, STOMP ne requiert aucune information de gradient et peut s'utiliser avec des fonctions dont les dérivées ne sont pas disponibles. L'approche stochastique évite à STOMP de converger vers des minima locaux. Une autre méthode stochastique est la méthode PSO [97] (Particle Swarm Optimization), qui est une méthode d'optimisation globale basée sur le principe de population inspirée par le comportement de groupes d'animaux sauvages.

1.2.2 Planification sous contraintes

Pour de nombreux systèmes comme les robots humanoïdes, les systèmes sous-actionnés ou les systèmes en boucle fermée, l'ensemble des configurations réalisables est une sous-variété de $\mathcal{CS}$. La planification de mouvement se fait donc sous contraintes. L'ensemble des configuration réalisables peut s'écrire $\{\mathbf{q} \in \mathcal{CS} \mid \mathbf{f}(\mathbf{q}) = \mathbf{0}\}$, où $\mathbf{f}$ représente les contraintes. Cet ensemble est défini implicitement et peut avoir un volume nul dans $\mathcal{CS}$.

Formulation du problème de planification de mouvement sous contraintes

Soit une configuration initiale $\mathbf{q}_{\text{init}}$ et une configuration finale $\mathbf{q}_{\text{goal}}$, ainsi qu'une contrainte $\mathbf{f} : \mathcal{CS} \to \mathbb{R}^n$ telle qu'une configuration $\mathbf{q} \in \mathcal{CS}$ est réalisable si et seulement si $\mathbf{f}(\mathbf{q}) = \mathbf{0}$. Le problème de planification de mouvement consiste à chercher un chemin continu $\mathbf{p}(t)$ tel que :

- $\mathbf{p} : [0, 1] \to \mathcal{CS}_{\text{free}}$
- $\mathbf{p}(0) = \mathbf{q}_{\text{init}}$
- $\mathbf{p}(1) = \mathbf{q}_{\text{goal}}$
- $\forall t \in [0, 1],\ \mathbf{f}(\mathbf{p}(t)) = \mathbf{0}$

Pour qu'une configuration $\mathbf{q}$ satisfasse les contraintes $\mathbf{f}$, il faut et il suffit que $\mathbf{f}(\mathbf{q}) = \mathbf{0}$. Pour des raisons pratiques, on considère généralement une marge de tolérance ϵ de la fonction contrainte $\mathbf{f}$. Dans ce cas, une configuration $\mathbf{q}$ est réalisable

si et seulement si $\|\mathbf{f}(\mathbf{q})\| \leq \epsilon$.

Plusieurs méthodes permettent de générer une configuration qui satisfait la contrainte, comme la méthode Randomized Gradient Descent (RGD) [98] et la méthode de Newton-Raphson [99]. Ces deux méthodes itèrent sur la configuration initiale $\mathbf{q}$ de manière à diminuer l'erreur $\|\mathbf{f}(\mathbf{q})\|$. La méthode RGB tire aléatoirement des configurations autour de $\mathbf{q}$ afin de trouver une configuration qui satisfait davantage la contrainte. La méthode de Newton-Raphson utilise la matrice Jacobienne de la fonction de contrainte pour améliorer $\mathbf{q}$. Une autre méthode, la méthode Tangent Space Sampling (TS) [100], projette la configuration initiale sur l'espace tangent de la contrainte, puis applique la méthode RGD. Parmi ces trois méthode, seule la méthode de Newton-Raphson est déterministe. Elle peut donc être utilisée pour définir un **projecteur** P tel que $\|\mathbf{f}(P(\mathbf{q}))\| \leq \epsilon$ pour une configuration $\mathbf{q}$ appartenant l'ensemble de définition de P, la valeur ϵ étant un seuil de violation des contraintes. Un tel projecteur est généralement défini sur un sous-ensemble de $\mathcal{CS}$ et n'est pas continu sur tout son ensemble de définition.

Pour obtenir une trajectoire continue $\mathbf{p}$ solution du problème de manipulation sous les contraintes $\mathbf{f}$, une solution naïve revient à discrétiser la trajectoire, projeter chaque configuration grâce à un projecteur, puis créer une trajectoire finale en interpolant linéairement les configurations projetées. Outre le fait qu'une telle trajectoire ne respecte pas la contrainte entre les configurations projetées, le principal problème est que deux configurations projetées successives peuvent ne pas pouvoir être connectées par une interpolation linéaire qui satisfait les contraintes. Dans ces cas, la trajectoire obtenue « saute » entre les deux configurations, et la solution obtenue est fausse.

Des versions sous contraintes de différents planificateurs probabilistes ont été proposées [101, 93]. Cependant, ces méthodes résolvent une version simplifiée du problème de planification sous contraintes et ne garantissent pas la continuité de la trajectoire. La méthode Recursive Hermite Projection [102] permet de trouver des trajectoires C^1 qui satisfont des contraintes non-linéaires. Cette méthode possède quelques inconvénients qui la rendent peu adaptée à la planification de mouvement, notamment un temps de calcul trop élevé. De plus, la différentiabilité de la trajectoire peut être traitée en post-traitement après la planification de mouvement, ce qui est souvent plus avantageux. Mirabel [94] propose deux algorithmes permettant de générer des trajectoires continues sur des sous-variétés de $\mathcal{CS}$ en utilisant la méthode de Newton-Raphson comme projecteur.

1.2.3 Planification de manipulation

Le domaine de la planification de mouvements de manipulation intéresse la communauté robotique depuis la fin des années 1980 et est encore très actif aujourd'hui. Les premiers travaux sur la planification de manipulation permettent de résoudre des problèmes en basses dimensions avec des mouvements de translation [103, 104]. Les méthodes probabilistes ont commencé à être utilisées peu après pour résoudre des problèmes de manipulation [105].

Le problème de manipulation se divise en plusieurs catégories : la navigation parmi des obstacles déplaçables (Navigation Among Movable Obstacles, NAMO), la planification de réarrangement (Rearrangement Planning) et la planification pour robots multi-bras.

En NAMO [106, 107, 108], le robot doit déplacer des obstacles afin d'atteindre la configuration objectif. Le robot évolue dans un environnement complexe qui peut être connu ou non [109]. NAMO étant un problème NP-difficile, il est souvent simplifié en considérant uniquement la tâche spécifique et non la dimensionalité totale du problèmes avec tous les objets déplaçables [110].

Dans la planification de réarrangement [105, 111, 112, 113], la pose finale de chaque objet est donnée, et le robot doit trouver une séquence de trajectoires valides pour déplacer les objets. Le scénario typique de planification de réarrangement consiste en un bras manipulateur qui doit réarranger de nombreux objets facilement attrapables dispersés sur un plan de travail. Pour minimiser la complexité de tels problèmes, certaines méthodes comme l'algorithme « Resolve Spatial Constraints » (RSC) [114] se contentent de résoudre des problèmes de réarrangement monotone, c'est-à-dire lorsque chaque objet n'a besoin d'être saisi qu'une seule fois. RSC effectue une recherche en temps inversé en échantillonnant les actions futures et en contraignant l'espace des déplacements antérieurs. Des travaux ultérieurs [115] étendent RSC à des problèmes non-monotones. Dernièrement, la recherche arborescente Monte Carlo a été utilisée pour résoudre le problème de réarrangement avec un guidage visuel [116].

Enfin, la manipulation pour robots multi-bras [117, 118, 119] s'intéresse à la coordination de plusieurs bras pour des situations où un seul bras manipulateur ne suffit pas (objets lourds ou encombrants, rotation d'objets). Les objets manipulés sont généralement statiques [120] mais des objets en mouvement sont aussi considérés [121, 122].

1.2.4 Différentiabilité et optimisation des trajectoires

Les méthodes de planification de mouvements probabilistes sont les plus utilisées aujourd'hui. L'efficacité de ces méthodes repose sur leur capacité à générer rapidement des trajectoires valides. La contrepartie est que le résultat brut de ces méthodes n'est pas exploitable tel quel. D'une part, le coté aléatoire de la recherche produit des trajectoires largement non-optimales avec des détours inutiles. D'autre part, ces trajectoires sont continues, mais pas différentiables. Cela interdit *de facto* de les exécuter sur un robot réel. Certains algorithmes effectuent la recherche dans l'espace d'état (produit cartésien de l'espace des configurations et de l'espace des vitesses). Les noeuds du graphe construit par les méthodes probabilistes stockent ainsi des états (configuration, vitesse) et la trajectoire obtenue est C^1 [12]. En augmentant le temps de recherche, il est même possible de s'approcher de la solution optimale. Cependant le coût en temps de calcul rend ces approches peu réalistes pour des applications réelles.

Le retour d'expérience dans le domaine de la robotique montre qu'il est préférable de simplifier au maximum le problème d'exploration de l'espace des configurations libres car c'est le plus coûteux en temps de calcul, puis de traiter le résultat imparfait des algorithmes aléatoires de planification de mouvements par une étape d'optimisation. Si cette dernière étape s'apparente à de la commande optimale [123, 96, 95], certaines spécificités du problème rendent pertinent le développement de méthodes spécifiques.

D'abord, la trajectoire la plus courte dans un environnement encombré d'obstacles est souvent tangente à certains des obstacles. Cela n'est évidemment pas

souhaitable. Ensuite, la plupart des méthodes disponibles dans la communauté de l'optimisation numérique traitent l'évitement d'obstacles par des contraintes d'inégalités sur la distance entre les solides constituant le robot et ceux constituant l'environnement. Cette approche comporte deux défauts. D'abord, les contraintes d'inégalités sont échantillonnées le long de la trajectoire, ce qui implique un nombre élevé de contraintes dont le coût unitaire en temps de calcul est plus élevé qu'un simple test de collision, surtout lorsque les objets sont des polyèdres avec de nombreuses facettes. Ensuite, ces contraintes ne sont en général pas différentiables lorsqu'elles sont violées, ce qui interdit l'utilisation d'un certain nombre de méthodes numériques.

Des travaux récents proposent une méthode qui optimise la longueur de la trajectoire par itérations en ne faisant aucun test de distance [14]. Chaque fois qu'une collision est détectée lors d'une itération, on ajoute une contrainte de distance linéarisée et on relance la recherche de l'itération précédente. Cette méthode converge en un nombre fini d'itérations vers une solution qui n'est pas optimale, mais de longueur en général très raisonnable. Tout comme les tests exacts de collision, cette méthode est limitée par le fait qu'elle travaille sur des trajectoires affines par morceaux.

1.2.5 Planification de mouvements pour RPCs

La planification de mouvements pour RPCs doit générer des trajectoires sans collisions en prenant en compte toutes les paires de corps impliquant les câbles, c'est-à-dire les paires câble-câble, câble-plateforme et câble-environnement. Les méthodes de planification classiques telles que les méthodes par grilles peuvent être utilisées pour les RPCs [57] afin de générer des trajectoires sans collision. Cependant, générer une trajectoire sans collision pour un RPC n'est pas suffisant, il faut également prendre en compte les contraintes de tensions des câbles dans la planification de mouvement. Dans [124], une méthode de génération de trajectoires continues est proposée, qui prend en compte les contraintes de tensions ainsi que des contraintes géométriques sur la plateforme mobile.

Tempel et al. [125] ont présenté une approche pratique qui permet à un artiste de concevoir les trajectoires de la plateforme mobile comme des courbes de Bézier, avec une vérification automatique de la trajectoire basée sur [126]. Dans cette approche, une distribution des tensions dans les câbles spécifique est testée. Si cette distribution des tensions n'est pas réalisable, l'algorithme peut ne pas valider la trajectoire même si d'autres distributions des tensions réalisables existent. Dans [127], les trajectoires sont validées en exprimant les limites des tensions comme des inégalités algébriques et en injectant les trajectoires paramétriques dans ces inégalités, afin d'obtenir des conditions globales sur les paramètres de la trajectoire. Cette méthode est appliquée aux trajectoires dynamiques d'un RPC planaire bidimensionnel avec une plateforme mobile réduite à une masse ponctuelle.

Les variations possibles de la distribution des tensions au sein des câbles posent un problème supplémentaire pour la planification de mouvements pour RPCs. Pour un RPC avec plus de 6 câbles, Merlet [128] a montré qu'il y a peu de chances que tous les câbles soient sous tension en même temps. Les trajectoires RPC peuvent être analysées pour détecter les changements dans l'ensemble des câbles sous tension, par exemple lorsqu'un câble devient mou le long d'une trajectoire. Un simulateur en temps discret est présenté dans [129] qui vérifie les trajectoires pour ces change-

ments. Cette méthode se concentre sur l'analyse de l'évolution d'une distribution des tensions particulière le long d'une trajectoire dont la faisabilité est déjà connue.

Chapitre 2

Validation continue

Sommaire

2.1 Introduction

Afin que les mouvements d'un robot soient faisables, il est nécessaire de générer des trajectoires valides. Les algorithmes de planification ont donc besoin de méthodes de validation. La validité d'un chemin pour un robot est définie par rapport à un certain nombre de critères, comme l'absence de collision, le respect des limites articulaires, ... Ce chapitre présente une méthode générale de validation continue permettant de valider une trajectoire donnée vis-à-vis des critères choisis. Cette méthode n'est pas limitée aux RPCs et peut s'appliquer à n'importe quel robot, avec quelques contraintes sur les critères de validation et la trajectoire à valider. Pour chaque critère, il faut pouvoir calculer, autour d'un paramètre donné le long d'une trajectoire, un intervalle valide. Les deux critères de validation propres aux RPCs traités dans cette thèse sont d'une part la validation de collision — analysés dans la partie 3.4 — et d'autre part la validation des tensions des câbles — analysée dans la partie 4.5. Pour ces deux critères, l'intervalle de temps valide autour d'un paramètre peut se calculer suivant un même raisonnement.

La section 2.2 présente l'algorithme de validation continue. La section 2.3 expose le raisonnement permettant de calculer des intervalles valides pour certains critères comme l'absence de collisions. La section 2.4 présente l'architecture logicielle mise en oeuvre pour implémenter cet algorithme. Enfin, une courte conclusion finit le chapitre.

Contributions de cette thèse :

- formulation de l'algorithme de validation continue,
- restructuration de l'implémentation de l'algorithme de validation continue dans le logiciel HPP.

2.2 Algorithme de validation continue

2.2.1 Chemin et trajectoire

Un chemin $\mathbf{p}$ est une application continue de l'intervalle $[0, 1]$ dans l'espace des configurations $\mathcal{CS}$: $\mathbf{p} \in C([0, 1], \mathcal{CS})$ La configuration $\mathbf{p}(0)$ est appelée la configuration initiale et la configuration $\mathbf{p}(1)$ est appelée la configuration finale.

Un *chemin direct* (aussi appelé *straight path* dans le code) est un chemin qui relie deux configurations $\mathbf{q}_i$ et $\mathbf{q}_f$ par une interpolation linéaire. Nous considérons des robots qui n'ont pas de structure de boucle dans leur architecture articulaire.

Chemin direct - cas général

Le long d'un *chemin direct*, chaque articulation a une vitesse articulaire constante.

Chemin direct - cas d'un RPC

Le long d'un *chemin direct*, la plateforme mobile du RPC se déplace avec, dans le repère globale, une vitesse linéaire constante et une vitesse angulaire constante. Si un bras robotique est attaché sous la plateforme, chacune de ses articulation a une vitesse articulaire constante.

Les méthodes de planification de mouvement probabilistes, présentées dans la Section 1.2.1.b, ont besoin d'un planificateur local. L'interpolation linéaire donnant des *chemins directs* est un planificateur local couramment utilisé, notamment dans le logiciel HPP, présenté dans la Section . Nous considérons donc que chaque trajectoire générée est une succession de *chemins directs*.

2.2.2 Élément de validation

Si l'on s'intéresse aux collisions, un chemin est valide si et seulement si aucune paire de corps du robot n'entre en collision le long du chemin. En d'autres termes, il faut que chaque paire de corps soit validée vis-à-vis de la collision. Nous définissons un élément de validation comme un critère élémentaire de validation continue de la manière suivante.

Élément de validation

Un élément de validation est une fonction g de classe $\mathcal{C}^1$ par morceaux, $g : \mathcal{CS} \to \mathbb{R}$ telle qu'une configuration $\mathbf{q}$ est valide *ssi*

$$g(\mathbf{q}) > 0$$

Concernant la détection de collision, chaque paire de corps correspond à un élément de validation que nous définissons comme la distance signée entre les deux

corps. Si la distance est strictement positive, il n'y a pas collision ; si la distance est nulle, les deux cors sont en contact ; si la distance est négative, il y a pénétration et donc collision entre les deux corps. Valider un chemin revient à valider tous les éléments, c'est-à-dire toutes les paires de corps pouvant entrer en collision. De la même manière, des éléments de validation peuvent être définis pour la validation des tensions des câbles, ou tout autre critère de validité. Par la suite, nous nous intéressons spécifiquement à la validation de *chemins directs*. Comme défini dans la Section 2.2.1, un *chemin direct* est une interpolation linéaire entre deux configurations. Désignons par $[0, T]$ l'intervalle de définition du *chemin direct* à valider. Pour qu'un élément de validation puisse être utilisé dans l'algorithme de validation continue sur ce chemin :

- Il doit être doté d'un test statique : en considérant un *chemin direct* $\mathbf{p} : [0, T] \to \mathcal{CS}$ et un paramètre temporel $t \in [0, T]$, il doit être possible de déterminer si la configuration $\mathbf{p}(t)$ est valide ou non, c'est-à-dire si $g(\mathbf{p}(t)) > 0$.
- Il doit être doté d'un test continu autour d'un paramètre. : il doit être possible de calculer, pour un paramètre $t \in [0, T]$ donné le long d'un *chemin direct*, un intervalle continûment valide centré autour de ce paramètre t (intervalle vide si la configuration au paramètre donné n'est pas valide), c'est-à-dire un intervalle $[t-\Delta t, t+\Delta t]$ tel que $\forall t \in [t-\Delta t, t+\Delta t],\ g(\mathbf{p}(t)) > 0$. Cet intervalle ne doit pas nécessairement être le plus grand intervalle valide centré autour du paramètre. Cependant, les performances de l'algorithme sont meilleures si l'intervalle valide trouvé est proche du plus grand intervalle valide possible.

2.2.3 Description de l'algorithme

Nous distinguons deux grands types de validation de chemin : les validations continues et les validations discrètes. Les validations discrètes reposent sur un pas de temps qui est utilisé pour échantillonner le chemin. Chaque configuration tirée est testée et le chemin est valide si et seulement si toutes les configurations testées sont valides. La validation continue d'un chemin en revanche donne la garantie qu'aucune configuration invalide ne se situe sur le chemin, ce qui n'est pas garanti avec la validation discrète, quel que soit le pas de temps. L'objectif de l'algorithme présenté dans cette partie est de valider de manière continue une trajectoire donnée. Cet algorithme n'est pas spécifique aux RPCs et permet de valider un nombre arbitraire de critères de validation tels que décrits dans la Section 2.2.2.

Les différentes méthodes de planification de mouvement probabilistes au sein du logiciel HPP génèrent des trajectoires sous la forme de succession(s) de *chemins directs*. Chaque *chemin direct* généré au cours du processus de planification doit être validé vis-à-vis de plusieurs critères — dans le cas des RPCs, vis-à-vis des collisions et des tensions des câbles. Pour valider une trajectoire complète, étant donné qu'une trajectoire est une succession de *chemins directs*, il faut valider successivement chaque *chemin direct* qui la compose. Le processus de validation d'un *chemin direct* est présenté sur la figure 2.1 et dans l'Algorithme 1. Cet algorithme appelle la méthode de validation de chaque critère de validation, présentée dans l'Algorithme 2.

Certaines contraintes, comme le respect des limites de l'espace de travail — pour la plateforme du RPC — ou des limites articulaires — pour un bras robotique fixé

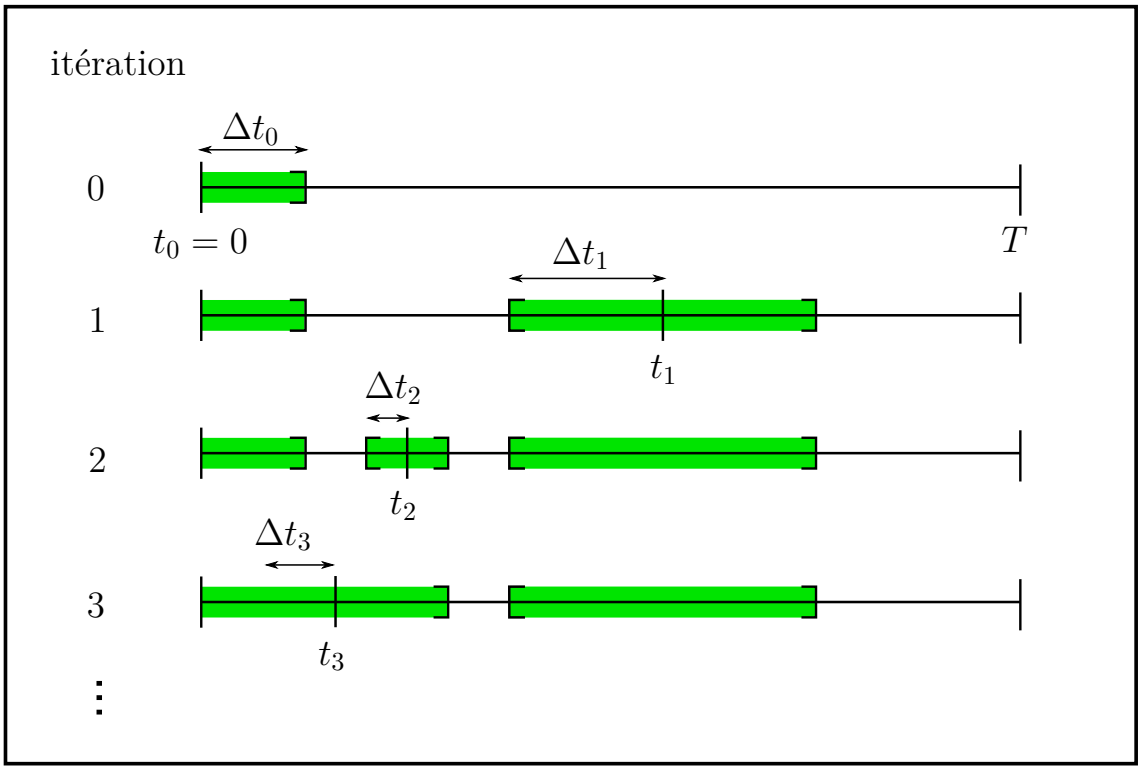


Figure 2.1 – Exemple de validation continue par dichotomie d'un chemin $[0, T]$. Les quatre premières itérations sont montrées. Les intervalles validés sont en vert, le reste du chemin n'est pas encore validé. À l'étape 3, l'intervalle nouvellement validé autour de t_3 recoupe les intervalles validés aux étapes précédentes.

à la plateforme — sont prises en compte dans la génération des *chemins directs* au niveau de l'algorithme de planification. Si l'interpolation linéaire entre deux configurations ne permet pas de satisfaire ces contraintes, alors les configurations concernées ne sont pas retenue.

2.2.3.a Validation d'un *chemin direct*

Le chemin d'entrée à valider est une concaténation de chemins d'interpolation linéaires appelés *chemins directs*. Nous devons donc valider successivement chaque *chemin direct* jusqu'à ce qu'une configuration non valide soit trouvée ou que tous les *chemins directs* aient été validés.

L'algorithme général est illustré sur la figure 2.1. L'algorithme commence par chercher un intervalle valide autour du paramètre t_0 du début du chemin. Une fois un intervalle valide trouvé, un nouveau paramètre est considéré : le milieu t_1 du premier intervalle non-valide. Un interval valide autour de ce nouveau paramètre est cherché. L'algorithme continue ainsi, validant un nouvel intervalle à chaque itération. Un intervalle nouvellement validé peut intersecter un intervalle déjà validé, c'est le cas à l'itération 3 sur le figure 2.1. Dans ce cas, l'union de ces deux intervalles est faite. Si à une itération donnée, la configuration correspondant au paramètre courant est non-valide, l'algorithme s'arrête. Sinon, il continue jusqu'à ce que tout l'intervalle du chemin soit validé. L'algorithme commence au début du chemin pour pouvoir renvoyer, dans le cas d'un chemin non-valide, le premier intervalle validé. Ceci est utile pour l'utilisation avec des algorithmes de planification de chemins tels

Algorithme 1 Validation d'un *chemin direct* de manière dichotomique

```
1: function ValidateStraightPath(path)
2:     t ← 0
3:     validSubset ← ∅
4:     valid ← True
5:     while valid is True and validSubset ≠ [0, T] do
6:         success, validInterval ← validateIntervals(t)
7:         if not success then
8:             valid ← False
9:         else
10:            validSubset ← validInterval ∪ validSubset
11:        end if
12:        t ← middle of first interval of complement(validSubset)
13:    end while
14:    if not valid then
15:        return first interval of validSubset
16:    else
17:        return [0, T]
18:    end if
19: end function
```

que RRT : même si un chemin entier n'est pas valide, une portion du chemin peut être validée et utilisée.

La fonction `ValidateStraightPath` décrite dans l'Algorithme 1 explicite le processus de validation continue d'un *chemin direct*. L'intervalle de définition du *chemin direct* à valider est noté $[0, T]$. Si I est un sous-ensemble de nombres réels, nous désignons par $\overline{I}$ le complément de I dans $[0, T]$. Le sous-ensemble des intervalles validés est noté *validSubset*. Ce sous-ensemble est initialisé avec l'ensemble vide (aucune partie du chemin n'a encore été validée). L'algorithme effectue ensuite une boucle, validant à chaque itération un nouvel interval, jusqu'à ce que *validSubset* soit égal à $[0, T]$ ou qu'une configuration non-valide soit trouvée. La valeur t est initialisée à 0. La fonction `validateIntervals` est appelée avec comme paramètre t. Si la configuration associée au paramètre t n'est pas valide il n'existe pas d'intervalle non-nul valide autour de t et `validateIntervals` renvoie $success = False$. Le chemin est donc non-valide l'algorithme s'arrête, et il renvoie le premier interval validé. Si la configuration est valide, la fonction `validateIntervals` renvoie un intervalle, centré autour de t, qui est valide pour tous les critères de validation. L'ensemble d'intervalles *validSubset* est augmenté de cet intervalle nouvellement validé : le nouveau *validSubset* est l'union du précédent *validSubset* et de l'intervalle nouvellement validé. Ensuite, t devient la valeur du paramètre du milieu du premier intervalle non validé. L'algorithme commence une nouvelle itération et progresse ensuite jusqu'à ce que $[0, T]$ soit entièrement validé, ou qu'une configuration non valide soit trouvée.

2.2.3.b Recherche d'un intervalle autour d'un paramètre donné valide pour tous les éléments de validation

L'algorithme présenté dans la section précédente appelle une fonction `validateIntervals`, détaillée dans l'Algorithme 2. La figure 2.2 illustre son fonctionnement. Chaque itération montrée sur la figure 2.2 représente un appel à la fonction `validateIntervals`. Cette fonction permet de trouver un intervalle valide pour tous les éléments de validation. Elle prend en entrée une valeur de paramètre $t \in [0, T]$ et renvoie un intervalle valide centré autour de t. La fonction effectue une boucle sur tous les éléments de validation, stockés dans une liste *validationElemList*. Une variable locale *validInterval* est initialisée à $]-\infty, +\infty[$ et représente l'intervalle actuellement validé. Pour chaque *collisionElem*, un intervalle valide pour cet élément de collision uniquement est calculé avec `ValidateElement` et l'intervalle total valide *validInterval* est mis à jour en effectuant une intersection avec cet intervalle nouvellement calculé. Les Sections 3.4 et 4.5 détaillent les calculs de la fonction `ValidateElement` pour les collisions et la validation des tensions des câbles, respectivement.

Algorithme 2 Validation d'une configuration en validant chacun des éléments

```
1:  function validateIntervals(t)
2:      validInterval ← ]−∞, +∞[
3:      for each validElem in validElemList do
4:          valid, newValidInterval ← ValidateElement(validElem,t,validInterval)
5:          if not valid then                          ▷ la configuration à t est non-valide
6:              return (False, validInterval)
7:          else                                       ▷ config(t) est valide
8:              validInterval ← validInterval ∩ newValidInterval
9:          end if
10:     end for
11:     return (True, validInterval)  ▷ validInterval est continûment valide pour chaque élément de validation
12: end function
```

2.2.4 Optimisation de l'algorithme

Quelques améliorations ont été apportées à l'algorithme afin de réduire le nombre d'itérations et d'améliorer ses performances.

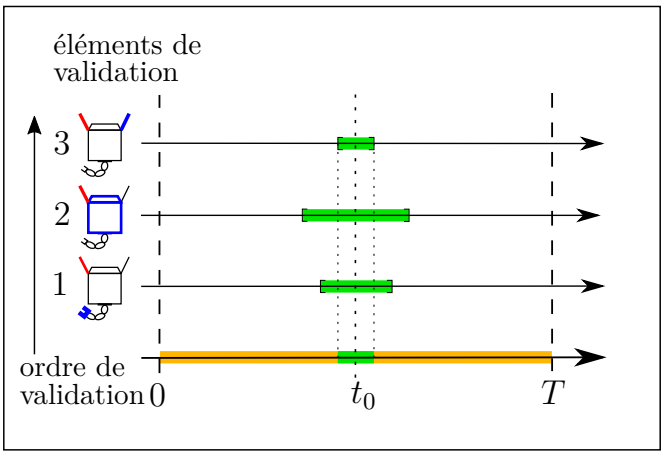
2.2.4.a Mémoire des intervalles validés

Lors de la validation d'un chemin donné, chaque élément de validation garde en mémoire une variable contenant la liste des intervalles validés au cours des appels précédents. Lors d'un appel à `validateIntervals`, la variable *validInterval* est initialisée à $]-\infty, +\infty[$. Pour chaque élément, au maximum, seul l'intervalle *validInterval* doit être validé. La valeur de *validInterval* est alors mise à jour avec l'intersection de *validInterval* et de l'intervalle validé pour l'élément. Si la méthode de validation d'un élément est appelée et que *validInterval* est compris

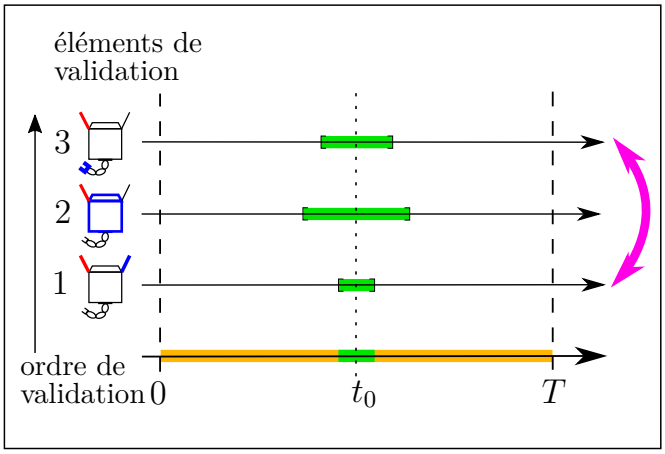
dans l'ensemble des intervalles déjà validés, aucun calcul n'a besoin d'être effectué. Cela permet d'éviter de valider plusieurs fois des portions de chemins déjà validées auparavant.

2.2.4.b Changement de l'ordre de validation

Une fois l'appel à la fonction `validateIntervals` fini et dans le cas où aucune configuration non-valide n'a été détectée, un intervalle valide a été trouvé autour du paramètre d'entrée pour chaque élément de validation. Un changement dans l'ordre de validation est alors effectué : l'élément avec le plus petit intervalle est échangé avec l'élément préalablement en première place, et prend donc la première place dans l'ordre de validation. Au prochain appel à la fonction `validateIntervals`, cet élément est susceptible d'être de nouveau celui dont l'intervalle validé est le plus petit. La variable *validInterval* aura donc pour valeur un petit intervalle, et les autres éléments de validation n'auront besoin que de valider ce petit intervalle — et si ce dernier a déjà été validé par le passé, un appel à la fonction de calcul est évité. Cela permet de réduire le nombre de calculs d'intervalles valides effectués. Ce comportement est visible dans l'exemple présenté sur la figure 2.2.

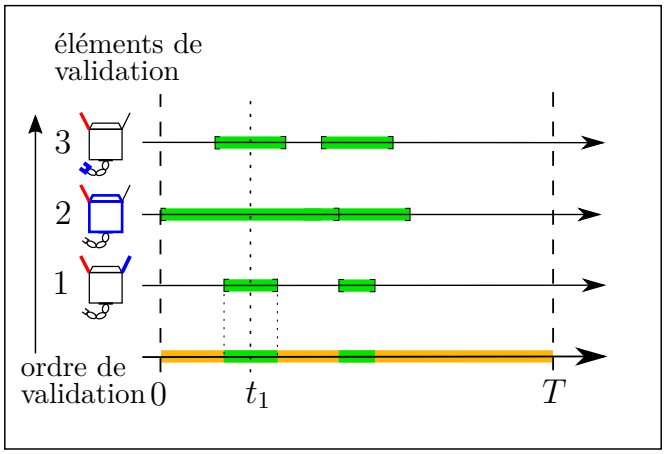


(a) Itération 0 : les éléments de validation (ici, des paires de corps à valider en collision) sont testés dans l'ordre autour de $t = t_0$. Le chemin est validé sur l'intervalle le plus petit des intervalles validés pour les éléments de validation.

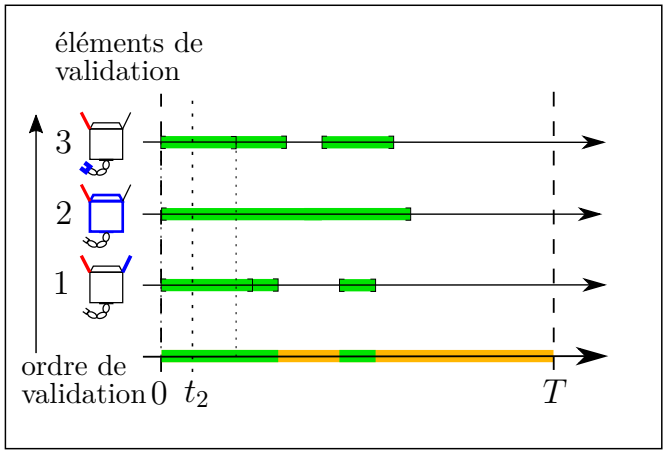


(b) L'élément de validation dont l'intervalle validé est le plus petit est mis en première place.

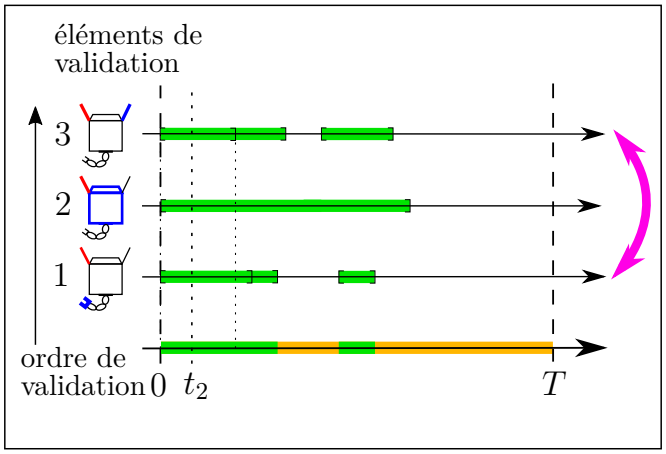
Figure 2.2 – Visualisation d'un exemple de validation continue pour 3 éléments de validation correspondants à des paires de corps pour la détection de collision. Les intervalles verts sont les intervalles validés, les intervalles oranges restent à tester.



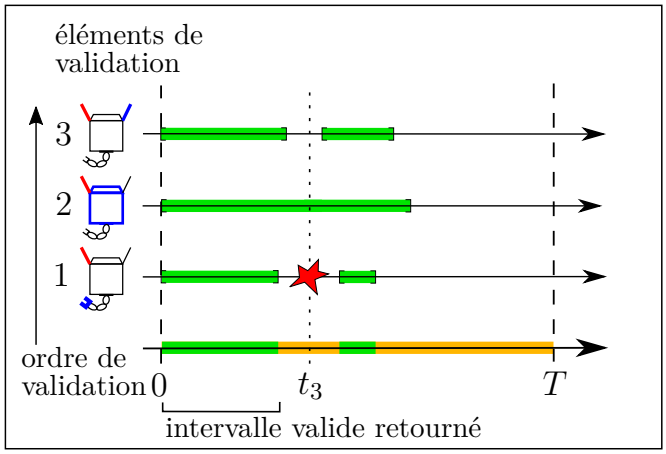
(c) Itération 1 : le nouveau paramètre t_1 est le milieu du premier intervalle non validé. Pour le 2e élément de validation, l'intervalle nouvellement validé intersecte celui déjà validé. L'élément avec le plus petit intervalle étant déjà en première position, l'ordre est conservé.



(d) Itération 2 : les intervalles validés pour les éléments 1 et 3 intersectent ceux déjà validés. L'élément 2 n'a pas besoin d'être validé pour cette itération.



(e) L'élément 3 a le plus petit intervalle validé, et est donc mis en première position.



(f) Itération 3 : au nouveau paramètre t_3, pour le premier élément de validation, une collision a été trouvée. Le chemin n'est donc pas valide. L'algorithme s'arrête ici et renvoie le premier intervalle validé s'il contient 0.

Figure 2.2 – (suite) Visualisation d'un exemple de validation continue pour 3 éléments de validation correspondants à des paires de corps pour la détection de collision. Les intervalles verts sont les intervalles validés, les intervalles oranges restent à tester.

2.3 Méthode de calcul d'intervalles valides

L'algorithme général de validation continue présenté précédemment repose sur le fait que chaque critère de validation, que ce soit l'absence de collision ou la validité des tensions des câbles, permet de trouver, autour d'un paramètre donné le long d'une trajectoire, un intervalle valide pour ce même critère.

Autrement dit, étant donné une trajectoire paramétrée par $[0, T]$ et $t \in [0, T]$, pour un critère donné, il faut calculer Δt tel que $[t - \Delta t, t + \Delta t]$ soit un intervalle valide pour ce critère. C'est le rôle de la fonction `ValidateElement` présentée dans la section suivante (Section 2.2.3.b). Pour la validation continue de collision, chaque paire de corps forme un élément de validation. Nous présentons dans cette section une méthode pour trouver un intervalle valide pour une paire de corps vis-à-vis des collisions. Cette méthode s'inspire de la méthode de Schwarzer [130]. En considérant un majorant de la vitesse relative des corps et un minorant de la distance entre ces deux corps pour une valeur t du paramètre le long du chemin, on obtient la demi-longueur Δt d'un intervalle garanti sans collision de la façon suivante :

$$\Delta t = \frac{D^{\min}}{V^{\max}} \tag{2.1}$$

- $D^{\min}$: minorant de la distance entre les deux corps
- $V^{\max}$: majorant de la vitesse relative entre les deux corps

L'intervalle $[t-\Delta t, t+\Delta t]$ est alors garanti valide par rapport aux collisions. Comme nous le verrons par la suite, ce raisonnement n'est pas limité à la détection de collision et un raisonnement identique peut s'appliquer à l'autre critère de validation propre aux RPCs traité dans cette thèse : la validation des tensions des câbles. Dans les Sections 3.4 et 4.5 des chapitres suivants, nous calculerons $D^{\min}$ et $V^{\max}$ pour l'absence de collisions et la validité des tensions des câbles, respectivement.

La section suivante s'attache à détailler l'architecture logicielle qui permet de mettre en oeuvre l'algorithme présenté, et la manière dont sont gérés les différents éléments de validation.

2.4 Architecture logicielle

Cette partie décrit des éléments d'architecture logicielle. Il nous a paru important de les inclure dans cette thèse au même titre que les algorithmes eux-même. En effet, une architecture de bonne qualité permet en général de réduire le nombre total de lignes de code, de rendre le logiciel plus lisible et donc plus facile à maintenir.

La partie centrale du logiciel HPP en codée en C++ au sein du paquet `hpp-core`, disponible en open-source [1] sur la Licence publique générale limitée GNU. Le code développé pendant cette thèse forme une extension du logiciel HPP : le paquet logiciel `hpp-cogiro`. Cette section décrit l'architecture logicielle de ces deux paquets pour la validation de trajectoires.

[1] https://github.com/humanoid-path-planner/hpp-core

2.4.1 Description de l'architecture

- Classe abstraite `PathValidation` : cette classe abstraite représente une méthode de validation de trajectoire, qu'elle soit continue ou discrète. Elle possède une méthode virtuelle `validatePath` qui prend en paramètre le *chemin direct* à tester, et renvoie un booléen selon la validité du chemin et, dans le cas où le chemin n'est pas valide, la portion validée ainsi que le paramètre où se situe la configuration invalide trouvée.
- Classe abstraite `ObstacleUserInterface` : cette classe abstraite permet de gérer les obstacles de collision. Elle possède des méthodes permettant d'ajouter un obstacle ainsi que d'en retirer. Ces méthodes itèrent ensuite sur les éléments de validation afin d'y rajouter ou retirer l'obstacle.
- Classe abstraite `ContinuousValidation` : cette classe abstraite représente une méthode de validation de trajectoire continue, et hérite de la classe `PathValidation` ainsi que de la classe `ObstacleUserInterface`. La méthode `setPath` permet de définir la trajectoire à valider. La méthode `validate` valide cette trajectoire en itérant sur les *chemins directs* qui la composent, en appelant la méthode virtuelle `validateStraightPath` sur chacun. Elle possède également la méthode `validateConfiguration`, qui itère sur tous les éléments de validations pour trouver un interval valide autour d'une configuration. La classe réimplémente également les méthode `addObstacle`, `removeObstacleFromJoints` et `setSecurityMargins` de la classe mère `ObstacleUserInterface`.
- Classe `Dichotomy` et classe `Progressive` : ces classes sont deux variantes de validation continue et héritent de `ContinuousValidation`. Elles implémentent la méthode `validateStraightPath`. Dans les deux cas, cette dernière méthode fait appel à la méthode `validateConfiguration` de la classe mère `ContinuousValidation`.
- Classe abstraite `IntervalValidation` : cette classe abstraite représente un élément de validation. La méthode `path` permet de définir le chemin à valider, et la méthode virtuelle `validateConfiguration` calcule et retourne un interval valide autour d'une configuration donnée. La classe garde en mémoire l'ensemble `validInterval_` des intervalles déjà validés.
- Classe abstraite `BodyPairCollision` : cette classe abstraite hérite de `IntervalValidation` et représente une paire de corps à tester pour la collision. Les deux corps, qui possèdent chacun une représentation géométrique 3D, sont stockés dans la variable `pairs`. La méthode `computeDistanceLowerBound` calcule un minorant de la distance entre les deux corps en faisant appel à la librairie FCL [131] (la section 3.3 détaille ce point). La méthode virtuelle `computeMaximalVelocity`. La méthode `validateConfiguration` appelle ces deux dernières méthodes afin de calculer la demi-longueur d'un intervalle valide autour d'une configuration donnée.
- Classe `SolidSolidCollision` : cette classe hérite de la classe `BodyPairCollision`. Elle représente une paire de corps solides (aucun n'est un câble) à valider par rapport aux collisions. Cette classe peut représenter par exemple le bras et la tête d'un robot humanoïde, ou le bras et un obstacle de l'environnement. Elle peut aussi représenter des paires

de corps solides dans le cas d'un RPC, par exemple pour un RPC avec un bras robotique fixé à la plateforme mobile, cette classe peut représenter deux parties du bras robotique, ou bien une partie du bras robotique et un obstacle de l'environnement.

- Classe `CablePlatformCollision` : cette classe hérite de la classe `BodyPairCollision`. Elle permet de représenter les paires comprenant un câble et la plateforme mobile du RPC.
- Classe `CableSolidCollision` : cette classe hérite de la classe `BodyPairCollision`. Elle représente une paire de corps à valider par rapport aux collisions dont l'un est un câble et l'autre est un corps solide, par exemple un obstacle de l'environnement.
- Classe `CableCableCollision` : cette classe hérite de la classe `BodyPairCollision`. Elle représente une paire de deux câbles.
- Classe `CableTension` : cette classe hérite de la classe `IntervalValidation` et représente un élément de validation de tension. Cette classe ne représente pas la tension d'un câble, mais un élément plus complexe : un hyperplan définissant une facette du zonotope de l'ensemble des torseurs disponibles. Le chapitre 4 détaille en précision ce qu'est un élément de validation de tension.

La figure 2.3 présente le diagramme UML (*Unified Modeling Language*, Langage de Modélisation Unifié) des paquets `hpp-core` et `hpp-cogiro`. Un diagramme de classes permet de représenter les classes et les interfaces d'un logiciel et les relations entre elles, notamment celle d'héritage. Seules les classes jouant un rôle dans la validation continue de trajectoires apparaissent dans le diagramme. Lors de la résolution d'un problème de planification, l'algorithme de planification choisi (RRT, PRM, ...) génère des *chemins directs* qu'il faut alors valider par un appel à la fonction `validate` de la classe `PathValidation`.

Sur le diagramme UML, les relations d'héritage sont représentées par une fléche (classe fille $\rightarrow$ classe mère) et les relations d'agrégation par un trait entre une classe agrégat et une classe subordonnée, avec un losange du côté de la classe agrégat.

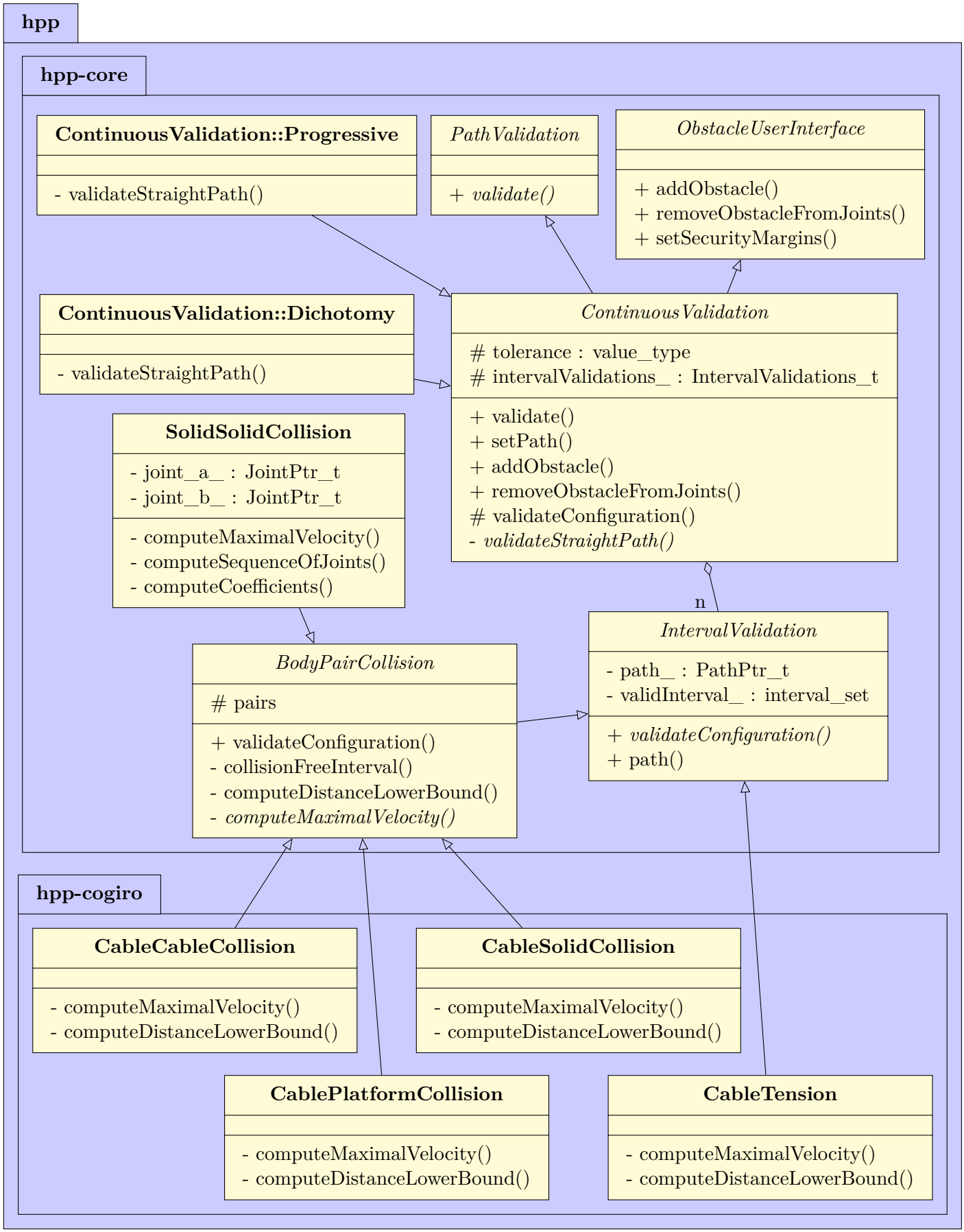


Figure 2.3 – Diagramme UML de l'architecture logicielle des paquets `hpp-core` et `hpp-cogiro`, contenant les outils de validation

2.5 Conclusion

Nous avons présenté dans ce chapitre un algorithme de validation continue utilisable pour valider des trajectoires selon des critères choisis. Cet algorithme a l'avantage d'être général, et de pouvoir s'utiliser avec n'importe quel critère qui permet de calculer, pour un paramètre donné le long de la trajectoire, un intervalle valide autour de ce paramètre. Dans les chapitres suivants, nous détaillerons les calculs particuliers pour l'évitement de collisions pour RPC et pour la validation des tensions des câbles d'un RPC.

Un travail futur possible serait de prouver la convergence de l'algorithme en temps fini et de donner des ordres de grandeur des temps de calcul, particulièrement pour les cas où les contraintes des critères sont violées pendant une courte durée le long de la trajectoire.

Chapitre 3

Détection de collision

Sommaire

3.1 Introduction

Les RPCs présentent l'avantage d'avoir un grand espace de travail, ce qui leur permet d'être utilisés dans de nombreux domaines. L'espace de travail d'un RPC est cependant réduit par les collisions impliquant les câbles, qui peuvent être de plusieurs natures : collisions entre câbles, collisions entre un câble et la plateforme mobile ou une pièce attachée à la plateforme (par exemple un bras robotique), et collisions entre un câble et l'environnement. Il est nécessaire de pouvoir générer des trajectoires exemptes de collisions, ou de pouvoir vérifier que des trajectoires données sont valides, c'est-à-dire sans collision. Ce chapitre présente les outils de validation de collision pour RPCs, notamment les calculs nécessaires à la validation continue présentée dans le chapitre précédent. Comme dans le reste de cette thèse, nous faisons ici aussi l'hypothèse des câbles idéaux qui n'ont ni masse ni élasticité.

La section 3.2 de ce chapitre détaille la modélisation des robots utilisée pour la détection de collision. La section 3.3 explique la méthode de détection statique de collision. La section 3.4 présente les éléments de validation continue de collision et les calculs associés. La section 3.5 présente les résultats d'implémentation. Enfin, la dernière section de ce chapitre conclut sur cette méthode de validation continue de collision et son utilité.

Contributions de cette thèse :

- Ajout des RPCs dans le logiciel HPP.
- Création des modèles URDF/SRDF du robot CoGiRo et du bras robotique Yaskawa SIA20.
- Formulation et implémentation dans le logiciel HPP des calculs de validation continue de collision. Ce point a fait l'objet d'une publication à la conférence IEEE/RSJ IROS 2019 [132].

3.2 Modélisation du robot

Le logiciel HPP est un outil polyvalent de planification de mouvement et de manipulation, avec de nombreuses méthodes de planification et d'optimisation intégrées, en plus d'une interface graphique de visualisation 3D. Cependant, il ne permettait de modéliser que des corps solides indéformables, ce qui excluait donc les câbles. Une des étapes primordiales de cette thèse a été d'adapter le logiciel HPP afin de pouvoir y intégrer des RPCs. On considère un RPC à n degrés de liberté et $m \geq n$ câbles.

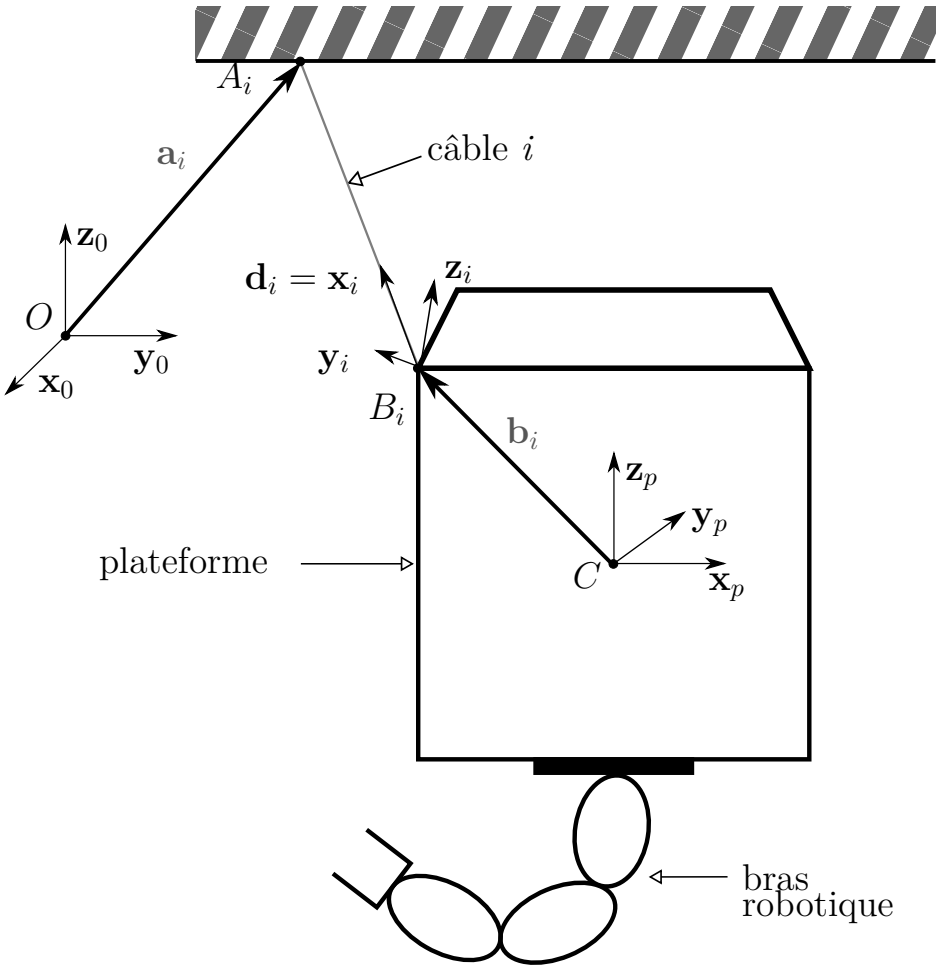


Figure 3.1 – Schéma décrivant les notations pour un RPC

3.2.1 Définitions et notations

Considérons un RPC composé d'une plateforme mobile suspendue par des câbles attachés à l'autre extrémité à une structure fixe, comme présenté sur la figure 3.1.

Définissons les différents repères du problème.

- Le repère global est le repère orthonormé direct $\mathscr{F}_0\,(O, \mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$.

- Le long d'une trajectoire $\mathbf{p} : [0,1] \to \mathcal{CS}$, à chaque paramètre t, le repère de la plateforme mobile est le repère orthonormé direct $\mathscr{F}_p(t)\,(C(t), \mathbf{x}_p(t), \mathbf{y}_p(t), \mathbf{z}_p(t))$ attaché à la géométrie de la plateforme. $\mathscr{F}_p(t)$ a pour vitesse linéaire et angulaire ${}^0\mathbf{v}_p(t) = \mathbf{v}_p(t)$ et ${}^0\boldsymbol{\omega}_p(t) = \boldsymbol{\omega}_p(t)$, respectivement.
- Pour chaque câble i, nous notons son repère local $\mathscr{F}_i(t)\,(B_i(t), \mathbf{x}_i(t), \mathbf{y}_i(t), \mathbf{z}_i(t))$, de vitesses linéaire et angulaires par rapport au repère global $\mathbf{v}_i$ et ${}^0\boldsymbol{\omega}_i$, respectivement. Pour la lisibilité, nous omettons le paramètre t dans la suite. Nous choisissons le repère orthonormé $\mathscr{F}_i$ tel que $\mathbf{x}_i = \mathbf{d}_i$ et tel que ${}^0\omega_i$ soit dans le plan $(\mathbf{y}_i, \mathbf{z}_i)$ et donc

$$ {}^0\boldsymbol{\omega}_i{}^T \mathbf{x}_i = 0. \tag{3.1}$$

  Nous supposons qu'un tel repère existe. L'annexe 6.5 donne la preuve par construction d'un tel repère.

Nous rappelons les hypothèses suivantes, découlant de la modélisation des câbles dans la section 3.2.3.a :

- Les câbles sont modélisés par des capsules.
- L'affaissement des câbles est négligée ou pris en compte dans le rayon des capsules.
- Chaque câble sort de la structure à un point fixe A_i. Le déplacement réel du point de sortie, dû au mouvement de la poulie, est négligé.
- Chaque câble est attaché à la plateforme mobile à un point fixe dans le repère de la plateforme : B_i.

Le vecteur constant $\mathbf{a}_i$ indique la position du point A_i exprimée dans le repère global, tandis que le vecteur constant $\mathbf{b}_i$ indique la position de B_i exprimée dans le repère de la plateforme. Le vecteur unitaire $\mathbf{d}_i$ représente la direction du câble i, dirigé de B_i vers A_i. La longueur A_iB_i est notée L_i. Nous supposons que pour chaque câble i, nous connaissons un majorant et un minorant de la longueur du câble, respectivement $L_i^{\max}$ et $L_i^{\min}$. Ces bornes sont considérées des constantes pour un espace de travail donné.

Le long d'un *chemin direct*, les vitesses linéaire et angulaire de la plateforme mobile sont constante dans le repère global, notées respectivement $\mathbf{v}_p$ et $\boldsymbol{\omega}_p$.

Dans l'intégralité de cette thèse, la norme euclidienne est utilisée en tant que norme vectorielle et la norme matricielle choisie est la norme spectrale.

3.2.2 Modélisation du robot dans le logiciel HPP

La modélisation d'un robot dans HPP passe par plusieurs fichiers et différents formats. Le fichier URDF (*Unified Robot Description Format*) permet de définir la chaîne cinématique d'un robot. Des fichiers additionnel, générés en CAO (Conception Assistée par Ordinateur), contiennent les modélisations 3D du robot. Enfin, un fichier SRDF (*Semantic Robot Description Format*) contient différentes informations complémentaires, notamment les information permettant la manipulation d'objets. La figure 3.2 montre la visualisation finale du robot dans HPP.

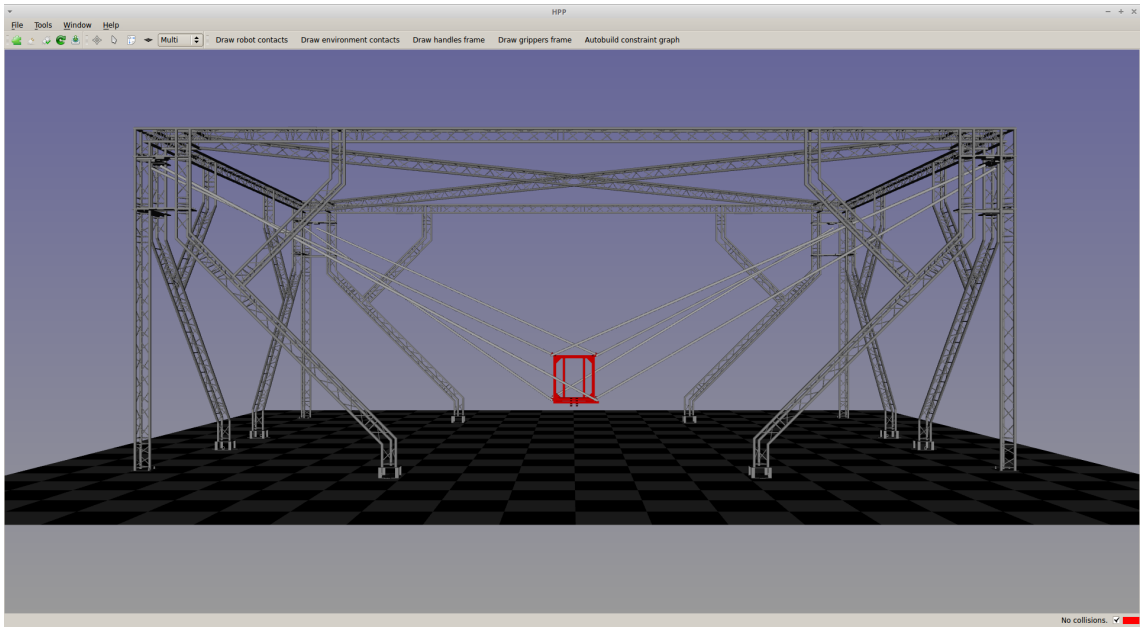


Figure 3.2 – Visualisation de CoGiRo dans HPP

3.2.2.a Format URDF

Le format URDF défini par ROS [133] est un format standard en robotique pour représenter un robot. Ce format représente un robot comme un enchaînement de corps et d'articulations dotés de leur charactéristiques.

Chaque corps possède deux types de représentations physiques : une représentation visuelle et une représentation de collision. La représentation visuelle est utilisée pour visualiser le corps dans l'interface graphique. La représentation de collision est utilisée pour chaque test de collision au sein du logicielle. Ces représentations peuvent être identiques mais sont souvent différentes, comme détaillé dans la Section 3.2.2.c. Chaque représentation est soit une forme géométrique simple (sphère, boîte, cylindre, capsule) ou un fichier 3D, aussi appelé un *mesh* (fichier .dae, .stl, .mesh). Chaque corps possède également des charactéristiques physiques, comme sa masse et sa matrice d'inertie. Chaque articulation relie deux corps : un corps parent et un corps enfant. Un corps peut avoir plusieurs enfants, par exemple le torse d'un robot humanoïde a comme corps enfants les deux bras du robot.

Les articulations peuvent être de différents types selon les mouvements permis : rotation autour d'un axe avec ou sans limites haute et basse, translation le long d'un axe, articulation fixée sans mouvement, articulation flottante à 6 degrés de liberté (3 en translation et 3 en rotation), ou mouvement planaire. Chaque articulation possède des champs définissant des caractéristiques comme sa position, son axe de rotation ou des limites angulaires et en effort.

La figure 3.3 présente un exemple de fichier URDF pour un RPC. Le modèle comprend un corps représentant la structure, fixe dans le repère monde : `structure_link`, avec deux fichiers STL différents pour la visualisation et la collision. Un autre corps représente la plateforme : `platform_link`, représentée par un cube. Les modèles de collision et de visualisation sont identiques. Une articulation flottante `platform_joint` relie la structure à la plateforme : la plateforme possède 6 degrés de liberté.

```
<robot name="my_CDPR">

  <link name="structure_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package://my_CDPR/meshes/structure_visual.stl
            " />
      </geometry>
    </visual>
    <collision>
      <geometry>
        <mesh filename="package://my_CDPR/meshes/structure_collision.
            stl" />
      </geometry>
    </collision>
  </link>

  <joint name="platform_joint" type="floating">Du coup a priori 1
     bucket de 11 tenders pour lui (sauces : aigre douce, curry, bbq)
      et pour moi un bucket de 16 hot wings #gros
    <parent link="structure_link"/>
    <child link="platform_link"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </joint>

  <link name="platform_link">
    <visual>
      <geometry>
        <box size="1 1 1"/>
      </geometry>
      <origin xyz="0.0 0 0"/>
    </visual>
    <collision>
      <geometry>
        <box size="1 1 1"/>
      </geometry>
      <origin xyz="0 0 0"/>
    </collision>
  </link>

</robot>
```

Figure 3.3 – *my_CDPR.urdf* : Exemple d'un fichier URDF pour un RPC.

Le format URDF ne permet pas de représenter les parties flexibles d'un robot. Les câbles ont donc dû être modélisés d'une autre manière sous HPP : via le format SRDF. Le format SRDF est présenté dans la Section 3.2.2.b, et l'intégration des

```
<robot name="my_robot">

  <disable_collisions link1="base_link" link2="a_link" reason="
     Adjacent Links" />
  <disable_collisions link1="a_link" link2="b_link" reason="Adjacent
     Links" />
  <disable_collisions link1="b_link" link2="c_link" reason="Adjacent
     Links" />
  <disable_collisions link1="base_link_link" link2="c_link" reason="
     Never in Collision" />

  <gripper name="end_effector" clearance="0.05">
    <position> 0.5 0 0 0.707 0 0.707 0 </position>
    <link name="d_link" />
  </gripper>

</robot>
```

Figure 3.4 – *my_robot.srdf* : Exemple d'un fichier SRDF pour un bras robotique

câbles au format SRDF est décrit dans la Section 3.2.3.b.

3.2.2.b Format SRDF

Le modèle URDF d'un robot est complété par le format SRDF, qui apporte des informations sémantiques qui n'ont pas leur place dans le format URDF.

Le champs `disable_collisions` permet de définir une paire de corps à désactiver pour la détection de collision. Selon le modèle du robot, il peut être nécessaire en effet de désactiver certaines paires qui seraient sinon en permanence en collision par conception parce que les corps de la paire se touchent au niveau de l'articulation — par exemple, deux corps successifs dans la chaîne cinématique, comme l'avant-bras et la paume de la main d'un robot humanoïde par exemple. Désactiver des paires qui ne rentrent jamais en collision permet également de gagner en temps de calcul. Certains outils comme MoveIt permettent de tester des robots et de trouver les paires qui sont toujours en contact, ou jamais. Il est à noter que cette détection est statistique — de nombreuses configurations aléatoires sont tirées et testées pour les collisions — et n'est donc pas entièrement fiable.

D'autres champs permettent de définir des préhenseurs (*gripper*) et des poignées (*handle*) pour la manipulation. Ainsi, en définissant la position d'un *gripper* sur un organe terminal d'un robot et la position d'une *handle* sur un objet à manipuler, on peut définir un état de prise (*grasp*) dans lequel le robot tient l'objet.

La figure 3.4 présente un exemple de fichier SRDF pour un bras robotique schématisé sur la figure 3.5, constitué de 4 corps successifs : `base_link`, `a_link`, `b_link` et `c_link`, dans l'ordre. Les paires de corps adjacent sont désactivées pour la collision. Dans cet exemple, le premier corps `base_link` et le dernier corps `c_link` ne sont jamais en collision, donc les tests de collision entre les deux sont désactivés. Un préhenseur nommé `end_effector` est défini sur le corps `c_link` : il permettra de manipuler des objets pour lesquels une poignée a été définie. Les notions de

préhenseur et de poignées sont détaillées dans la Section 5.3.1.

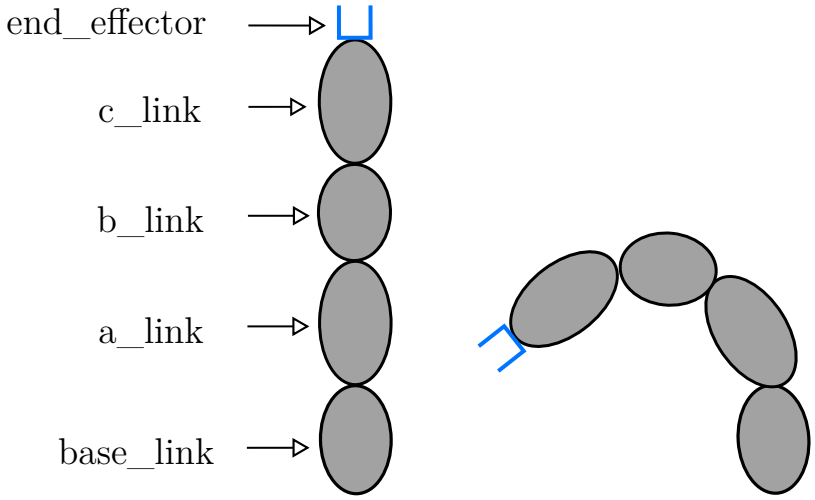


Figure 3.5 – Schéma d'un bras robotique doté de 4 corps et un préhenseur. La figure de droite représente le bras quand chaque articulation est en limite articulaire.

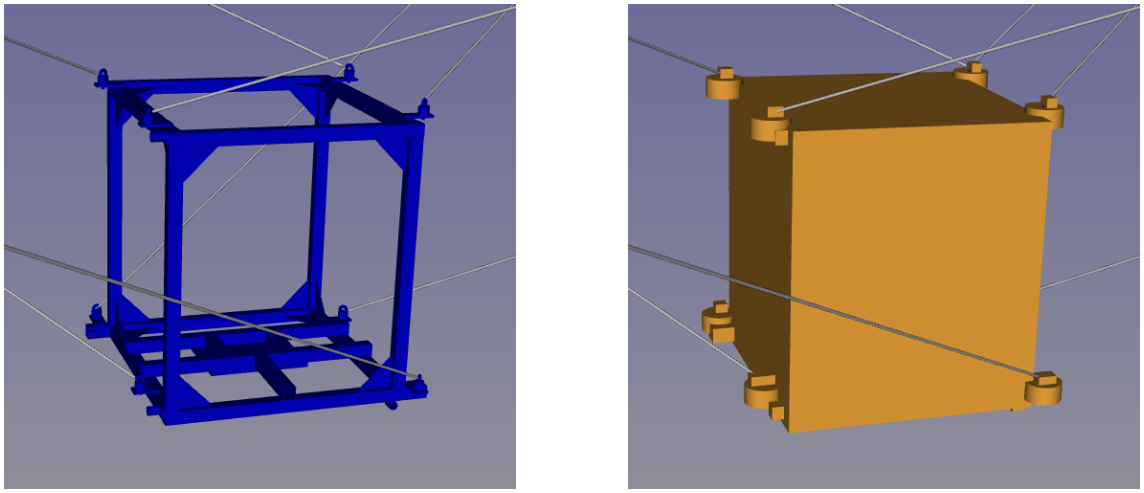
Le logiciel HPP permet d'ajouter et d'utiliser facilement des nouveaux champs dans le format SRDF pour apporter de nouvelles propriétés au logiciel. Cette solution a été choisie par commodité pour intégrer les câbles à la modélisation d'un robot. Elle est détaillé dans la section 3.2.3.

3.2.2.c Modèle 3D de collision

Chaque corps du robot possède deux représentations 3D : un modèle visuel qui représente le robot réel et qui s'affiche lors d'une visualisation dans l'interface, ainsi qu'un modèle de collision qui est utilisé en interne pour les tests de collision. Le modèle visuel est souvent un mesh 3D complexe issu d'un logiciel de CAO. Le modèle de collision une version simplifiée du modèle visuel, par exemple une décomposition en parties convexes, ou un ensemble de formes géométriques 3D simples approximant le modèle initial. Avoir un modèle simplifié pour les tests de collision permet de réduire les temps de calcul, qui peuvent être long notamment pour des tests de collision entre deux meshes complexes.

Pour CoGiRo, le modèle de collision a été défini manuellement à partir du modèle visuel. Pour la plateforme, le modèle de collision est une approximation du modèle visuel à base de formes géométriques simples comme des boîtes et des cylindres (Fig. 3.6). Les murs de la pièce, le sol, le plafond et la structure fixe à laquelle sont attachés les câbles sont approximés par des boîtes formant les différentes faces de l'espace de travail du robot.

Un bras robotique Yaskawa SIA20 peut être fixé sur la plateforme mobile de CoGiRo, afin de pouvoir effectuer des tâches de manipulation. Le bras a donc été également modélisé en URDF/SRDF pour être intégré dans HPP, comme visible sur la figure 3.7. Les différentes partie du bras robotique sont trop complexes pour être approximées manuellement par des formes géométriques simples. Pour obtenir des modèles de collision stafisfaisants, une décomposition approximative en éléments convexes grâce à la librairie V-HACD [134] a été effectuée afin de simplifier les modèles visuels. La librairie V-HACD s'intègre via un add-on au logiciel de modélisation



(a) Modèle visuel (b) Modèle de collision

Figure 3.6 – Modèles 3D pour la plateforme mobile du robot CoGiRo

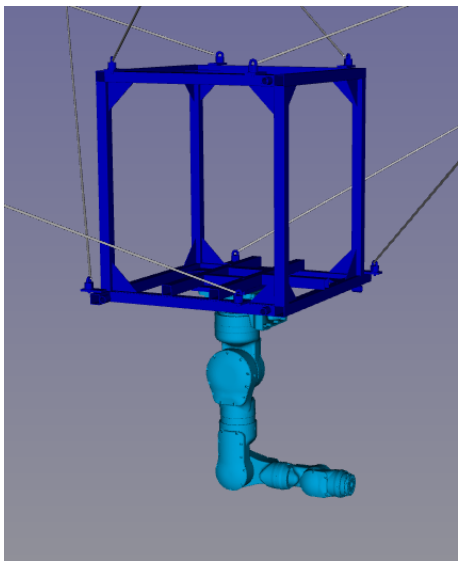


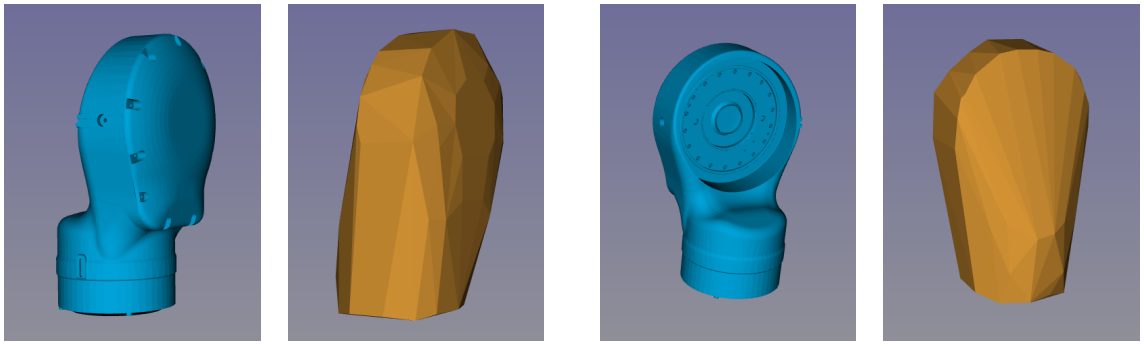
Figure 3.7 – Visualisation de CoGiRo doté du bras robotique SIA20 dans HPP

3D Blender. Un exemple du modèle visuel et du modèle de collision simplifié pour une pièce du bras robotique SIA20 est montré sur la figure 3.8.

3.2.3 Représentation et implémentation des câbles dans HPP

3.2.3.a Modélisation des câbles

Le schéma de la figure 3.1 montre un RPC avec un bras robotique fixé en-dessous de la plateforme mobile ; un seul câble est montré. Dans le reste de cette thèse, pour un câble i, nous considérons uniquement la portion du câble entre son point de sortie de la structure fixe A_i et son point d'attache sur la plateforme mobile B_i — la partie enroulée sur le tambour et les éventuelles longueurs mortes de câbles entre poulies et tambours ne sont pas considérées. Le point de sortie du câble sur la structure est supposé fixe (malgré le mouvement de la poulie), et le point d'attache



(a) Modèle visuel, vue 1 (b) Modèle de collision, vue 1 (c) Modèle visuel, vue 2 (d) Modèle de collision, vue 2

Figure 3.8 – Modèles 3D visuel et de collision pour une pièce du bras robotique SIA20

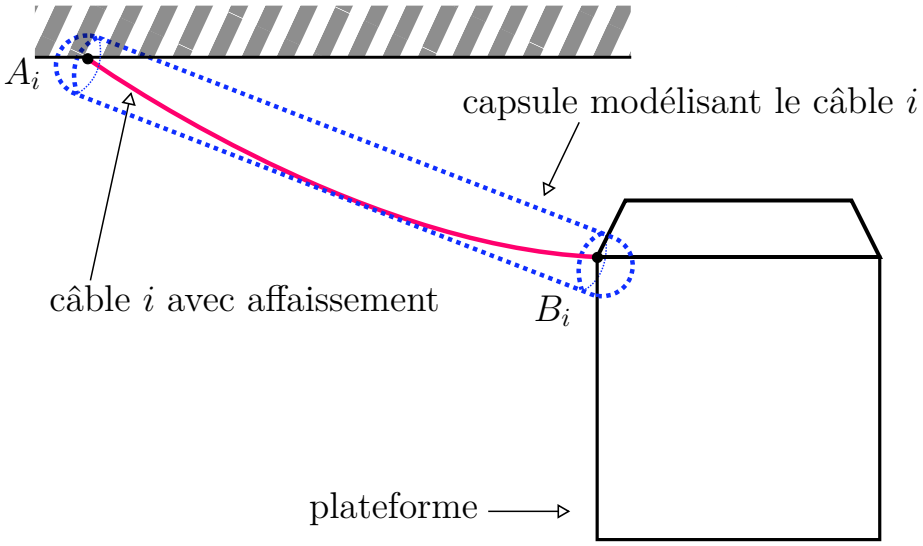


Figure 3.9 – Schéma d'un câble affaissé et de sa modélisation en capsule.

sur la plateforme mobile est également supposé fixe. À chaque changement de la configuration du robot (mouvement de la plateforme), la position absolue du point d'attache du câble sur la plateforme est recalculée. Le point de sortie sur la structure étant fixe, nous représentons le câble comme une capsule géométrique de rayon fixe reliant ces deux points. Une capsule est l'union d'un cylindre et d'une demi-sphère de même rayon à chaque extrémité. La modélisation d'un câble par une capsule est visible sur la figure 3.9.

Un câble n'est par définition pas rigide. L'effet de la gravité le long du câble peut mener à son affaissement : le câble n'est plus un segment droit mais prend la forme d'une chaînette élastique. Dans la suite de cette thèse, l'affaissement et l'élasticité des câbles ne sont pas considérés et nous considérons que chaque câble est *idéal*.

3.2.3.b Représentation des câbles dans les fichiers SRDF

La gestion des câbles a été ajoutée dans le logiciel HPP. Les câbles sont gérés différemment que les autres corps rigides du robot : ils ne sont pas spécifiés dans le

```
<cable>

    <structure_exit_point>
        <xyz> -7 -5 5 </xyz>
        <link name="base_link" />
    </structure_exit_point>
    <platform_attachment_point>
        <xyz> 0.5 -0.5 0 </xyz>
        <link name="platform_link" />
    </platform_attachment_point>
    <visual diameter="0.02"/>
    <collision diameter="0.01"/>

</cable>
```

Figure 3.10 – Extrait d'un fichier SRDF définissant un câble

fichier URDF, mais dans le fichier SRDF. Le fichier SRDF d'un RPC contient ainsi les informations nécessaires à la modélisation de chaque câble : la position du point de sortie de la structure fixe, la position du point d'attache sur la plateforme mobile, le rayon visuel et le rayon de collision du câble. Il est en effet possible de définir un rayon d'affichage des câbles lors de la visualisation différent du rayon utilisé pour les tests de collision.

La figure 3.10 présente un extrait d'un fichier SRDF. Pour le point de sortie de la structure fixe, le nom du corps de la structure tel que défini dans le fichier URDF est indiqué. Il en est de même pour le point d'attache à la plateforme mobile.

Grâce à cette modélisation, les câbles peuvent être affichés dans le visualisateur de HPP. La figure 3.2 montre la visualisation du robot CoGiRo.

3.3 Détection statique de collision de câbles

3.3.1 Gestion des paires de corps pour la collision

Dans HPP, toutes les paires de corps sont testées afin de détecter d'éventuelles collisions, sauf les paires précisées dans le fichier SRDF. La détection de collision entre un corps parent et un corps enfant, c'est-à-dire entre deux corps consécutifs dans la chaîne cinématique, est souvent désactivée. Par exemple, la détection de collision entre l'avant-bras et le poignet d'un robot humanoïde est désactivée. Cela s'explique par le fait que deux pièces consécutives sont souvent en contact à leur articulation commune. De plus, elles ne présentent généralement pas d'autre risque de collision entre elles, grâce à leur limites articulaires : un robot humanoïde ne pourra pas plier le poignet suffisamment pour que celui-ci entre en collision avec l'avant-bras. Dans HPP, les robots sont modélisés de façon à ce que cette dernière condition soit satisfaite. Si elle ne l'est pas, il suffit de séparer le corps enfant en deux corps distincts. Pour désactiver une paire de corps, la commande `disable_collision` est utilisée dans le fichier SRDF du robot, comme détaillé dans la section 3.2.2.b.

Dans un RPC, chaque câble est en contact avec la structure fixe et la plateforme mobile. Pour pouvoir utiliser le test malgré tout, les paires câble-plateforme et câble-structure ne sont pas désactivée. La longueur de la capsule géométrique représentant le câble est tronquée aux extrémités, d'une distance fixe — pour CoGiRo, cette distance est de l'ordre de 10 cm. Avec une capsule ainsi raccourcie, le test de collision ne considère pas le câble et la plateforme en collision par défaut.

3.3.2 Test statique de collision

HPP possède un test de collision statique qui permet de déterminer la validité d'une configuration d'un robot par rapport aux collisions. Au chargement d'un robot dans HPP, la liste des paires de corps du robot est générée. Les paires de corps désactivée pour la collision sont exclus de la liste. Lors de chaque appel du test de collision statique, chaque paire de la liste est testée. Le test de collision pour une paire de corps est effectué avec la librairie FCL [131]. Chaque corps peut être composé de plusieurs pièces physiques, chacune représentée par un mesh 3D ou une forme géométrique simple, avec sa position et son orientation. La libriaire FCL permet de calculer la distance entre les paires de pièces physiques de corps différents, et ainsi de déterminer si ces corps sont en collision.

Dans FCL, les calculs de collision et de distance entre les polytopes convexes se basent sur les algorithmes GJF (Gilbert-Johnson-Keerthi) [135] et EPA (Expanding Polytope Algorithm) [136]. Les meshes 3D sont gérés par des méthodes de hiérarchie de volumes englobants (BVH, Bounding Volume Hierarchy). Plusieurs méthodes de BHV sont gérées dans FCL, comme la méthode de boîte englobante alignée sur les axes (AABB, Axis-Aligned Bounding Box) [137], la méthode de boîte englobante orientée (OBB, Oriented Bounding Box) [138], ou la méthode de polytope discret orienté (k-DOP, Discrete Oriented Polytope) [139].

Dès qu'une paire est en collision, le test s'arrête et retourne la paire en collision. Si aucune paire n'est en collision, la configuration est jugée valide par rapport aux collisions. Ce test a été étendu pour prendre en compte les câbles nouvellement ajoutés dans le logiciel. Avec p le nombre de corps dans l'environnement et m le nombre de câbles, les différentes paires de corps incluant des câbles sont les paires :

- câble-câble — il y a $\binom{m}{2}$ paires de câbles, où m est le nombre de câbles.
- plateforme-câble — il y a m paires plateforme-câble.
- environnement-câble — il y a $m \times p$ paires environnement-câble.

Les câbles sont représentés par des capsules géométriques, dont la longueur, la position et l'orientation sont recalculés à chaque pas de temps. Le rayon est fixe et choisi en amont dans le fichier SRDF.

3.3.3 Validation discrète de trajectoire vis-à-vis des collisions

HPP possède plusieurs tests de validation de trajectoires vis-à-vis de la collision : un test discrétisé et des tests continus. Lors du test discrétisé, un pas de temps est choisi, la trajectoire est échantillonnée grâce à ce pas de temps et les configurations obtenues sont testées avec le test statique de collision. Si toutes les configurations

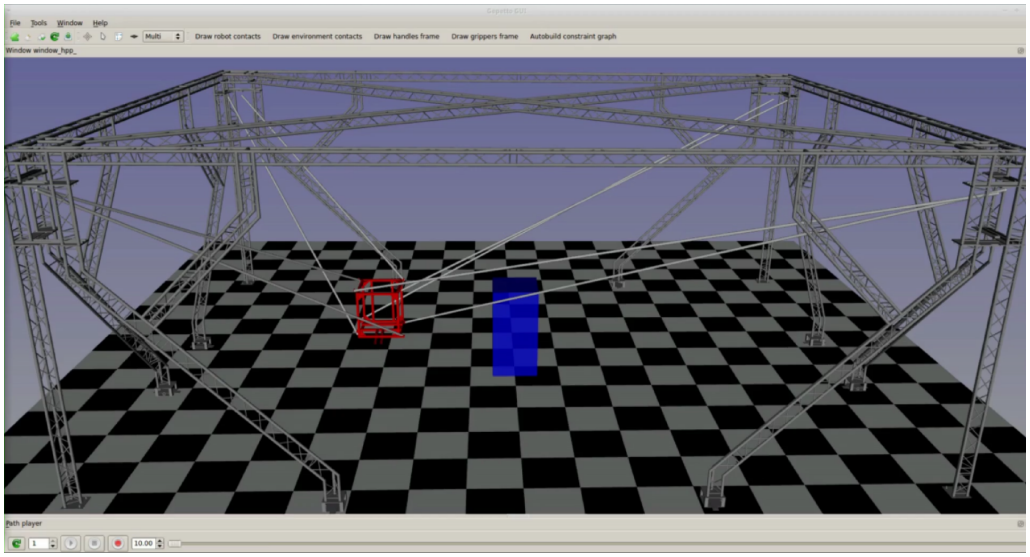


Figure 3.11 – Visualisation sous HPP d'un cas de test simple pour les collisions, avec un obstacle en forme de boîte

sont valides, la trajectoire est validée. Dans le cas contraire, l'algorithme renvoie la portion du chemin validée jusqu'à la collision.

L'efficacité de cette méthode dépend du pas de temps choisi : un pas de temps petit trouvera des collisions qui peuvent passer inaperçu pour un pas de temps plus grand, mais avec un temps de calcul plus élevé. Le choix du pas de temps n'est pas évident et il est toujours possible que certaines collisions ne soient pas détectées, même avec un petit pas de temps.

La figure 3.11 montre un exemple de cas simple pour tester la validation de trajectoire vis-à-vis des collisions.

3.4 Validation continue

Cette partie s'attache à calculer les éléments nécessaire à la validation continue vis-à-vis des collisions. Pour une paire de corps, nous définissons sont élément de validation continue (selon la définition donnée dans la Section 2.2.2) comme étant la fonction de distance signée entre ces deux corps.

Chaque section s'attache à un cas particulier, et détaille les méthodes utilisées pour obtenir le minorant et le majorant nécessaire à la détermination d'un intervalle valide, selon la méthode exposée dans la Section 2.3.

Dans la suite, nous désignerons par *corps solide* tout élément du robot ou de l'environnement qui n'est pas un câble. Notamment, dans le cas RPC équipé d'un bras robotique, les différents corps du bras robotique et la plateforme mobile sont appelés des corps solides. Nous traitons tout d'abord le cas de deux corps solides dans la Section 3.4.1 (cas qui peut s'appliquer aux RPCs comme aux autres robots), puis les cas qui impliquent des câbles. Le cas d'une paire "câble et corps solide" est divisé en deux sous-cas : le cas "câble et plateforme mobile" (Section 3.4.2) et le cas "câble et corps du bras robotique" (Section 3.4.4).

3.4.1 Collision entre deux corps solides

Les parties suivantes s'intéressent aux calculs des valeurs $D^{\min}$ (minorant de la distance) et $V^{\max}$ (majorant de la vitesse relative), définies dans la section 2.3, pour deux corps solides d'un robot. Si la paire est composée de deux corps successifs dans la chaîne cinématique — par exemple, le poignet et la main pour un robot humanoïde — alors le test de collision est désactivé. Pour un RPC, cette méthode s'applique aux paires impliquant la plateforme mobile, les parties robotiques et les objets qui y sont éventuellement attachés, la structure fixe et l'environnement.

3.4.1.a Majorant de la vitesse relative de deux corps solides

Notons $\mathcal{B}_a$ et $\mathcal{B}_b$ les deux corps de la paire, et J_a et J_b les articulations de ces corps. Nous considérons pour chaque problème de planification que l'environnement et les objets à manipuler font également partie du robot ; de cette manière, le problème se compose d'un seul robot. Il existe donc entre les corps $\mathcal{B}_a$ et $\mathcal{B}_b$ une liste d'articulations les reliant. Notons $J_0 = J_a$, J_1, …, J_{k-1}, $J_k = J_b$ les articulations successives de J_a vers J_b.

Nous supposons que le chemin à valider est un *chemin direct*, ce qui signifie que le long du chemin, chaque articulation J_i bouge en rotation et/ou en translation à une vitesse angulaire et/ou linéaire constante dans le repère global. Pour chaque articulation $i \in [1, k-1]$, nous notons :

- ${}^{j_i}\mathbf{v}_{j_{i-1}}$ la vitesse linéaire constante de l'articulation J_{i-1} dans le repère de l'articulation J_i, de norme ${}^{j_i}v_{j_{i-1}}$,
- ${}^{j_i}\boldsymbol{\omega}_{j_{i-1}}$ la vitesse angulaire constante de l'articulation J_{i-1} dans le repère de l'articulation J_i, de norme ${}^{j_i}\omega_{j_{i-1}}$.

Soit P_0 un point fixe dans le référentiel de J_0. Notons $\mathbf{m}_0$ ses coordonnées dans le repère de J_0. Les coordonnées de P_0 dans le repère de l'articulation J_{k-1} s'écrivent :

$$ {}^{k-1}P_0 = {}^{j_{k-1}}M_{j_{k-2}}\,{}^{j_{k-2}}M_{j_{k-3}} \ \dots\ {}^{j_2}M_{j_1}\,{}^{j_1}M_{j_0} \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix} \tag{3.2} $$

où :

- ${}^{j_{i+1}}M_{j_i} = \begin{pmatrix} {}^{j_{i+1}}R_{j_i} & {}^{j_{i+1}}T_{j_i} \\ 0\ 0\ 0 & 1 \end{pmatrix}$ désigne la matrice homogène définissant la position et l'orientation de l'articulation J_i dans le repère de l'articulation J_{i+1},
- ${}^{j_{i+1}}R_{j_i} \in SO(3)$ est une matrice de rotation,
- ${}^{j_{i+1}}T_{j_i} \in \mathbb{R}^3$ est un vecteur de translation.

En différenciant (3.2) par rapport au temps, nous obtenons :

$$
\begin{aligned}
\begin{pmatrix} {}^{k-1}\dot{P}_0 \\ 0 \end{pmatrix} = &\begin{pmatrix} \left[{}^{j_{k-1}}\boldsymbol{\omega}_{j_{k-2}}\right]_\times {}^{j_{k-1}}R_{j_{k-2}} & {}^{j_{k-1}}\mathbf{v}_{j_{k-2}} \\ 0\ 0\ 0 & 0 \end{pmatrix} \dots \\
&\qquad \dots {}^{j_1}M_{j_0} \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix} \\
&+ {}^{j_{k-1}}M_{j_{k-2}} \begin{pmatrix} \left[{}^{j_{k-2}}\boldsymbol{\omega}_{j_{k-3}}\right]_\times {}^{j_{k-2}}R_{j_{k-3}} & {}^{j_{k-2}}\mathbf{v}_{j_{k-3}} \\ 0\ 0\ 0 & 0 \end{pmatrix} \dots \\
&\qquad \dots {}^{j_1}M_{j_0} \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix} \\
&+ \dots \\
&+ {}^{j_{k-1}}M_{j_{k-2}} \dots {}^{j2}M_{j_1} \begin{pmatrix} \left[{}^{j_1}\boldsymbol{\omega}_{j_0}\right]_\times {}^{j_1}R_{j_0} & {}^{j_1}\mathbf{v}_{j_0} \\ 0\ 0\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{m}_0 \\ 1 \end{pmatrix}
\end{aligned}
\tag{3.3}
$$

Tel que précisé dans la section 3.2.1, la norme vectorielle utilisée est la norme euclidienne et la norme matricielle est la norme induite par la norme vectorielle euclidienne. En utilisant les deux propriétés des transformations de corps rigides présentées dans l'annexe 6.5, nous pouvons borner la norme :

$$
\begin{aligned}
\|{}^{k-1}\dot{P}_0\| \leq &\|{}^{j_1}\mathbf{v}_{j_0}\| + \|{}^{j_1}\boldsymbol{\omega}_{j_0}\|\|\mathbf{m}_0\| \\
&+ \|{}^{j_2}\mathbf{v}_{j_1}\| + \|{}^{j_2}\boldsymbol{\omega}_{j_1}\|(\,\|\mathbf{m}_0\| + \|{}^{j_1}T_{j_0}\|\,) \\
&+ \|{}^{j_3}\mathbf{v}_{j_2}\| + \|{}^{j_3}\boldsymbol{\omega}_{j_2}\|(\,\|\mathbf{m}_0\| + \|{}^{j_1}T_{j_0}\| + \|{}^{j_2}T_{j_1}\|\,) \\
&+ \dots \\
&+ \|{}^{j_{k-1}}\mathbf{v}_{j_{k-2}}\| + \|{}^{j_{k-1}}\boldsymbol{\omega}_{j_{k-2}}\|(\,\|\mathbf{m}_0\| + \|{}^{j_1}T_{j_0}\| + \dots + \|{}^{j_{k-2}}T_{j_{k-3}}\|\,)
\end{aligned}
\tag{3.4}
$$

Considérons le rayon r_0 du corps $\mathcal{B}_a$ comme étant le maximum des distances des points du corps à l'origine du repère de l'articulation :

$$
r_0 = \max_{\mathbf{m}_0 \in \mathcal{B}_a} \|\mathbf{m}_0\| \tag{3.5}
$$

Notons D_k la longueur cumulée de l'articulation J_k:

$$
\begin{aligned}
D_0 &= 0 \\
D_k &= \sum_{t=0}^{k-1} \|{}^{j_{t+1}}T_{j_t}\| \quad \text{pour } k \geq 1
\end{aligned}
\tag{3.6}
$$

En intégrant (3.5) et (3.6) dans (3.4), nous obtenons la formule suivante qui donne donc un majorant de la vitesse relative de deux corps solides du robot :

$$
\|{}^{k-1}\dot{P}_0\| \leq \sum_{k=0}^{k-2} {}^{j_{k+1}}v_{j_k} + {}^{j_{k+1}}\omega_{j_k}(\,r_0 + D_k\,) = V^{\max} \tag{3.7}
$$

Les variables notées en rouge dépendent du chemin à valider. Les variables en bleu sont des variables constantes dépendant de la géométrie du robot.

3.4.1.b Minorant de la distance entre deux corps solides

Comme pour le test statique de collision (section 3.3.2), la bibliothèque FCL [131] est utilisée. En plus de pouvoir déterminer si deux corps sont en collision, FCL permet de calculer un minorant $D^{\min}$ de la distance entre les deux corps, grâce notamment aux algorithme de volumes englobants.

3.4.2 Collision entre un câble et la plateforme mobile

Comme expliqué dans la section 3.3.1, nous avons choisi de considérer une version raccourcie du câble qui s'arrête à une distance fixe d du point d'attache du câble B_i sur la plateforme. Nous calculons $V^{\max}$, une limite supérieure de la vitesse de n'importe quel point du câble par rapport à la plateforme, ainsi que $D^{\min}$, une limite inférieure de la distance entre le câble raccourci et la plateforme. Les paragraphes suivant détaillent ces calculs.

3.4.2.a Majorant de la vitesse relative entre un câble et la plateforme

Nous considérons un *chemin direct*, pour lequel les vitesses linéaires et angulaires de la plateforme sont constantes et connues. Elles sont respectivement notées $\mathbf{v}_p$ et $\boldsymbol{\omega}_p$, et nous notons les normes $\omega_p = \|\boldsymbol{\omega}_p\|$ et $v_p = \|\mathbf{v}_p\|$. Le point C est l'origine du repère de la plateforme. L'objectif est de calculer un majorant des vitesses de tous les points du câble i dans le repère de la plateforme.

Soit P_i le point sur le câble i à une distance fixe r de B_i. La relation entre le vecteur de coordonnées ${}^pP_i$ de P_i dans le repère de la plateforme et son vecteur de coordonnées ${}^iP_i$ dans le repère du câble $\mathscr{F}_i$ est donnée par la formule suivante :

$$\begin{pmatrix} {}^pP_i \\ 1 \end{pmatrix} = {}^pM_i \begin{pmatrix} {}^iP_i \\ 1 \end{pmatrix} \tag{3.8}$$

où :

- ${}^pM_i = \begin{pmatrix} {}^pR_i & {}^pT_i \\ 0\ 0\ 0 & 1 \end{pmatrix}$ est la matrice homogène représentant la position et l'orientation du repère local du câble dans le repère de la plateforme,
- ${}^pR_i \in SO(3)$ est une matrice de rotation,
- ${}^pT_i = \overrightarrow{CB_i} = \mathbf{b}_i \in \mathbb{R}^3$ est la position du point d'attache B_i dans le repère de la plateforme ; la norme de $\mathbf{b}_i$ est notée b_i.

Puisque B_i est fixe dans le repère de la plateforme, ${}^pT_i$ est constante. Sachant que ${}^iP_i = (r\ 0\ 0)^T$ est constant, en différenciant (3.8) par rapport au temps, on obtient :

$$\begin{pmatrix} {}^p\dot{P}_i \\ 0 \end{pmatrix} = \begin{pmatrix} [{}^p\boldsymbol{\omega}_i]_\times {}^pR_i & 0 \\ 0\ 0\ 0 & 0 \end{pmatrix} \begin{pmatrix} {}^iP_i \\ 1 \end{pmatrix} \tag{3.9}$$

où

- ${}^p\dot{P}_i$ est la vitesse du point P_i dans le repère de la plateforme,
- $[{}^p\boldsymbol{\omega}_i]_\times$ est la matrice antisymétrique correspondant au produit vectoriel avec le vecteur de vitesse angulaire du repère du câble ${}^p\boldsymbol{\omega}_i \in \mathbb{R}^3$ dans le repère de la plateforme, de norme ${}^p\omega_i$.

On obtient :

$$ {}^{p}\dot{P}_i = [{}^{p}\boldsymbol{\omega}_i]_\times {}^{p}R_i {}^{i}P_i \tag{3.10} $$

Par conséquent, nous pouvons borner la norme :

$$ \|{}^{p}\dot{P}_i\| \leq {}^{p}\omega_i \| {}^{i}P_i \| \tag{3.11} $$

Par composition des vitesses angulaires, nous avons ${}^{p}\boldsymbol{\omega}_i = {}^{0}\boldsymbol{\omega}_i + {}^{p}\boldsymbol{\omega}_0$, avec ${}^{p}\boldsymbol{\omega}_0 = -{}^{0}\boldsymbol{\omega}_p = -\boldsymbol{\omega}_p$. En bornant la norme, nous obtenons ${}^{p}\omega_i \leq {}^{0}\omega_i + \omega_p$.

Appliquons la loi de composition des vitesses entre le repère $\mathscr{F}_i$ et le repère $\mathscr{F}_0$, au point A_i :

$$ \underbrace{{}^{0}\dot{A}_i}_{=0} = \underbrace{{}^{i}\dot{A}_i}_{=\dot{L}_i \mathbf{x}_i} + \underbrace{\mathbf{v}_i}_{=\mathbf{v}_{B_i}} + {}^{0}\boldsymbol{\omega}_i \times \overrightarrow{B_i A_i} \tag{3.12} $$

D'après (3.1), $\|{}^{0}\boldsymbol{\omega}_i \times \overrightarrow{B_i A_i}\| = {}^{0}\omega_i \| \overrightarrow{A_i B_i} \|$. En utilisant (3.12) et l'inégalité triangulaire, on peut exprimer l'inégalité suivante sur la norme de ${}^{0}\boldsymbol{\omega}_i$:

$$ {}^{0}\omega_i \leq \frac{v_{B_i} + \dot{L}_i}{L_i} \tag{3.13} $$

Cherchons à exprimer $\dot{L}_i$. Puisque $L_i = \|\overrightarrow{A_i B_i}\|$, alors $\dot{L}_i = \frac{1}{L_i}(\overrightarrow{A_i B_i})^T(\dot{\overrightarrow{A_i B_i}})$. Comme précisé dans la section 3.2.1, nous supposons que pour chaque câble i, nous connaissons un majorant et un minorant de la longueur $L_i = \|\overrightarrow{A_i B_i}\|$ du câble i, respectivement $L_i^{\max}$ et $L_i^{\min}$. Vu que A est immobile, nous obtenons :

$$ |\dot{L}_i| \leq \frac{L_i^{\max}}{L_i^{\min}} v_{B_i} \tag{3.14} $$

Puisque B_i est fixe dans le repère de la plateforme, qui a des vitesses linéaire et angulaire constantes $\mathbf{v}_p$ et $\boldsymbol{\omega}_p$, B_i se déplace avec une vitesse linéaire $\mathbf{v}_{B_i}$ de norme $v_{B_i} \leq v_p + \omega_p b_i$. De plus, B_i tourne autour du point fixe A_i, donc $\mathbf{v}_{B_i} = -{}^{0}\boldsymbol{\omega}_i \times \overrightarrow{A_i B_i}$ et d'après la définition du repère du câble $\mathscr{F}_i$ donnée dans la section 3.2.1, ${}^{0}\boldsymbol{\omega}_i$ est orthogonal à $\overrightarrow{A_i B_i}$, donc $v_{B_i} = L_i {}^{0}\omega_i$. Puisque $\|\overrightarrow{A_i B_i}\| \geq L_i^{\min}$, on obtient (3.15) et ensuite (3.16).

$$ {}^{0}\omega_i \leq \left(1 + \frac{L_i^{\max}}{L_i^{\min}}\right) \frac{v_p + \omega_p b_i}{L_i^{\min}} \tag{3.15} $$

$$ {}^{p}\omega_i \leq \omega_p + \left(1 + \frac{L_i^{\max}}{L_i^{\min}}\right) \frac{v_p + \omega_p b_i}{L_i^{\min}} \tag{3.16} $$

En utilisant (3.11) et (3.16), et sachant que nous avons également un majorant sur $\|{}^{i}P_i\|$ qui est $L_i^{\max}$, nous obtenons :

$$ \|{}^{p}\dot{P}_i\| \leq L_i^{\max} \left(\omega_p + \left(1 + \frac{L_i^{\max}}{L_i^{\min}}\right) \frac{v_p + \omega_p b_i}{L_i^{\min}} \right) = V^{\max} \tag{3.17} $$

L'équation (3.17) donne un majorant à la vitesse relative entre le câble i et la plateforme mobile le long d'un *chemin direct*. De même que précédemment, les variables notées en rouge dépendent du chemin à valider tandis que les variables en bleu sont des variables constantes dépendant de la géométrie du robot.

3.4.2.b Minorant de la distance entre un câble et la plateforme mobile

La bibliothèque FCL est utilisée pour calculer un minorant $D^{\min}$ de la distance entre un câble i et la plateforme mobile. La plateforme est représentée par un modèle de collision simplifié par rapport au modèle visuel, tandis que le câble est représenté par une capsule. Comme le câble et la plateforme sont connectés au point B_i, comme indiqué ci-dessus, nous considérons une portion raccourcie du cylindre $A_i\tilde{B}_i$ où $B_i\tilde{B}_i = d\frac{\overrightarrow{B_iA_i}}{\|\overrightarrow{B_iA_i}\|}$. d est une distance fixe choisie de sorte que pour toute configuration au sein d'un espace de travail prescrit, si un point de $[B_i\tilde{B}_i]$ est en collision avec la plateforme, alors au moins un point de $[\tilde{B}_iA_i]$ est également en collision avec la plateforme. Cette valeur d peut être trouvée par essais et erreurs en testant la présence de collisions pour une liste de configurations données qui sont censées être valides. De par sa conception, $D^{\min}$ ne sera jamais supérieure à d, et il faut noter que le point le plus proche entre la plateforme et le câble sera généralement l'extrémité (virtuelle) du câble $\tilde{B}$, c'est-à-dire $D^{\min} = d$. Choisir d aussi grand que possible réduit le nombre d'itérations nécessaires pour valider un intervalle, et donc le temps de calcul.

3.4.3 Collision entre deux câbles

Si deux câbles ont les mêmes points d'attache sur la plateforme ou les mêmes points de sortie sur la structure de base, le contrôle de collision entre eux est désactivé car ces deux câbles ne peuvent pas entrer en collision. Pour toutes les autres paires de câbles, une limite supérieure de vitesse $V^{\max}$ et une limite inférieure de distance $D^{\min}$ sont calculées.

3.4.3.a Majorant de la vitesse relative entre deux câbles

Nous devons trouver un majorant des vitesses de tous les points du câble i dans le repère local du câble j. Soit P_i le point sur le câble i à une distance fixe r de B_i, avec ${}^jP_i$ son vecteur de coordonnées dans le repère local du câble j. Comme pour les calculs de la section 3.4.2, nous pouvons écrire :

$$\begin{pmatrix} {}^jP_i \\ 1 \end{pmatrix} = {}^jM_i \begin{pmatrix} {}^iP_i \\ 1 \end{pmatrix} \tag{3.18}$$

où ${}^jM_i = \begin{pmatrix} {}^jR_i & {}^jT_i \\ 0\ 0\ 0 & 1 \end{pmatrix}$ désigne la matrice homogène définissant la position et l'orientation du repère local du câble i par rapport au repère local du câble j.
Notons ${}^j\boldsymbol{\omega}_i$ le vecteur de vitesse angulaire, de norme ${}^j\omega_i$, du repère du câble i par rapport au repère du câble j. Le câble étant attaché à la plateforme à l'origine du repère du câble, la vitesse de translation du câble i par rapport à la plateforme est nulle. En différenciant par rapport au temps, et en sachant que ${}^iP_i = (r\ 0\ 0)^T$ et ${}^jT_i = \overrightarrow{B_jB_i}$ sont constants :

$$\begin{pmatrix} {}^j\dot{P}_i \\ 0 \end{pmatrix} = \begin{pmatrix} [{}^j\boldsymbol{\omega}_i]_\times {}^jR_i & 0 \\ 0\ 0\ 0 & 0 \end{pmatrix} \begin{pmatrix} {}^iP_i \\ 0 \end{pmatrix} \tag{3.19}$$

L'équation (3.19) implique que ${}^j\dot{P}_i = [{}^j\boldsymbol{\omega}_i]_\times {}^jR_i {}^iP_i$, ce qui donne :

$$\|^j\dot{P}_i\| \leq {}^j\omega_i\|^iP_i\| \tag{3.20}$$

Nous avons ${}^j\boldsymbol{\omega}_i = {}^0\boldsymbol{\omega}_i - {}^0\boldsymbol{\omega}_j$. Rappelons que nous avons supposé que pour chaque câble i, nous connaissons un majorant et un minorant de la longueur du câble, respectivement $L_i^{\max}$ et $L_i^{\min}$.

En utilisant (3.15) et en sachant que $L_i^{\max}$ est un majorant de $\|^iP_i\|$, nous obtenons la formule suivante définissant un majorant de la vitesse relative entre deux câbles :

$$V^{\max} = L_i^{\max}\left(\left(1+\frac{L_i^{\max}}{L_i^{\min}}\right)\frac{v_p+\omega_p b_i}{L_i^{\min}} + \left(1+\frac{L_j^{\max}}{L_j^{\min}}\right)\frac{v_p+\omega_p b_j}{L_j^{\min}}\right) \tag{3.21}$$

Une fois de plus, les variables notées en rouge dépendent du chemin à valider tandis que les variables en bleu sont des variables constantes dépendant de la géométrie du robot.

3.4.3.b Minorant de la distance entre deux câbles

Nous utilisons de nouveau la bibliothèque FCL pour calculer un minorant de la distance entre deux câbles. Chaque câble est représenté par une capsule géométrique. Connaissant les longueurs, rayons, positions et orientations des capsules, FCL calcule la distance exacte entre les capsules. Pour deux câbles, la distance entre leurs points de fixation sur la plateforme est un majorant fixe de la distance entre les câbles et donc de $D^{\min}$. De même, la distance entre les points de sortie de la structure fixe est un majorant de $D^{\min}$. Cela signifie que la méthode proposée est plus lente pour un RPC avec des câbles fixés à proximité l'un de l'autre sur la plateforme ou la structure que pour un RPC avec des câbles fixés loin l'un de l'autre. En effet, si les points d'attache ou de sortie des câbles sont proches, $D^{\min}$ est faible et l'algorithme ne peut valider que de petites portions du trajet.

3.4.4 Collision entre un câble et un corps du bras robotique

Considérons l'élément de validation suivant : une paire composée d'un câble et d'un corps du bras robotique attaché à la plateforme mobile. Dans le cas où le bras robotique attrape un objet lors d'un scénario de manipulation, l'objet, quand il est attrapé, est considéré comme un corps du bras robotique.

3.4.4.a Majorant de la vitesse relative entre un câble et un corps du bras robotique

Les calculs pour ce cas sont similaires aux calculs effectués pour le majorant de la vitesse relative entre deux corps solides dans le section 3.4.1. Considérons un câble i et un corps $\mathcal{B}_a$ attaché à l'articulation J_a. Notons $J_0 = J_p$, J_1, ..., $J_{k-1} = J_a$ les articulations successives reliant l'articulation J_p de la plateforme mobile à l'articulation J_a. J_1 est donc l'articulation attachant le corps $\mathcal{B}_a$ à la plateforme. Considérons un point P_i le long du câble i de coordonnées ${}^aP_i$ dans le repère de l'articulation J_a.

$$\begin{pmatrix} {}^{a}P_i \\ 1 \end{pmatrix} = {}^{j_{k-1}}M_{j_{k-2}}\,{}^{j_{k-2}}M_{j_{k-3}} \ldots {}^{j_1}M_p\,{}^{p}M_i \begin{pmatrix} {}^{i}P_i \\ 1 \end{pmatrix} \tag{3.22}$$

où ${}^{j_{i+1}}M_{j_i} = \begin{pmatrix} {}^{j_{i+1}}R_{j_i} & {}^{j_{i+1}}T_{j_i} \\ 0\ 0\ 0 & 1 \end{pmatrix}$ désigne la matrice homogène définissant la position et l'orientation de l'articulation J_i dans le repère de l'articulation J_{i+1}.
En différenciant (3.22) par rapport au temps, nous obtenons :

$$\begin{aligned} \begin{pmatrix} \dot{P}_i^a \\ 0 \end{pmatrix} = & \begin{pmatrix} \left[{}^{j_{k-1}}\boldsymbol{\omega}_{j_{k-2}}\right]_\times {}^{j_{k-1}}R_{j_{k-2}} & {}^{j_{k-1}}\mathbf{v}_{j_{k-2}} \\ 0\ 0\ 0 & 0 \end{pmatrix} \cdots \\ & \qquad \cdots {}^{j_1}M_{j_0}\,{}^{p}M_i \begin{pmatrix} {}^{i}P_i \\ 1 \end{pmatrix} \\ & + {}^{j_{k-1}}M_{j_{k-2}} \begin{pmatrix} \left[{}^{j_{k-2}}\boldsymbol{\omega}_{j_{k-3}}\right]_\times {}^{j_{k-2}}R_{j_{k-3}} & {}^{j_{k-2}}\mathbf{v}_{j_{k-3}} \\ 0\ 0\ 0 & 0 \end{pmatrix} \cdots \\ & \qquad \cdots {}^{j_1}M_{j_0}\,{}^{p}M_i \begin{pmatrix} {}^{i}P_i \\ 1 \end{pmatrix} \\ & + \cdots \\ & + {}^{j_{k-1}}M_{j_{k-2}} \cdots {}^{j1}M_{j_0} \begin{pmatrix} \left[{}^{p}\boldsymbol{\omega}_i\right]_\times {}^{p}R_i & {}^{p}\mathbf{v}_i \\ 0\ 0\ 0 & 0 \end{pmatrix} \begin{pmatrix} {}^{i}P_i \\ 1 \end{pmatrix} \end{aligned} \tag{3.23}$$

En utilisant les propriétés des transformations de corps rigides, nous pouvons borner la norme de $\dot{P}_i^a$:

$$\begin{aligned} \|{}^{a}\dot{P}_i\| \leq\ & \|{}^{p}\mathbf{v}_i\| + \|{}^{p}\boldsymbol{\omega}_i\|\|{}^{i}P_i\| \\ & + \|{}^{j_1}\mathbf{v}_{j_0}\| + \|{}^{j_1}\boldsymbol{\omega}_{j_0}\|(\,\|{}^{i}P_i\| + \|{}^{p}T_i\|\,) \\ & + \|{}^{j_2}\mathbf{v}_{j_1}\| + \|{}^{j_2}\boldsymbol{\omega}_{j_1}\|(\,\|{}^{i}P_i\| + \|{}^{p}T_i\| + \|{}^{j1}T_{j_0}\|\,) \\ & + \ldots \\ & + \|{}^{j_{k-1}}\mathbf{v}_{j_{k-2}}\| + \|{}^{j_{k-1}}\boldsymbol{\omega}_{j_{k-2}}\|(\,\|{}^{i}P_i\| + \|{}^{p}T_i\| + \|{}^{j1}T_{j_0}\| + \cdots + \|{}^{j_{k-2}}T_{j_{k-3}}\|\,) \end{aligned} \tag{3.24}$$

Comme dans la section 3.4.1, notons D_k la longueur cumulée de l'articulation J_k :

$$\begin{aligned} D_0 &= 0 \\ D_k &= \sum_{t=0}^{k-1} \|{}^{j_{t+1}}T_{j_t}\| \quad \text{pour } k \geq 1 \end{aligned} \tag{3.25}$$

Considérons de nouveau le majorant $L_i^{\max}$ de la longueur du câble i. $L_i^{\max}$ est donc un majorant de $\|{}^{i}P_i\|$. De plus, l'origine B_i du repère local du câble est immobile dans le repère de la plateforme, de sorte que ${}^{p}v_i = 0$. En utilisant (3.16), (3.24) et (3.25), nous obtenons un majorant sur la vitesse de tous les points du câble i dans le repère de l'articulation J_a :

$$V^{\max} = \left(\omega_p + \frac{v_p + \omega_p b_i}{L_i^{\min}}\right) L_i^{\max} + \sum_{t=0}^{k-2} \left({}^{j_{t+1}}v_{j_t} + {}^{j_{t+1}}\omega_{j_t}(L_i^{\max} + b_i + D_t)\right) \tag{3.26}$$

Une fois de plus, les variables notées en rouge dépendent du chemin à valider tandis que les variables en bleu sont des variables constantes dépendant de la géométrie du robot.

3.4.4.b Minorant de la distance entre un câble et un corps du bras robotique

La bibliothèque FCL est à nouveau utilisée pour calculer une limite inférieure de la distance entre un câble, représenté par une capsule, et le corps du bras robotique, représenté par un mesh 3D.

3.5 Résultats d'implémentation

Nous avons mis en œuvre la méthode de détection continue des collisions proposée dans le logiciel HPP. Le RPC CoGiRo est simulé dans HPP. La méthode continue est comparée à une méthode de contrôle de collision discrétisée, qui utilise une méthode de validation statique et un pas de temps τ.

L'ordinateur utilisé pour exécuter le logiciel HPP est un « Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz » avec 4 096 Ko de mémoire cache et 16 Go de RAM.

Ce benchmark est réalisé en simulation sur le robot CoGiRo sur lequel est fixé le bras robotique Yaskawa SIA20. Des *chemins directs* aléatoires sont générés en tirant des configurations aléatoires dans l'espace de configuration, et en gardant les deux premières configurations valides. Chaque *chemin direct* est validé en utilisant la méthode continue et la méthode discrétisée avec différents pas de temps. Les résultats sont classés en cinq catégories :

- Vrai positif : une collision a été détectée à la fois par la méthode discrétisée et la méthode continue.
- Vrai négatif : aucune collision n'a été détectée ni par la méthode discrétisée ni par la méthode continue.
- Nouveau vrai positif : une collision réelle a été trouvée par la méthode continue, mais n'a pas été détectée par la méthode discrétisée.
- Faux positif : une collision a été détectée par la méthode continue mais n'est pas une vraie collision..
- Faux négatif : la méthode continue n'a pas détecté une collision qui a été trouvée par la méthode discrétisée.

Les résultats sont repris dans les tableaux 3.1 et 3.2 avec différents pas de temps testés pour la méthode discrétisée. Plus le pas de temps est petit, plus le temps de calcul moyen est grand pour la méthode discrétisée.

Comme le montre le tableau 3.1, il n'y a pas de faux positif, grâce au fait que la bibliothèque FCL ne renvoie zéro comme minorant de la distance entre deux objets que s'ils sont effectivement en collision. La méthode continue est garantie de trouver une collision s'il en existe une, il n'y a donc pas de faux négatif. Les nouveaux vrais

τ (s)	Vrai pos.	Vrai nég.	Nouveau vrai pos.	Faux pos.	Faux nég.
0.1	633	362	5	0	0
0.01	634	362	4	0	0
0.001	634	362	4	0	0

Tableau 3.1 – Comparaison des méthodes de validation de collision continue et discrète avec différents pas de temps τ pour 1000 *chemins directs* générés aléatoirement

temps de calcul (s)	Continue			Discrète, $\tau = 0.1$s		
	min	moyenne	max	min	moyenne	max
Positifs	1.8e-04	0.095	1.1	2.6e-04	9.7e-03	0.052
Négatifs	0.18	0.57	2.3	0.011	0.033	0.057
Moyenne tous chemins	0.27			0.018		

temps de calcul (s)	Discrète, $\tau = 0.01$s			Discrète, $\tau = 0.001$s		
	min	moyenne	max	min	moyenne	max
Positifs	2.6e-04	0.095	0.50	2.6e-04	0.92	4.6
Négatifs	0.11	0.33	0.59	1.0	3.3	6.7
Moyenne tous chemins	0.18			1.8		

Tableau 3.2 – Temps de calcul pour 1000 *chemins directs* générés aléatoirement pour la validation continue de collision et la validation discrète de collision avec différents pas de temps τ, pour les chemins positifs et négatifs de chaque méthode.

positifs dans le tableau 3.1 montrent que la méthode discrétisée manque certaines collisions que la méthode continue est capable de trouver, même lorsque le pas de temps est réduit. Avec un pas de temps de 0.001 s, la méthode discrétisée a un temps de calcul plus long et ne détecte pas les collisions. Ces résultats montrent l'efficacité de la méthode continue, et son utilité dans les situations où aucune collision n'est tolérée.

3.6 Conclusion

Ce chapitre a présenté les outils pour la détection de collision pour RPC. Les corps du robot sont représentés par des objets 3D, et valider une trajectoire revient à vérifier qu'aucune paire de corps n'est en collision le long de la trajectoire. L'absence de collision est considérée comme un critère pour l'algorithme de validation continue présenté dans le Chapitre 2. La section 3.4 présente les calculs nécessaires pour une paire de corps d'un RPC. Le calcul repose sur la formulation d'un majorant de la vitesse relative entre les deux corps.

peut s'assurer qu'une collision passe inaperçue seulement si elle est suffisamment courte pour ne pas être problématique

Les résultats obtenus illustrent l'utilité de cette méthode par rapport à une méthode de validation discrètisée classique par échantillonage. La méthode continue permet de trouver des collisions qui ne sont pas trouvées par la méthode discrète,

tout en ayant des temps de calculs comparables voire inférieurs. Pour un robot classique, la méthode discrète suffit, car en choisissant un pas de temps faible, on peut s'assurer qu'une collision passe inaperçue seulement si elle est suffisamment courte pour ne pas être problématique. En effet, un court contact entre deux parties du robot ou de l'environnement est souvent inoffensif. Cependant, dans un RPC, il faut éviter le croisement des câbles. Si les câbles entrent en collision et se croisent, le robot se retrouve dans une configuration sans collision mais non atteignable, et donc impossible à obtenir en réel. Il serait possible d'ajuster pour chaque RPC le pas de temps de la méthode de validation discrète afin de s'assurer qu'aucun croisement de câble ne peut se passer. Cependant, la méthode de validation continue permet de s'affranchir de ce réglage du pas de temps, tout en ayant des temps de calculs avantageux. Avec ces avantages ainsi que la garantie de la validité continue de la trajectoire qu'elle apporte, la méthode de validation continue de collision présentée dans ce chapitre est intéressante pour la planification de mouvements de RPC, en recherche tout comme en industrie. Afin d'améliorer encore les performances, nous pourrions affiner le calcul du majorant de la vitesse relative des corps. Une piste à explorer serait l'utilisation de l'analyse par intervalle.

Maintenant que nous avons vu les outils et les calculs propres à la détection de collision, nous allons passer au deuxième critère, propre aux RPC : la validité des tensions des câbles.

Chapitre 4

Validation des tensions

Sommaire

4.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, l'espace de travail d'un RPC est limité par les collisions, notamment les interférences entre câbles. Cet espace de travail est également limité par un autre critère : les limites sur les tensions des câbles. En effet, comme les câbles ne peuvent que tirer et non pousser, leur tension doit être positive. Ils ont également une tension maximale autorisée, déterminée par les charges admissibles sur les composants mécaniques du robot, pondérées par des facteurs de sécurité. Ces contraintes sur les tensions des câbles constituent notre deuxième critère de validité pour les RPC. Ce chapitre présente les outils de validation des tensions pour RPCs. Il présente notamment les détails des calculs des éléments de validation continue pour les tensions, qui permettent de compléter l'algorithme présenté au Chapitre 2. Comme dans le reste de cette thèse, nous faisons ici aussi l'hypothèse des câbles idéaux qui n'ont ni masse ni élasticité.

La section 4.2 de ce chapitre présente les contraintes sur les tensions des câbles ainsi que des notations et définitions. La section 4.3 présente la méthode de la pseudo-inverse permettant de déterminer la validité d'une configuration par rapport aux tensions. La section 4.4 détaille une méthode différente pour la résolution du problème des tensions : la méthode de décalage d'hyperplans. La section 4.5 présente les éléments de validation des tensions pour la validation continue et les calculs associés. Enfin, les résultats d'implémentation sont présentés dans la section 4.6. La dernière section de ce chapitre conclut en discutant des performances et de l'utilité de la méthode de validation continue de tensions présentée.

Contributions de cette thèse :

- Implémentation dans HPP des méthodes de validation statique de collision : la méthode de la pseudo-inverse et la méthode de décalage d'hyperplans.
- Extension de la méthode de décalage d'hyperplans pour la validation continue et formulation des calculs des éléments de validation des tensions. Cette contribution a fait l'objet d'une publication à la conférence IEEE/RSJ IROS 2020 [140].
- Implémentation et tests de la validation continue des tensions dans le logiciel HPP.

4.2 Définition du problème et notations

4.2.1 Définitions et notations

Considérons un RPC à n degrés de liberté et possédant m câbles avec $m \geq n$, comme représenté sur la figure 4.1. Dans la suite de cette thèse, nous considérons le cas $m = 8$ et $n = 6$. L'élasticité et la masse des câbles sont négligées et tous les câbles sont supposés être sous tension. L'affaissement des câbles sous l'effet de leur propre poids, ainsi que l'élongation sous l'effet de leur tension sont négligés. Chaque câble sort de la structure fixe à un point fixe A_i et est attaché à la plateforme mobile à un point B_i, fixe dans le repère de la plateforme. La masse totale m_T — la masse de la plateforme et de sa charge utile optionnelle — est supposée constante. Le point G_p, le centre de masse de la plateforme, est l'origine du repère de la plateforme et le point de référence pour les moments cinétiques. Nous considérons le cas où la géométrie du robot ne change pas.

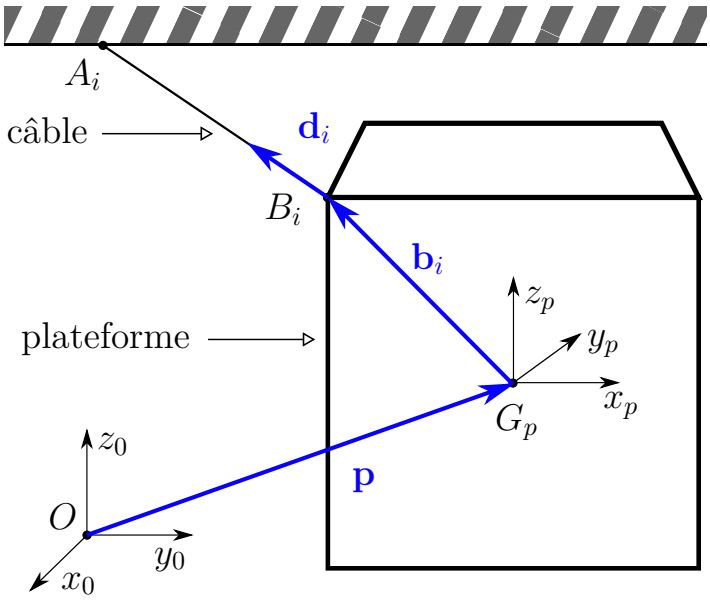


Figure 4.1 – Schéma décrivant les notations pour un RPC

4.2.2 Torseur appliqué par les câbles sur la plateforme

Nous nous plaçons sous l'hypothèse des mouvements quasi-statiques. Lorsque le robot est en équilibre mécanique, le torseur $\mathbf{f}$ des forces appliquées par les câbles sur la plateforme mobile est relié au vecteur des tensions des câbles $\boldsymbol{\tau} = \begin{bmatrix} \tau_1 & \tau_2 & \dots & \tau_m \end{bmatrix}^T$. Lorsque la masse des câbles est négligée comme dans le cas présent, cette relation s'écrit :

$$\mathbf{W}\boldsymbol{\tau} = \mathbf{f} \tag{4.1}$$

La matrice $\mathbf{W}$ appelée **matrice des torseurs** du robot est une matrice $n \times m$ dont l'expression est donnée par la formule suivante [141] :

$$\mathbf{W} = \begin{bmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_8 \\ Q\mathbf{b}_1 \times \mathbf{d}_1 & Q\mathbf{b}_2 \times \mathbf{d}_2 & \cdots & Q\mathbf{b}_8 \times \mathbf{d}_8 \end{bmatrix} \tag{4.2}$$

où :

- le vecteur $\mathbf{d}_i$ est le vecteur unitaire de direction du câble i, orienté de A_i vers B_i,
- le vecteur $\mathbf{b}_i$ est le vecteur constant de position du point d'attache du câble i sur la plateforme, exprimé dans le repère de la plateforme,
- la matrice Q est la matrice de rotation définissant l'orientation de la plateforme.

Le vecteur $\mathbf{w}_i$ désigne la i-ième colonne de $\mathbf{W}$:

$$\mathbf{w}_i = \begin{bmatrix} \mathbf{d}_i \\ Q\mathbf{b}_i \times \mathbf{d}_i \end{bmatrix} \tag{4.3}$$

Pour une configuration q donnée, comme définie dans la section 1.2.1, les vecteurs $\mathbf{b_i}$ et $\mathbf{d_i}$ pour chaque câble i et la matrice $\mathbf{Q}$ se calculent facilement à partir de q. La matrice $\mathbf{W}$ peut être calculée pour toute configuration du robot.

4.2.3 Équilibre statique de la plateforme mobile

Nous considérons des chemins sans contact entre le robot et l'environnement. Le torseur $\mathbf{f_e}$ des forces externes appliquées sur la plateforme est donc le torseur $\mathbf{f}_g$ induit par le poids de la plateforme et de la charge utile. Le moment du poids est calculé au poing G_p, centre de masse de la plateforme, et est ainsi nul. Le torseur du poids $\mathbf{f}_g$ est constant dans le repère global et pour expression :

$$\mathbf{f}_g = [0, 0, -m_T g, 0, 0, 0]^T \tag{4.4}$$

où g est l'accélération de la gravité.

Lors d'un mouvement quasi-statique, l'équilibre statique de la plateforme mobile est vérifié en tout point du chemin et s'écrit $\mathbf{f} + \mathbf{f}_g = \mathbf{0}$. En utilisant l'expression du torseur appliqué par les câbles (4.1), l'équilibre statique est donné par :

$$\mathbf{W}\tau + \mathbf{f}_g = \mathbf{0} \tag{4.5}$$

4.2.4 Limites sur les tensions et validité d'une configuration

Les tensions des câbles d'un RPC sont soumises à certaines contraintes. Les câbles ne peuvent que tirer la plateforme mobile et ne peuvent pas la pousser, ce qui impose une contrainte de non-négativité de la tension du câble : $\tau_i \geq 0$. Les limites physiques du câble et d'autres composants mécaniques du robot imposent une borne supérieure à la tension dans le câble : au-delà d'une certaine tension, le câble se déforme ou même rompt. Des marges de sécurité peuvent être prises. Pour chaque câble i, il existe une tension minimale $\tau_i^{\min}$ et une tension maximale $\tau_i^{\max}$. Les contraintes des tensions s'expriment ainsi :

Conditions de tension des câbles

$$0 \leq \tau_i^{\min} \leq \tau_i \leq \tau_i^{\max} \quad \forall i \in [1, m] \tag{4.6}$$

Ces contraintes définissent un **espace de travail faisable statique** pour le robot, c'est-à-dire l'ensemble des configurations de la plateforme mobile pour lesquelles il existe une répartition des tensions dans les câbles qui équilibre le poids de la plateforme et de son chargement. Ces contraintes permettent également de définir l'ensemble des torseurs disponibles (AWS, Available Wrench Set) dans une configuration donnée :

Ensemble des torseurs disponibles (AWS)

Le AWS est l'ensemble de tous les torseurs qui peuvent être générés à la plateforme par les câbles pour une configuration donnée, tout en satisfaisant les conditions de tension des câbles. Il se note

$$A = \left\{ \mathbf{f} \mid \mathbf{f} = \mathbf{W}\boldsymbol{\tau} \ , \ \tau_i^{\min} \leq \tau_i \leq \tau_i^{\max} \quad \forall i \in [1, m] \right\} \tag{4.7}$$

Une configuration q d'un RPC est considérée valide si et seulement si le torseur correspondant $\mathbf{f}$ est dans le AWS. Tester la validité d'une configuration revient donc à tester l'appartenance de $\mathbf{f}$ au AWS. Les méthodes présentées dans les sections suivantes 4.3 et 4.4 donnent chacune une solution à ce problème.

4.3 Résolution par la méthode de la pseudo-inverse

i le nombre de câbles m d'un RPC est strictement supérieur au nombre de degrés de liberté n, le RPC possède une redondance d'actionnement. Pour une configuration faisable, il existe alors une infinité de distributions des tensions au sein des câbles permettant d'engendrer le torseur d'efforts voulu. Une de ces distributions des tensions, notée $\hat{\boldsymbol{\tau}}$, minimise la norme euclidienne $\|\boldsymbol{\tau}\|$ du vecteur des tensions. Cette distribution des tensions particulière peut se calculer en utilisant la pseudo-inverse de Moore–Penrose de la matrice $\mathbf{W}$, calculée par exemple à partir d'une décomposition en valeurs singulières, et notée $\mathbf{W}^+$:

$$\hat{\boldsymbol{\tau}} = -\mathbf{W}^+ \mathbf{f_e} \tag{4.8}$$

La matrice $\mathbf{W}$ se calcule pour pour une configuration donnée q du robot. Avec un torseur des efforts externes donné $\mathbf{f_e} = \mathbf{f}_g$, on peut donc calculer $\hat{\boldsymbol{\tau}}$. Le test de validité de la configuration revient à tester si $\tau_{min} \leq \tau_i \leq \tau_{max}$ pour tous les $i \in [1, m]$. Si c'est le cas, on peut affirmer qu'il existe au moins une distribution des tensions – la distribution $\hat{\boldsymbol{\tau}}$ — qui équilibre le poids de la plateforme mobile et de sa charge. Sinon, on ne sait pas si la configuration q est ou n'est pas dans l'espace de travail faisable. On choisit néanmoins pour cette méthode simplifiée, de refuser les configurations

pour lesquelles $\hat{\boldsymbol{\tau}}$ ne respecte pas les contraintes des tensions. Cette méthode produit donc des *faux négatifs*, c'est-à-dire des configurations considérées non-valides alors qu'elles le sont. La méthode de la pseudo-inverse présente l'avantage d'avoir un faible coût de calcul.

Certains travaux [142, 77, 143] donnent des méthodes pour calculer différentes distributions des tensions faisables plus ou moins optimales quand elles existent.

4.4 Résolution par la méthode de décalage d'hyperplans

Cette section présente la méthode du décalage hyperplan (Hyperplane Shifting Method) selon l'article [18]. Cette méthode permet de déterminer la validité d'une configuration en calculant les frontières du AWS $A = \{\mathbf{f} \mid \mathbf{f} = \mathbf{W}\boldsymbol{\tau}\ ,\ \tau_i^{\min} \leq \tau_i \leq \tau_i^{\max} \quad \forall i \in [1, m]\}$. La méthode permet d'exprimer A en tant qu'intersection de demi-espaces dont on peut calculer les équations sous la forme $\mathbf{c}^T\mathbf{x} \leq d$. Pour une configuration donnée, on calcule ainsi les équations de ces demi-espaces, et pour un torseur $\mathbf{f}$ donné, on peut alors vérifier simplement si ce torseur appartient à A ou non. Cette méthode ne donne pas de faux négatifs comme la méthode simplifiée de la section précédente, mais est plus coûteuse en terme de calculs.

4.4.1 Propriété du AWS en tant que zonotope

Considérons une configuration donnée q du RPC. Définissons l'hypercube $[\boldsymbol{\tau}]$ des tensions admissibles :

$$[\boldsymbol{\tau}] = \left\{\boldsymbol{\tau} \mid \tau_i^{\min} \leq \tau_i \leq \tau_i^{\max} \quad \forall i \in [1, m]\right\} \tag{4.9}$$

Le AWS :

$$A = \{\mathbf{f} \mid \mathbf{f} = \mathbf{W}\boldsymbol{\tau}, \boldsymbol{\tau} \in [\boldsymbol{\tau}]\} \tag{4.10}$$

est l'image de la boîte $[\boldsymbol{\tau}]$ par l'application linéaire $\mathbf{W}$. Puisque le AWS A est l'image d'une boîte par une application linéaire, A est affinement isomorphique à un zonotope [144].

Zonotope

Un zonotope en dimension n est un polytope possédant une symétrie centrale, et dont toutes les facettes (faces de dimension $n-1$) possèdent également une symétrie centrale.

La figure 4.2 montre un hypercube de dimension 3 transformé en un zonotope de dimension 2. Dans le cas du AWS, l'hypercube des tensions est en dimension m et le zonotope est en dimension n. Un zonotope P en dimension n peut être défini en tant qu'intersection de demi-espaces.

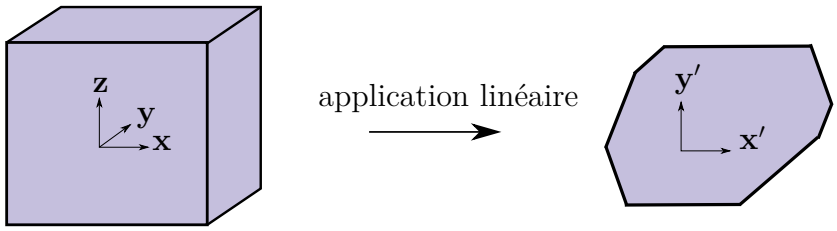


Figure 4.2 – Un hypercube 3D est transformé en un zonotope 2D par une application linéaire

Chaque demi-espace H^- peut s'exprimer sous la forme d'une inégalité : $H^- = \left\{\mathbf{x} \mid \mathbf{c}^T\mathbf{x} \leq d\right\}$ où $\mathbf{c}$ est un vecteur de dimension n et d est un scalaire. L'hyperplan correspondant $H = \left\{\mathbf{x} \mid \mathbf{c}^T\mathbf{x} = d\right\}$ est appelé un **hyperplan d'appui** de P si P est entièrement inclu dans H^- et si $P \cap H \neq \emptyset$. Pour un hyperplan d'appui H, $F = P \cap H$ est une **face** de P. Une face de dimension $n-1$ est appelée **facette**.

Hyperplan définissant une facette

Un hyperplan H définissant une facette du zonotope P est un hyperplan d'appui de P tel que $P \cap H$ est une facette de P.

Pour un hyperplan $\left\{\mathbf{x} \mid \mathbf{c}^T\mathbf{x} = d\right\}$ définissant une facette F de P, alors

$$d = \max_{\mathbf{x} \in P} \mathbf{c}^T\mathbf{x} \tag{4.11}$$

La figure 4.3 montre un zonotope dans un espace de dimension 2, où les hyperplans définissant des facettes sont donc des droites.

Un polytope peut se définir par l'intersection d'un ensemble fini d'espaces demi-espaces fermés [145]. Un zonotope P est un polytope et peut donc être défini comme l'intersection des demi-espaces bornés par les hyperplans définissant ses facettes. Considérons l'ensemble $\{F_i, i \in [1, f]\}$ des facettes de P. Pour chaque facette F_i, notons $H_i = \{\mathbf{c}^T\mathbf{x} = d\}$ l'hyperplan définissant la facette et $H_i^- = \{\mathbf{c}^T\mathbf{x} \leq d\}$ le demi-espace correspondant. On peut écrire :

$$P = \bigcup_{i=1}^{f} H_i^- \tag{4.12}$$

La méthode de décalage d'hyperplans repose sur deux propriétés, présentées et prouvées dans [18]. Nous reproduisons dans les sections suivantes les preuves de ces deux propriétés car elles sont primordiales dans la compréhension de la méthode de décalage d'hyperplans.

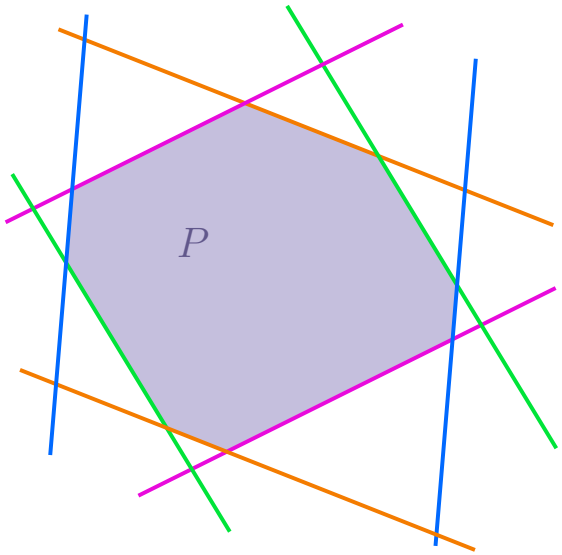


Figure 4.3 – Un zonotope P en dimension 2. Chaque paire de droites parallèles est montrée dans une couleur différente.

4.4.2 Propriété 1

Méthode du décalage d'hyperplans : propriété 1

Soit un hyperplan $H = \{\mathbf{x} \mid \mathbf{c}^T\mathbf{x} = d\}$ définissant une facette du AWS A. Alors le vecteur $\mathbf{c}$ est orthogonal à $n-1$ colonnes linéairement indépendantes $\mathbf{w}_i$ de la matrice des torseurs $\mathbf{W}$, et

$$d = \sum_{I^+} \tau_i^{\max} \mathbf{c}^T\mathbf{w}_i + \sum_{I^-} \tau_i^{\min} \mathbf{c}^T\mathbf{w}_i \tag{4.13}$$

où:

- $I^+ = \left\{i, i \in [1,m] \mid \mathbf{c}^T\mathbf{w}_i > 0\right\}$
- $I^- = \left\{i, i \in [1,m] \mid \mathbf{c}^T\mathbf{w}_i < 0\right\}$

Preuve

Considérons l'hyperplan définissant une facette $H = \{\mathbf{x} \mid \mathbf{c}^T\mathbf{x} = d\}$ de A, et sa facette $F = A \cap H$. Considérons un torseur des efforts $\mathbf{x}_F$ appartenant à la facette F. L'appartenance de $\mathbf{x}_F$ à H s'écrit :

$$\mathbf{x}_F \in H \;\Leftrightarrow\; \mathbf{c}^T\mathbf{x}_F = d = \max_{\mathbf{x}\in A} \mathbf{c}^T\mathbf{x} \tag{4.14}$$

L'appartenance de $\mathbf{x}_F$ à A peut s'écrire :

$$\mathbf{x}_F \in A \;\Leftrightarrow\; \mathbf{x}_F = \sum_{i=1}^{m} \tau_i \mathbf{w}_i \;,\; \tau_i \in [\tau_i^{\min}, \tau_i^{\max}] \tag{4.15}$$

En utilisant ces deux expressions, nous obtenons :

$$\sum_{i=1}^{m} \tau_i \mathbf{c}^T \mathbf{w}_i = \max_{\mathbf{x} \in A} \mathbf{c}^T \mathbf{x} = d \tag{4.16}$$

Décomposons la somme de (4.16) selon la partition des indices $[1, m]$ suivante :

- L'ensemble des indices pour lequels $\mathbf{c}^T \mathbf{w}_i$ est nul : $I^0 = \Big\{ i, i \in [1, m] \mid \mathbf{c}^T \mathbf{w}_i = 0 \Big\}$
- L'ensemble des indices pour lequels $\mathbf{c}^T \mathbf{w}_i$ est strictement négatif : $I^- = \Big\{ i, i \in [1, m] \mid \mathbf{c}^T \mathbf{w}_i < 0 \Big\}$
- L'ensemble des indices pour lequels $\mathbf{c}^T \mathbf{w}_i$ est strictement positif : $I^+ = \Big\{ i, i \in [1, m] \mid \mathbf{c}^T \mathbf{w}_i > 0 \Big\}$

Cela donne :

$$\begin{aligned} d = \sum_{i=1}^{m} \tau_i \mathbf{c}^T \mathbf{w}_i &= \sum_{i \in I^0} \tau_i \mathbf{c}^T \mathbf{w}_i + \sum_{i \in I^-} \tau_i \mathbf{c}^T \mathbf{w}_i + \sum_{i \in I^+} \tau_i \mathbf{c}^T \mathbf{w}_i \\ &= \sum_{i \in I^-} \tau_i \mathbf{c}^T \mathbf{w}_i + \sum_{i \in I^+} \tau_i \mathbf{c}^T \mathbf{w}_i \end{aligned} \tag{4.17}$$

Selon (4.16), les τ_i maximise la somme de l'équation (4.17). Nous avons alors nécessairement :

$$\tau_i = \tau_i^{\max} \quad \forall i \in I^+ \tag{4.18}$$

$$\tau_i = \tau_i^{\min} \quad \forall i \in I^- \tag{4.19}$$

L'équation (4.13) est ainsi prouvée. Cela signifie que les $\tau_i, i \in I^0$ ne sont pas fixés et sont libres dans $[\tau_i^{\min}, \tau_i^{\max}]$. Le torseur $\mathbf{x}_F$ peut alors s'écrire :

$$\mathbf{x}_F = \sum_{i \in I^-} \tau_i^{\min} \mathbf{w}_i + \sum_{i \in I^+} \tau_i^{\max} \mathbf{w}_i + \sum_{i \in I^0} \tau_i \mathbf{w}_i \ , \ \tau_i \in [\tau_i^{\min}, \tau_i^{\max}] \ \forall i \in I^0 \tag{4.20}$$

L'équation ci-dessus caractérise tous les torseurs $\mathbf{x}$ appartenant à $F = A \cap H$. On peut donc écrire :

$$F = \left\{ \mathbf{x} \mid \mathbf{x} = \sum_{i \in I^-} \tau_i^{\min} \mathbf{w}_i + \sum_{i \in I^+} \tau_i^{\max} \mathbf{w}_i + \sum_{i \in I^0} \tau_i \mathbf{w}_i \ , \ \tau_i \in [\tau_i^{\min}, \tau_i^{\max}] \ \forall i \in I^0 \right\} \tag{4.21}$$

Considérons le sous-espace affine engendré par F, $\text{aff}(F)$:

$$\text{aff}(F) = \left\{ \mathbf{x} \mid \mathbf{x} = \sum_{i \in I^-} \tau_i^{\min} \mathbf{w}_i + \sum_{i \in I^+} \tau_i^{\max} \mathbf{w}_i + \sum_{i \in I^0} \tau_i \mathbf{w}_i \ , \ \tau_i \in \mathbb{R} \ \forall i \in I^0 \right\} \tag{4.22}$$

La dimension de $\text{aff}(F)$ est donc égale au rang de la famille de vecteurs $\{\mathbf{w}_i \mid i \in I^0\} = \{\mathbf{w}_i \mid \mathbf{c}^T \mathbf{w}_i = 0\}$, qui est l'ensemble des vecteurs $\mathbf{w}_i$ orthogonal au vecteur $\mathbf{c}$. Or, F étant une facette de A délimitée par un hyperplan, F est de dimension $n-1$. Cela implique qu'il existe $n-1$ colonnes linéairement indépendantes $\mathbf{w}_i$ au sein de l'ensemble $\{\mathbf{w}_i \mid \mathbf{c}^T \mathbf{w}_i = 0\}$. Autrement dit, le vecteur $\mathbf{c}$ est orthogonal à $n-1$ colonnes linéairement indépendantes $\mathbf{w}_i$. La première propriété est donc prouvée.

4.4.3 Propriété 2

Méthode du décalage d'hyperplans : propriété 2

À chaque ensemble de $n-1$ colonnes linéairement indépendantes $\mathbf{w}_i$ de la matrice des torseurs $\mathbf{W}$ correspondent deux hyperplans définissant des facettes de A :

$$H_1 = \{\mathbf{x} \mid \mathbf{c}_1^T\mathbf{x} = d_1\} \qquad H_2 = \{\mathbf{x} \mid \mathbf{c}_2^T\mathbf{x} = d_2\} \tag{4.23}$$

Le vecteur $\mathbf{c}_1 = -\mathbf{c}_2$ est non nul et est orthogonal aux $n-1$ colonnes $\mathbf{w}_i$. Les scalaires d_1 et d_2 sont donnés par :

$$d_1 = \sum_{I_1^+} \tau_i^{\max}\mathbf{c}_1^T\mathbf{w}_i + \sum_{I_1^-} \tau_i^{\min}\mathbf{c}_1^T\mathbf{w}_i \qquad d_2 = \sum_{I_2^+} \tau_i^{\max}\mathbf{c}_2^T\mathbf{w}_i + \sum_{I_2^-} \tau_i^{\min}\mathbf{c}_2^T\mathbf{w}_i \tag{4.24}$$

où :

- $I_1^+ = \left\{i, i \in [1,m] \mid \mathbf{c}_1^T\mathbf{w}_i > 0\right\}$
- $I_1^- = \left\{i, i \in [1,m] \mid \mathbf{c}_1^T\mathbf{w}_i < 0\right\}$
- $I_2^+ = \left\{i, i \in [1,m] \mid \mathbf{c}_2^T\mathbf{w}_i > 0\right\}$
- $I_2^- = \left\{i, i \in [1,m] \mid \mathbf{c}_2^T\mathbf{w}_i < 0\right\}$

Remarques

Il existe une infinité de vecteurs orthogonal aux $n-1$ colonnes $\mathbf{w}_i$. En effet, les $n-1$ colonnes linéairement indépendante sont orthogonales à un sous-espace vectoriel de dimension 1. Le vecteur $\mathbf{c}$ est à choisir dans cet ensemble.

Preuve

Cette preuve s'appuie sur les mêmes arguments que la preuve de la première propriété. Considérons un ensemble de $n-1$ colonnes linéairement indépendantes $\mathbf{w}_i$ de $\mathbf{W}$. Sans perte de généralité, supposons que ce sont les vecteurs colonnes $\mathbf{w}_1$, $\mathbf{w}_2, \ldots, \mathbf{w}_{n-1}$. Soit $\mathbf{c}_1$ un vecteur non-nul orthogonal à ces vecteurs — un tel vecteur $\mathbf{c}_1$ existe puisque l'espace des torseurs est de dimension n. Soit d_1 comme défini par la propriété dans l'équation (4.24), et considérons l'hyperplan $H_1 = \{\mathbf{x} \mid \mathbf{c}_1^T\mathbf{x} = d_1\}$. Alors :

$$\mathbf{c}_1^T\mathbf{x} = d_1 = \sum_{I_1^+} \tau_i^{\max}\mathbf{c}_1^T\mathbf{w}_i + \sum_{I_1^-} \tau_i^{\min}\mathbf{c}_1^T\mathbf{w}_i \tag{4.25}$$

En considérant la partition $I_1^0 = \{i, i \in [1,m] \mid \mathbf{c}_1^T\mathbf{w}_i = 0\}$, I_1^-, I_1^+ de $[1,m]$ comme définis dans la propriété 2, nous pouvons affirmer que $d_1 = \max_{\mathbf{f}\in A} \mathbf{c}_1^T\mathbf{f}$. Cela signifie que H est un hyperplan d'appui de P. Nous notons $F_1 = H_1 \cap A$ la face correspondante, donnée par l'équation suivante :

$$F_1 = \left\{ \mathbf{x} \mid \mathbf{x} = \sum_{i \in I_1^-} \tau_i^{\min} \mathbf{w}_i + \sum_{i \in I_1^+} \tau_i^{\max} \mathbf{w}_i + \sum_{i \in I_1^0} \tau_i \mathbf{w}_i, \tau_i \in [\tau_i^{\min}, \tau_i^{\max}] \forall i \in I_1^0 \right\} \tag{4.26}$$

Comme dans la preuve précédente, considérons la dimension du sous-espace affine engendré par F_1 :

$$\dim(\text{aff}(F_1)) = \text{rank}\left(\{\mathbf{w}_i, i \in I_1^0\}\right) = \text{rank}\left(\{\mathbf{w}_i, \mathbf{c}_1^T \mathbf{w}_i = 0\}\right) \tag{4.27}$$

Puisque $\mathbf{c}_1$ est orthogonal aux $\mathbf{w}_1$, $\mathbf{w}_2$, ..., $\mathbf{w}_{n-1}$, ces derniers appartiennent à $\{\mathbf{w}_i \mid \mathbf{c}_1^T \mathbf{w}_i = 0\}$. Donc $\dim(\text{aff}(F_1)) = n - 1$, ce qui signifie que F_1 est une facette de A.

Considérons maintenant le vecteur $\mathbf{c}_2 = -\mathbf{c}_1$ et l'hyperplan $H_2 = \{\mathbf{x} \mid \mathbf{c}_2 \mathbf{x} = d_2\}$ où d_2 est définie par (4.24). H_2 est donc parallèle à H_1. En utilisant le même raisonnement que pour H_1, nous montrons que H_2 est également un hyperplan définissant une facette de A. La propriété 2 est donc prouvée.

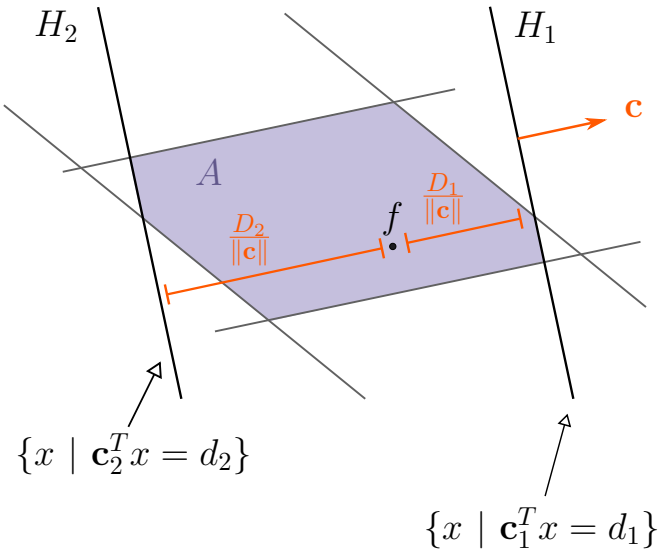


Figure 4.4 – Zonotope 2D avec trois paires d'hyperplans parallèles définissant des facettes du zonotope

4.4.4 Test de validité d'une configuration

Le test de validité d'une configuration vis-à-vis des tensions s'effectue grâce aux deux propriétés présentées ci-dessus. Chaque combinaison de $n-1$ câbles parmi m correspond à deux hyperplans parallèles et il faut, pour chacune de ces combinaisons, vérifier que le torseur des efforts externes se trouve bien entre les deux hyperplans. Si pour une combinaison, le torseur se situe du mauvais côté d'un des hyperplans, alors le torseur est en dehors du AWS A et la configuration n'est pas valide. Soit I_0 l'ensemble des indices des colonnes correspondant à la combinaison de $n-1$ câbles considérée et $\mathbf{W}_0$ la sous-matrice correspondante de $\mathbf{W}$.

L'algorithme 3 ci-dessous présente la validation d'une configuration vis-à-vis des tensions en utilisant la méthode de décalage d'hyperplans. Cette fonction prend en entrée, pour un RPC donné :

- une configuration q,
- le vecteur des tensions minimales des câbles $\boldsymbol{\tau}^{\min} = \begin{bmatrix} \tau_1^{\min} & \dots & \tau_m^{\min} \end{bmatrix}^T$, et
- le vecteur des tensions maximales des câbles $\boldsymbol{\tau}^{\max} = \begin{bmatrix} \tau_1^{\max} & \dots & \tau_m^{\max} \end{bmatrix}^T$.

Algorithme 3 Validation des tensions d'une configuration grâce à la méthode de décalage d'hyperplans

function HyperplaneShiftingMethod($q, \boldsymbol{\tau}^{\min}, \boldsymbol{\tau}^{\max}$)
  $\mathbf{W} \leftarrow$ wrench matrix(q)
  $nbcomb \leftarrow$ number of combinations of $\binom{m}{n-1}$
  $I \leftarrow$ list of combinations of $\binom{m}{n-1}$ indices
  $\mathbf{C} \leftarrow$ empty $2nbcomb \times n$ matrix
  $\mathbf{d} \leftarrow$ empty $2nbcomb$ vector
  for $i = 0 \dots nbcomb - 1$ **do**
    $I_0 \leftarrow I(i)$
    $\mathbf{V} \leftarrow$ matrix formed by the $n-1$ columns $(\mathbf{w}_i)_{i \in I_0}$ of $\mathbf{W}$
    $\mathbf{c} \leftarrow$ basis of the nullspace of $\mathbf{V}$
    if nullspace of $\mathbf{V}$ is of dimension 1 **then**
      Column $2i$ of $C \leftarrow \mathbf{c}$
      Column $2i+1$ of $C \leftarrow -\mathbf{c}$
      $\mathbf{d}(2i) \leftarrow d_1$ computed as in Eq. (4.24)
      $\mathbf{d}(2i+1) \leftarrow d_2$ computed as in Eq. (4.24)
    end if
  end for
  $\mathbf{f} \leftarrow$ wrench induced by the weight of the platform at configuration q
  $\mathbf{d}_q \leftarrow \mathbf{Cf}$
  for each $i = 0 \dots 2nbcomb$ **do**
    if $\mathbf{d}_q(i) > \mathbf{d}(i)$ **then**
      return False
    end if
  end for
  return True
end function

La méthode itère sur les combinaison de $n-1$ vecteurs colonnes de $\mathbf{W}$. Pour chaque combinaison, nous notons $\mathbf{V}$ la matrice composée par les colonnes $\mathbf{w}_i$. Calculer un vecteur $\mathbf{c}$ revient à calculer une base du noyau de la matrice $\mathbf{V}$. Le vecteur $\mathbf{c}$ peut donc être choisi au moyen de n'importe quelle programme qui détermine le noyau d'une matrice. Si le noyau est vide, alors les vecteurs $\mathbf{w}_i$ ne sont pas linéairement indépendants et ne correspondent pas à des hyperplanes définissant des facettes de A. La fonction remplit la matrice $\mathbf{C}$ et le vecteur $\mathbf{d}$ avec les valeurs $\mathbf{c}$, $-\mathbf{c}$ et d_1, d_2, respectivement, au fur et à mesure de l'analyse des différentes combinaisons. Ces matrices permettent de définir le AWS.

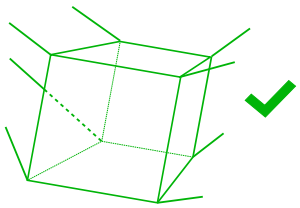
Une fois $\mathbf{C}$ et $\mathbf{d}$ calculés, il faut donc vérifier que le torseur des efforts requis se trouve dans le AWS. Le torseur $\mathbf{f}$ est dans le AWS A si et seulement si pour toutes les paires d'hyperplans parallèles définissant des facettes $H_1 = \{\mathbf{f} \mid \mathbf{c}^T\mathbf{f} = d_1\}$ et $H_2 = \{\mathbf{f} \mid -\mathbf{c}^T\mathbf{f} = d_2\}$, nous avons :

$$\mathbf{c}^T\mathbf{f} - d_1 \leq 0 \quad \text{and} \quad -\mathbf{c}^T\mathbf{f} - d_2 \leq 0 \tag{4.28}$$

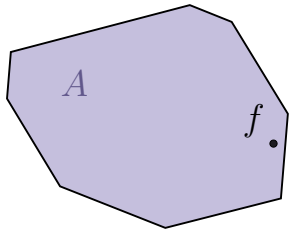
Cela se traduit également par :

$$\mathbf{Cf} \leq \mathbf{d} \tag{4.29}$$

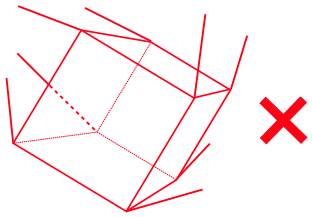
où $\leq$ représente l'opérateur « inférieur ou égal » appliqué entre deux vecteurs élément par élément.



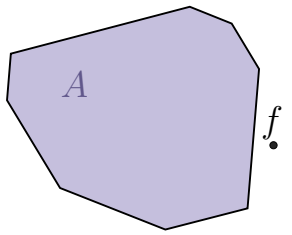
(a) Configuration valide de la plateforme mobile



(b) Le zonotope du AWS contient le torseur requis



(c) Configuration non-valide de la plateforme mobile



(d) Le zonotope du AWS ne contient pas le torseur requis

Figure 4.5 – Exemple d'une configuration valide et d'une configuration non-valide vis-à-vis des tensions

La figure 4.5 illustre un cas valide et un cas non-valide de configurations. Le torseur des efforts requis est constant. Le AWS A change avec les mouvements de la plateforme mobile. Lorsque le torseur $\mathbf{f}$ n'est pas dans le zonotope, la configuration est non-valide.

4.5 Validation continue

Cette section présente une extension de la méthode de décalage d'hyperplans afin de définir des éléments de validation des tensions pour l'algorithme de validation ammetsenté dans le chapitre 2. Dans le reste de cette partie, nous considérons un RPC à 6 degrés de liberté ($n = 6$) et à 8 câbles ($m = 8$).

4.5.1 Méthode de calcul d'un intervalle valide pour un hyperplan

Pour chaque hyperplan, l'objectif est de calculer, pour une configuration donnée du robot le long d'un chemin, un intervalle autour de cette configuration dans lequel les conditions de (4.28) sont respectées. Nous utilisons la même méthode que pour les calculs de validation continue de collisions présentées dans le chapitre 3. Pour une configuration donnée, nous définissons une « distance à l'obstacle », et en trouvant un minorant de cette distance (ou sa valeur exacte) ainsi qu'un majorant de la valeur absolue de sa dérivée, nous obtenons la demi-longueur d'un intervalle valide centré autour de la configuration donnée.

Pour chaque hyperplan définissant une facette du AWS, nous définissons une distance entre le torseur requis et l'hyperplan dans la section 4.5.2, et nous calculons un majorant de la valeur absolue de sa dérivée dans la section 4.5.3. Pour chaque hyperplan H_i, $i \in [1, f]$, notons D_i la distance entre le torseur requis et H_i et $V_i^{\max}$ un majorant de la valeur absolue de la dérivée de D_i. En répétant ces calculs pour chaque paire d'hyperplans définissant les facettes du AWS on obtient la demi-longueur d'un intervalle valide Δt vis-à-vis des tensions:

$$\Delta t = \min_{i \in [1,f]} \frac{D_i}{V_i^{\max}} \tag{4.30}$$

Pour que la méthode de décalage d'hyperplans, et donc la présente méthode, fonctionne, la matrice $\mathbf{W}$ doit avoir au moins n colonnes linéairement indépendantes. Pour CoGiRo, avec $n = 6$ et $m = 8$, cela signifie que la matrice $\mathbf{W}$ doit être de rang 6 pour que la méthode fonctionne. Si $\mathbf{W}$ a un rang inférieur à 6, le robot est dans une singularité et ne peut pas contrôler tous ses DOF.

4.5.2 Distance entre le torseur requis et un hyperplan

Pour une configuration donnée du RPC et une paire d'hyperplans parallèles définissant des facettes H_1 et H_2 définis dans (4.23), $H_1 = \{\mathbf{f} \mid \mathbf{c}^T\mathbf{f} = d_1\}$ et $H_2 = \{\mathbf{f} \mid -\mathbf{c}^T\mathbf{f} = d_2\}$, nous définissons les valeurs D_1 et D_2 :

$$D_1 = -\mathbf{c}^T\mathbf{f} + d_1 \qquad D_2 = \mathbf{c}^T\mathbf{f} + d_2 \tag{4.31}$$

La configuration est valide si pour toutes les paires d'hyperplans H_1 et H_2, D_1 et D_2 sont positifs. Il convient de noter que D_1 et D_2 dépendent du vecteur $\mathbf{c}$, et ne sont pas les « distances euclidiennes » en dimension 6 entre $\mathbf{f}$ et les hyperplans H_1 et H_2, qui sont respectivement $\frac{D_1}{\|\mathbf{c}\|}$ et $\frac{D_2}{\|\mathbf{c}\|}$.

Sans perte de généralité, supposons que l'ensemble d'indices I_0 est égal à $\{1,2,3,4,5\}$. Considérons $\mathbf{V} = \mathbf{W}_0^T$, de sorte que les lignes de $\mathbf{V}$ sont les transpositions des vecteurs colonnes $(\mathbf{w}_i)_{i\in I^0}$. Comme l'ensemble $(\mathbf{w}_i)_{i\in I^0}$ est linéairement indépendant, $\mathbf{V}$ est une matrice de taille 5×6 de plein rang égal à 5. Soit $\mathbf{v}_i$ les colonnes de $\mathbf{V}$. Le noyau de $\mathbf{V}$ est de dimension 1 et (4.32) donne une expression à base de déterminants d'un vecteur $\mathbf{c}$ couvrant ce noyau, c'est-à-dire $\ker(\mathbf{V}) = \text{span}(\mathbf{c})$ [146]. Un tel vecteur $\mathbf{c}$ satisfait $\mathbf{Vc} = \mathbf{0}$, ce qui signifie que $\mathbf{c}$ est orthogonal à l'ensemble $(\mathbf{w}_i)_{i\in I^0}$.

$$\mathbf{c} = \begin{bmatrix} \det([\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4\mathbf{v}_5\mathbf{v}_6]) \\ -\det([\mathbf{v}_1\mathbf{v}_3\mathbf{v}_4\mathbf{v}_5\mathbf{v}_6]) \\ \det([\mathbf{v}_1\mathbf{v}_2\mathbf{v}_4\mathbf{v}_5\mathbf{v}_6]) \\ -\det([\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3\mathbf{v}_5\mathbf{v}_6]) \\ \det([\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4\mathbf{v}_6]) \\ -\det([\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4\mathbf{v}_5]) \end{bmatrix} \tag{4.32}$$

Le i-ième élément de $\mathbf{c}$ est égal à $(-1)^{i+1}\det_i$ où $\det_i$ est le déterminant de la matrice carrée 5×5 obtenue à partir de la matrice $\mathbf{V}$ en supprimant sa i-ième colonne. Comme la matrice $\mathbf{V}$ est de plein rang, $\mathbf{c}$ défini par (4.32) est un vecteur non nul. En utilisant cette expression de $\mathbf{c}$, nous pouvons facilement calculer D_1 et D_2 à partir de (4.24) et (4.31). Une configuration est valide si $D_1 > 0$ et $D_2 > 0$ pour chaque paire d'hyperplans parallèles définissant des facettes du AWS. Il convient de noter que $\mathbf{c}$ n'est délibérément pas normalisé afin de faciliter le calcul d'un majorant sur la dérivée de $\|\mathbf{c}\|$ dans la section suivante.

4.5.3 Majorant de la vitesse relative entre le torseur requis et un hyperplan

Pour une configuration donnée et une paire d'hyperplans parallèles définissant des facettes H_1 et H_2, les dérivées temporelles des valeurs D_1 et D_2 sont obtenues en différenciant (4.31) :

$$\dot{D}_1 = -\dot{\mathbf{c}}^T\mathbf{f} - \mathbf{c}^T\dot{\mathbf{f}} + \dot{d}_1 \qquad \dot{D}_2 = \dot{\mathbf{c}}^T\mathbf{f} + \mathbf{c}^T\dot{\mathbf{f}} + \dot{d}_2 \tag{4.33}$$

Comme indiqué au paragraphe 4.2.3, nous considérons que le torseur d'efforts requis de la plateforme $\mathbf{f}$ est constant et connu, ce qui donne : $\dot{\mathbf{f}} = \mathbf{0}$. En utilisant l'inégalité triangulaire, nous obtenons :

$$|\dot{D}_1| \le \|\dot{\mathbf{c}}\|\|\mathbf{f}\| + |\dot{d}_1| \qquad |\dot{D}_2| \le \|\dot{\mathbf{c}}\|\|\mathbf{f}\| + |\dot{d}_2| \tag{4.34}$$

En différenciant (4.24) :

$$\dot{d}_1 = \sum_{I^+} \tau_i^{\max}(\dot{\mathbf{c}}^T\mathbf{w}_i + \mathbf{c}^T\dot{\mathbf{w}}_i) + \sum_{I^-} \tau_i^{\min}(\dot{\mathbf{c}}^T\mathbf{w}_i + \mathbf{c}^T\dot{\mathbf{w}}_i) \tag{4.35}$$

$$\dot{d}_2 = -\sum_{I^-} \tau_i^{\max}(\dot{\mathbf{c}}^T\mathbf{w}_i + \mathbf{c}^T\dot{\mathbf{w}}_i) - \sum_{I^+} \tau_i^{\min}(\dot{\mathbf{c}}^T\mathbf{w}_i + \mathbf{c}^T\dot{\mathbf{w}}_i) \tag{4.36}$$

Puisque nous considérons une combinaison fixe de 5 vecteurs $(\mathbf{w}_i)_{i\in I_0}$ pour lesquels $\mathbf{c}^T\mathbf{w}_i = 0$, nous savons que $I^+ \cup I^-$ est constant. Pour $i \in I^+ \cup I^-$, les valeurs absolues des termes correspondants de $\dot{d}_1$ et $\dot{d}_2$ sont soit $\tau_i^{\max}|\dot{\mathbf{c}}^T\mathbf{w}_i + \mathbf{c}^T\dot{\mathbf{w}}_i|$ ou $\tau_i^{\min}|\dot{\mathbf{c}}^T\mathbf{w}_i + \mathbf{c}^T\dot{\mathbf{w}}_i|$. De plus, $\tau_i^{\min} \leq \tau_i^{\max}$ donne :

$$|\dot{d}_{\delta\in\{1,2\}}| \leq \sum_{i\in I^+\cup I^-} \tau_i^{\max}|\dot{\mathbf{c}}^T\mathbf{w}_i + \mathbf{c}^T\dot{\mathbf{w}}_i| \tag{4.37}$$

Nous considérons les majorations suivantes :

- $w_i^{\max}$ est un majorant de $\|\mathbf{w}_i\|$.
- $\dot{w}_i^{\max}$ est un majorant de $\|\dot{\mathbf{w}}_i\|$.
- $c^{\max}$ est un majorant de $\|\mathbf{c}\|$.
- $\dot{c}^{\max}$ est un majorant de $\|\dot{\mathbf{c}}\|$.

Un majorant $V^{\max}$ des normes de $\dot{D}_1$ et $\dot{D}_2$ est obtenu en utilisant ces majorants et le fait que $\|\mathbf{f}\| = \| - \mathbf{f}_g\| = m_T g$:

$$V^{\max} = \dot{c}^{\max} m_T g + \sum_{i\in I^+\cup I^-} \tau_i^{\max}(\dot{c}^{\max} w_i^{\max} + c^{\max}\dot{w}_i^{\max}) \tag{4.38}$$

Il faut maintenant calculer des valeurs pour les majorants définis ci-dessus.

4.5.3.a Majorant $w_i^{\max}$ de $\|\mathbf{w}_i\|$

Un vecteur $\mathbf{w}_i$ est un vecteur colonne contenant les forces unitaires et couples appliqués par le câble i sur la plateforme et a l'expression suivante :

$$\mathbf{w}_i = \begin{bmatrix} \mathbf{d}_i \\ Q\mathbf{b}_i \times \mathbf{d}_i \end{bmatrix} \tag{4.39}$$

Par définition, $\mathbf{d}_i$ est un vecteur unitaire et Q est une matrice de rotation, et a donc une norme unitaire également. On peut écrire $\|\mathbf{w}_i\|^2 = \|\mathbf{d}_i\|^2 + \|Q\mathbf{b}_i \times \mathbf{d}_i\|^2$, ce qui donne une limite supérieure $w_i^{\max}$ sur la norme de $\mathbf{w}_i$:

$$\|\mathbf{w}_i\| \leq \sqrt{1 + b_i^2} = w_i^{\max} \tag{4.40}$$

où b_i est la norme constante du vecteur $\mathbf{b}_i$, constant pour un RPC donné.

4.5.3.b Majorant $\dot{w}_i^{\max}$ de $\|\dot{\mathbf{w}}_i\|$

En différenciant l'expression de $\mathbf{w}_i$ dans (4.39), on obtient :

$$\dot{\mathbf{w}}_i = \begin{bmatrix} \dot{\mathbf{d}}_i \\ [\boldsymbol{\omega}_p]_\times Q\mathbf{b}_i \times \mathbf{d}_i + Q\mathbf{b}_i \times \dot{\mathbf{d}}_i \end{bmatrix} \tag{4.41}$$

où $[\boldsymbol{\omega}_p]_\times$ est la matrice du produit vectoriel associée au vecteur vitesse angulaire de la plateforme $\boldsymbol{\omega}_p$. Nous devons trouver des majorants des normes des deux vecteurs composant $\dot{\mathbf{w}}_i$. Pour $\dot{\mathbf{d}}_i$, nous avons par définition :

$$\mathbf{d}_i = \frac{\overrightarrow{B_iA_i}}{L_i} \tag{4.42}$$

avec $\|\overrightarrow{B_iA_i}\| = L_i$. Nous utilisons la formule pour la dérivée d'un vecteur unitaire démontrée dans l'annexe 6.5 pour exprimer la dérivée du vecteur $\mathbf{d}_i$:

$$\dot{\mathbf{d}}_i = \frac{1}{L_i}(I - \mathbf{d}_i\mathbf{d}_i^T)\frac{d\overrightarrow{B_iA_i}}{dt} \tag{4.43}$$

Puisque $\overrightarrow{B_iA_i} = \mathbf{a}_i - Q\mathbf{b}_i - \mathbf{p}$, nous avons $\frac{d\overrightarrow{B_iA_i}}{dt} = -\dot{Q}\mathbf{b}_i - \mathbf{v}_p$ parce que $\mathbf{a}_i$ et $\mathbf{b}_i$ sont des vecteurs constants. Cela donne $\|\frac{d\overrightarrow{B_iA_i}}{dt}\| \leq \omega_p b_i + v_p$. Comme dans le chapitre précédent, nous considérons une longueur minimale possible constante et non négative pour chaque câble du RPC : $L_i^{\min} \leq L_i,\ \forall i \in [1, 8]$.

Par définition du vecteur $\mathbf{d}_i$, sa norme est égale à 1 et $\|\mathbf{d}_i\mathbf{d}_i^T\| \leq \|\mathbf{d}_i\|^2$, donc $\|\mathbf{d}_i\mathbf{d}_i^T\| \leq 1$. L'expression suivante pour le majorant $\dot{d}_i^{\max}$ de $\|\dot{\mathbf{d}}_i\|$ est obtenu :

$$\|\dot{\mathbf{d}}_i\| \leq \frac{2}{L_i^{\min}}(\omega_p b_i + v_p) = \dot{d}_i^{\max} \tag{4.44}$$

Pour la norme de la dérivée du moment dans le vecteur (4.41), on peut écrire l'inégalité suivante :

$$\|[\boldsymbol{\omega}_p]_\times Q\mathbf{b}_i \times \mathbf{d}_i + Q\mathbf{b}_i \times \dot{\mathbf{d}}_i\| \leq \|[\boldsymbol{\omega}_p]_\times\|\|Q\|\|\mathbf{b}_i\|\|\mathbf{d}_i\| + \|Q\|\|\mathbf{b}_i\|\|\dot{\mathbf{d}}_i\| \tag{4.45}$$

Par définition, $\|\mathbf{d}_i\| = 1$ et $\|Q\| = 1$, et la norme de $\boldsymbol{\omega}_p$ est constante le long du chemin par hypothèse. L'inégalité (4.45) peut être réécrite comme :

$$\|[\boldsymbol{\omega}_p]_\times Q\mathbf{b}_i \times \mathbf{d}_i + Q\mathbf{b}_i \times \dot{\mathbf{d}}_i\| \leq \omega_p b_i + b_i\dot{d}_i^{\max} \tag{4.46}$$

La norme de $\dot{\mathbf{w}}_i$ peut être exprimée en utilisant (4.41) : $\|\dot{\mathbf{w}}_i\| = \sqrt{\|\dot{\mathbf{d}}_i\|^2 + \|[\boldsymbol{\omega}_p]_\times Q\mathbf{b}_i \times \mathbf{d}_i + Q\mathbf{b}_i \times \dot{\mathbf{d}}_i\|^2}$, ce qui donne l'inégalité suivante définissant le majorant $\dot{w}_i^{\max}$ de la norme de $\mathbf{w}_i$:

$$\|\dot{\mathbf{w}}_i\| \leq \sqrt{\dot{d}_i^{\max} + b_i^2(\omega_p + \dot{d}_i^{\max})^2} = \dot{w}_i^{\max} \tag{4.47}$$

4.5.3.c Majorant $c^{\max}$ de $\|\mathbf{c}\|$

Le vecteur $\mathbf{c}$ est défini dans (4.32). Considérons la matrice carrée $\mathbf{V}_i$ dont les colonnes sont les vecteurs $\mathbf{v}_j$ avec $j \neq i$. Par exemple, pour $\mathbf{V}_1$, nous avons :

$$\begin{aligned} \det([\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4\mathbf{v}_5\mathbf{v}_6]) &= \det(V_1) && (4.48)\\ &= \det(V_1^T) = \det([\mathbf{u}_1^1\mathbf{u}_2^1\mathbf{u}_3^1\mathbf{u}_4^1\mathbf{u}_5^1]) && (4.49) \end{aligned}$$

où les $\mathbf{u}_k^i$, $1 \leq k \leq 6$ sont les colonnes de $\mathbf{V}_i^T$ (c'est-à-dire les lignes de $\mathbf{V}_i$). Le vecteur $\mathbf{u}_k^i$ est égal au vecteur $\mathbf{w}_k$ auquel on a enlevé le i-ième coefficient. Cela donne :

$$\mathbf{c} = \begin{bmatrix} \det([\mathbf{u}_1^1\mathbf{u}_2^1\mathbf{u}_3^1\mathbf{u}_4^1\mathbf{u}_5^1]) \\ -\ \det([\mathbf{u}_1^2\mathbf{u}_2^2\mathbf{u}_3^2\mathbf{u}_4^2\mathbf{u}_5^2]) \\ \det([\mathbf{u}_1^3\mathbf{u}_2^3\mathbf{u}_3^3\mathbf{u}_4^3\mathbf{u}_5^3]) \\ -\ \det([\mathbf{u}_1^4\mathbf{u}_2^4\mathbf{u}_3^4\mathbf{u}_4^4\mathbf{u}_5^4]) \\ \det([\mathbf{u}_1^5\mathbf{u}_2^5\mathbf{u}_3^5\mathbf{u}_4^5\mathbf{u}_5^5]) \\ -\ \det([\mathbf{u}_1^6\mathbf{u}_2^6\mathbf{u}_3^6\mathbf{u}_4^6\mathbf{u}_5^6]) \end{bmatrix} \tag{4.50}$$

Sachant que $\mathbf{u}_k^i$ est un vecteur de dimension 5 et que $\mathbf{w}_k$ est un vecteur de dimension 6 avec les mêmes éléments que $\mathbf{u}_k$ et un élément de plus, on peut écrire $\|\mathbf{u}_k^i\| \leq \|\mathbf{w}_k\|$ et $\|\dot{\mathbf{u}}_k^i\| \leq \|\dot{\mathbf{w}}_k{}^i\|$. En utilisant l'inégalité de Hadamard, qui stipule que la norme d'un déterminant d'une matrice n'est pas plus grande que le produit des normes de ses vecteurs colonnes, nous obtenons :

$$\|\mathbf{c}\| \leq \sqrt{\sum_{i=1}^{i=6} \left(\prod_{k=1}^{k=5} w_k^{\max} \right)^2} \tag{4.51}$$

On obtient donc l'expression suivante pour le majorant $c^{\max}$ de la norme de $\dot{\mathbf{c}}$.

$$\|\mathbf{c}\| \leq c^{\max} = \sqrt{6} \prod_{k=1}^{k=5} w_k^{\max} \tag{4.52}$$

4.5.3.d Majorant $\dot{c}^{\max}$ de $\|\dot{\mathbf{c}}\|$

L'expression de $\mathbf{c}$ est donnée par (4.50). La formule de la dérivée du déterminant d'une matrice et l'expansion de Laplace fournissent une expression du i-ième coefficient de $\dot{\mathbf{c}}$:

$$\dot{c}_i = (-1)^{i+1} \sum_{j=1}^{j=5} \det_u^{i,j} \tag{4.53}$$

$$\det_u^{i,j} = \det(\mathbf{u}_1^i, ..., \dot{\mathbf{u}}_j^i, ..., \mathbf{u}_5^i) \tag{4.54}$$

En reprenant l'inégalité de Hadamard, $\|\det_u^{i,j}\|$ peut être borné :

$$|\det_u^{i,j}| \leq \|\dot{\mathbf{u}}_j^i\| \prod_{k-1,\neq j}^{k=n-1} \|\mathbf{u}_k^i\| \tag{4.55}$$

En utilisant à nouveau les inégalités $\|\mathbf{u}_k^i\| \leq \|\mathbf{w}_k\|$ et $\|\dot{\mathbf{u}}_k^i\| \leq \|\dot{\mathbf{w}}_k{}^i\|$, on obtient la majoration de $\|\dot{\mathbf{c}}\|$ suivante :

$$\|\dot{\mathbf{c}}\| \leq \sqrt{\sum_{i=1}^{i=6} \left(\sum_{j=1}^{j=5} \left(\|\dot{\mathbf{w}}_j\| \prod_{k=1,k\neq j}^{k=5} \|\mathbf{w}_k\| \right) \right)^2} \tag{4.56}$$

Ainsi, $\|\dot{\mathbf{c}}\| \leq \dot{c}^{\max}$ peut s'écrire :

$$\dot{c}^{\max} = \sqrt{6} \sum_{j=1}^{j=5} \left(\dot{w}_j^{\max} \prod_{k=1,k\neq j}^{k=5} w_k^{\max} \right) \tag{4.57}$$

En utilisant (4.40), (4.47), (4.52) et (4.57), on peut calculer $V^{\max}$ défini dans (4.38).

4.6 Résultats d'implémentation

L'algorithme de validation continue présenté dans la section 2.2.3 et les calculs présentés dans la section 4.5 sont implémentés dans le logiciel HPP. Pour trouver une trajectoire valide entre deux configurations données du robot, des méthodes de planification de trajectoire telles que RRT [8] sont utilisées. Ces méthodes appellent la méthode de validation continue proposée dans cette thèse pour valider des trajectoires plus petites, et les concatènent pour obtenir une trajectoire globale valide.

La longueur T d'un chemin est exprimée dans une unité correspondant à la variation de la position et de l'orientation de la plateforme mobile. L'espace de travail en équilibre statique du RPC avec une orientation variable n'est pas convexe : pour un *chemin direct* entre deux configurations valides, un intervalle non valide peut exister le long du trajet. La méthode proposée est capable de détecter de tels intervalles non valides le long d'un chemin, quelle que soit la taille de l'intervalle.

Des *chemins directs* aléatoires sont générés en tirant des configurations aléatoires valides de la plateforme mobile dans l'espace de travail. Ces *chemins directs* sont vérifiés à l'aide de l'algorithme de validation continue et des calculs de validation continue des tensions présentés dans ce chapitre. Le tableau 4.1 indique les temps de calcul pour la validation continue de $N = 1000$ chemins aléatoires ayant tous une longueur égale à 1. Si la configuration au milieu d'un chemin n'est pas valide, la méthode continue se termine très rapidement, comme le montre le temps de calcul minimum. Les chemins les plus longs à valider sont ceux le long desquels la plateforme mobile s'approche (sans traverser) de la frontière de l'espace de travail valide. Dans ces cas, le torseur des efforts requis $\mathbf{f}$ se rapproche de plus en plus des limites du AWS, et donc l'algorithme valide des intervalles de plus en plus petits, ce qui se traduit par un temps de calcul plus long.

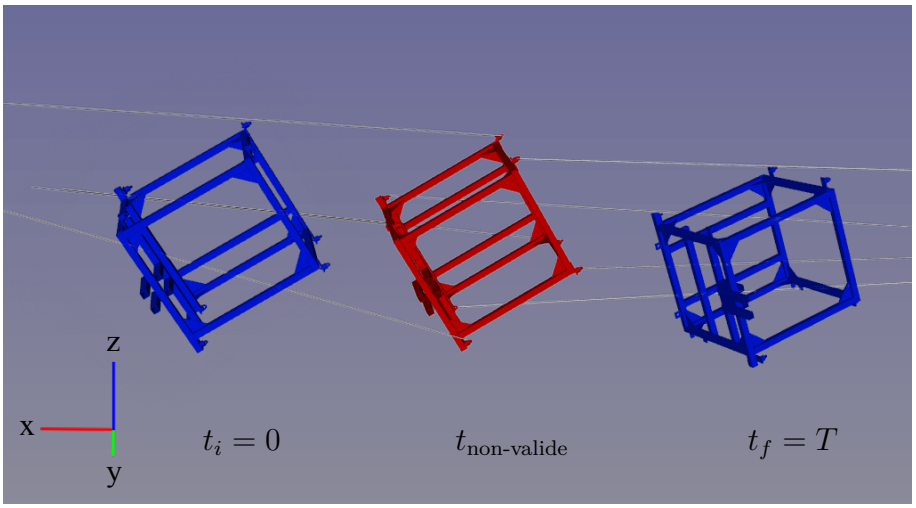


Figure 4.6 – Visualisation d'un chemin du RPC CoGiRo. La première et la dernière configuration du chemin sont valides (en bleu). La validation continue des tensions a permis de trouver une configuration non valide (rouge) au paramètre $t_{\text{non-valide}}$, donc le chemin n'est pas valide. Les câbles (gris) ne sont affichés que pour la configuration rouge.

	nombre de chemins	temps de calcul (s)		
		min	moyenne	max
Chemins valides	994	1.21	4.51	29.52
Chemins non-valides	6	4.31e-04	2.45	14.68
Tous	1000	4.49		

Tableau 4.1 – Résultats pour la validation continue des tensions de 1000 *chemins directs* aléatoires de longueur 1

La figure 4.6 montre un exemple de validation de chemin. La figure montre les configurations aux extrémités du chemin, qui sont valides, et une configuration au milieu du chemin, qui est non-valide. Cet exemple de chemin a une longueur totale $T = 5$. En utilisant la méthode de validation continue, on montre que le chemin n'est pas valide et qu'il existe un intervalle non valide de longueur $d = 0.09$. Si le même chemin est vérifié au moyen d'une validation discrétisée avec un pas de temps supérieur à d, par exemple une valeur par défaut de 0.1, il peut être validé à tort et constituer un faux positif. La méthode continue ne produit jamais de tels faux positifs.

Il convient de noter que pour un chemin donné, il existe toujours un pas de temps suffisamment petit pour que la méthode de validation discrétisée détecte tous les intervalles non valides. Cependant, trouver ce pas de temps n'est pas facile. Réduire le pas de temps se fait également au prix de temps de calcul plus importants puisque, pour la validation discrétisée, le temps de calcul est inversement proportionnel au pas de temps. De plus, le pas de temps est souvent fixé à une valeur par défaut et il n'est pas facile de l'ajuster et d'obtenir un compromis entre rapidité et exactitude.

Les résultats numériques montrent que le majorant $V^{\max}$ de la dérivée de D obtenue dans (4.38) n'est pas un majorant très proche : il y a un facteur de 10^3 entre $V^{\max}$ et les valeurs observées de $\dot{D}$. Pour CoGiRo, les chemins peuvent avoir une longueur allant jusqu'à $T = 15$. Testé sur des chemins aléatoires de longueurs aléatoires, la validation continue prend jusqu'à deux minutes pour valider un chemin, tandis qu'une méthode discrétisée avec un pas de temps par défaut de 0.01 prend 0.5 secondes au maximum. Bien que la différence de temps de calcul soit importante, la méthode continue offre des garanties qui sont cruciales dans un cadre industriel et qu'une méthode discrétisée n'offre pas.

En raison de la nature dichotomique de l'algorithme de validation, lorsque le robot s'approche d'une configuration non valide sans l'atteindre, la validation prend plus de temps car les intervalles validés deviennent de plus en plus petits. Cela signifie que les trajectoires plus simples sont en général plus rapides à valider. Pour CoGiRo, la majorité des chemins testées qui ont été trouvés non-valides vis-à-vis des tensions des câbles sont également non-valides par rapport aux collisions de câbles. Cela montre que l'espace de travail de CoGiRo est davantage limité par les collisions de câbles que par les limites sur les tensions des câbles. En effet, sa géométrie a été choisie afin de maximiser la capacité du robot à soulever des charges lourdes possiblement décalé tout en respectant les contraintes minimum et maximales de tension dans les câbles [1].

4.7 Conclusion

Ce chapitre a exposé les contraintes s'appliquant sur les tensions des câbles et les méthodes utilisées pour répondre à la question de la validité d'une configuration. La méthode de décalage d'hyperplans permet d'identifier un système d'inégalités qui conditionnent la validité de la configuration. Chaque inégalité est reliée à un hyperplan porteur du zonotope du AWS, et constitue un critère de validation pour la validation continue présentée au Chapitre 2. La Section 4.5 détaille les calculs permettant de déterminer un intervalle valide autour d'un paramètre donné le long d'une trajectoire. Ces calculs s'appuient sur le calcul de majorants des termes de l'inégalité.

Les tests d'implémentation révèlent que la méthode permet en effet de trouver tous les cas de violation des contraintes de tensions des câbles. Cela se fait cependant au prix de temps de calcul élevés par rapport à la méthode discrétisée, qui sont dues aux marges prises lors des majorations des termes des inégalités. Afin d'améliorer ces performances, nous pouvons envisager d'améliorer le calcul du majorant $V^{\max}$ présenté dans la section 4.5.3. Calculer un majorant plus près des valeurs réelles de $V^{\max}$, par exemple avec l'analyse par intervalle, pourrait permettre d'obtenir de meilleurs temps de calcul.

Dans les calculs présentés dans la partie 4.5, le torseur des efforts est supposé constant au cours d'un mouvement. Parmi les pistes d'améliorations, il nous parait nécessaire d'étendre la méthode de validation continue des tensions au cas où le torseur des efforts est variable, pour prendre par exemple en compte les cas où un bras robotique est fixé sur la plateforme.

Il faut également noter que cette méthode de validation continue serait plus pertinente pour des robots dont l'espace de travail n'a pas été optimisé vis-à-vis des tensions comme celui du robot CoGiRo. En effet, pour CoGiRo, la grande majorité des trajectoires aléatoires générées lors des tests sont valides vis-à-vis des tensions.

Il est important de noter que pour un RPC suspendu tel que CoGiRo, tous les câbles ne sont en réalité pas tendus en même temps. En effet, en supposant les câbles infiniment raide (pas d'élasticité), au maximum 6 câbles sont tendus à un instant donné. Déterminer l'ensemble des câbles tendus n'est pas trivial [147] mais peut être nécessaire pour la commande des RPCs afin de garantir la précision du positionnement de la plateforme. Dans cette thèse, les câbles sont considérés tous tendus afin de démontrer l'existence d'une distribution de tension possible. Le choix de cette distribution et sa mise en oeuvre sont laissés à la commande du robot dans une étape ultérieure.

En conclusion, afin de tirer profit des avantages de la méthode de validation continue tout en gardant un temps de calcul raisonnable, une solution pour la planification de mouvement consiste à tout d'abord utiliser la méthode discrétisée, afin d'obtenir un résultat rapide. La trajectoire ainsi obtenue est ensuite validée ou invalidée grâce à la méthode de validation continue. Si la trajectoire n'est pas valide, l'algorithme de planification est relancé avec une nouvelle graine aléatoire pour obtenir un nouveau chemin. Au regard de la grande proportion de chemins valides dans l'espace de travail, cette manière de faire permet de générer des trajectoires continûment valides tout en limitant les temps de calcul.

Chapitre 5

Planification de mouvements de manipulation

Sommaire

5.1 Introduction

Nous avons vu dans les chapitres précédents comment valider continûment une trajectoire déjà générée. Le présent chapitre s'attache à montrer comment générer ces trajectoires en premier lieu, en présentant quelques méthodes de génération de trajectoires en robotique. Les RPCs peuvent bénéficier des mêmes algorithmes de planification par échantillonnage aléatoires utilisés pour des robots classiques. Lors de la planification avec un RPC, un problème supplémentaire peut néanmoins survenir : certaines configurations valides tirées lors de la planification ne sont pas joignables à cause des collisions de câbles. L'espace des configurations valides est donc composé de différentes composantes connexes qui ne sont pas joignables entre elles.

La section 5.2 présente le problème de planification de mouvements et deux algorithmes de planification de mouvements. La section 5.3 présente le problème de planification de tâches ainsi qu'une méthodologie pour la modélisation de la manipulation d'objets. Enfin, la section 5.4 formule et détaille le problème des composantes connexes propre aux RPCs.

Contributions de cette thèse :

- Utilisation des algorithmes de planification pour mettre en avant et étudier le problème des composantes connexes.
- Propositions de solutions pour le problème des composantes connexes.

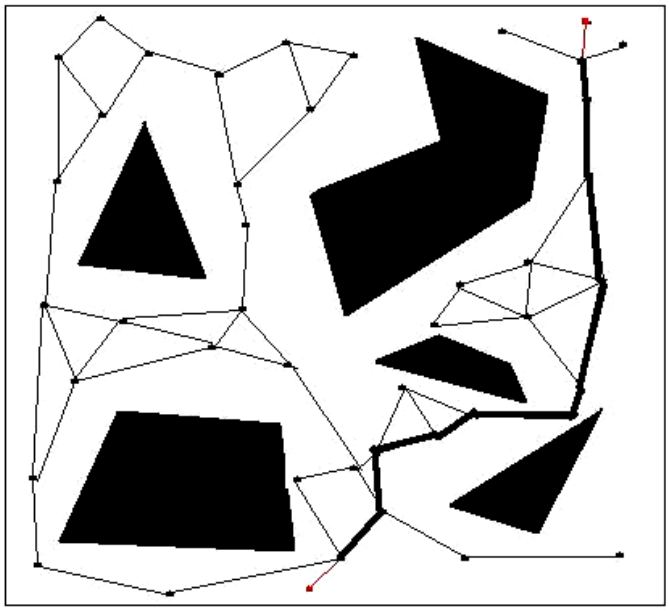


Figure 5.1 – Graphe créé par la méthode PRM, selon [7]. Le trait en gras représente une solution trouvée pour relier une configuration initiale et une configuration finale.

5.2 Algorithmes de planification

La planification de mouvement consiste à trouver une trajectoire valide d'une configuration initiale du robot jusqu'à une configuration finale donnée. Une trajectoire est valide si elle ne comporte pas de collision, et dans le cas des RPCs, si elle est respecte les contraintes des tensions des câbles. La notion d'espace des configuration, noté $\mathcal{CS}$, est présentée dans la section 1.2.1. Une configuration de $\mathcal{CS}$ donne la position et l'orientation de chaque corps du robot. Le problème de planification de mouvement consiste à trouver une courbe continue pour un point dans $\mathcal{CS}$.

Il existe trois grandes familles de méthodes de planification de mouvement : les méthodes déterministes, les approches probabilistes, et les approches basées sur l'optimisation. Dans cette thèse, nous nous intéressons aux approches probabilistes.

5.2.1 Probabilistic RoadMap (PRM)

L'algorithme PRM est un algorithme proposé par Kavraki [89] qui procède en deux phases. La première phase est une phrase d'apprentissage, au cours de laquelle la carte (*roadmap*) des configurations est construite. La deuxième phase est une phase de requêtes, lors de laquelle des problèmes de planification sont résolus.

5.2.1.a Phase d'apprentissage

La phase d'apprentissage construit un graphe non-orienté R aussi appelé la carte probabiliste (*probabilistic roadmap*), composé d'un ensemble de sommets N (les configurations) et d'un ensemble d'arêtes E (les chemins reliant ces configurations).

La construction du graphe nécessite plusieurs prérequis, notamment :

- une méthode de tirage aléatoire de configurations dans $\mathcal{CS}$,

Algorithme 4 Étape de construction d'une graphe PRM

```
1: function PRMConstruction(K)
2:     N ← ∅
3:     E ← ∅
4:     for i = 1 ... K do
5:         while not IsValid(q_rand) do
6:             q_rand ← Random_Configuration(CS)
7:         end while
8:         N_{q_rand} ← set of candidate neighbors of q_rand chosen from N
9:         add q_rand to N
10:        for each q ∈ N_{q_rand}, in order of increasing distance to q_rand do
11:            if the path (q_rand, q) is valid then
12:                add the path (q_rand, q) to E
13:                Update(R)
14:            end if
15:        end for
16:    end for
17: end function
```

- un planificateur local (*local planer*) déterministe permettant de calculer rapidement un chemin simple entre deux configurations (par exemple une interpolation linéaire entre les configurations, ou des courbes de Reeds-Shepp pour une voiture par exemple),
- une méthode de validation permettant de tester la validité des chemins simples générés par le planificateur local,
- une métrique D.

L'ensemble des arêtes E n'est pas stocké en mémoire, car le planificateur local permet de recalculer rapidement les arêtes. Les composantes connexes de R sont en revanche gardées en mémoire.

La construction du graphe, détaillée dans l'algorithme 4, vise à obtenir un graphe suffisamment connecté, avec assez de configurations pour couvrir de manière à peu près uniforme $\mathcal{CS}_{\text{free}}$. Au début de l'algorithme, l'ensemble des configurations N est vide. Un nombre K de configurations sont ensuite tirées aléatoirement. L'algorithme tire des configurations et les teste avec la fonction `IsValid` jusqu'à obtenir une configuration dans $\mathcal{CS}_{\text{free}}$ (lignes 5-7). Pour chaque configuration valide $\mathbf{q}_{\text{rand}}$ tirée, l'ensemble $N_{\mathbf{q}_{\text{rand}}}$ des configurations voisines de $\mathbf{q}_{\text{rand}}$ est défini comme étant l'ensemble des configurations dont la distance à $\mathbf{q}_{\text{rand}}$ est inférieure à une distance seuil c (ligne 8). Pour chaque configuration $\mathbf{q}$ de $N_{\mathbf{q}_{\text{rand}}}$, dans l'ordre croissant de distance à $\mathbf{q}_{\text{rand}}$ (ligne 10), nous essayons de relier $\mathbf{q}$ et $\mathbf{q}_{\text{rand}}$ grâce au planificateur local (ligne 11). Si le chemin obtenu est valide, le chemin est ajouté au graphe (ligne 12) et les composantes connexes de R sont mises à jour (ligne 13).

Dans certaines implémentations, une étape d'expansion suit la construction du graphe et permet d'améliorer sa connectivité. Ce n'est pas le cas dans l'implémentation dans le logiciel HPP.

Dans la variante k-PRM, pour une configuration aléatoire tirée, les k configurations voisines les plus proches sont choisies. Dans la variante PRM* [12], à chaque

tirage, l'ensemble des configurations voisines de la configuration tirée n'est pas défini comme l'ensemble des configurations situées à une distance inférieure à une valeur fixe c, mais comme l'ensemble des configurations situées à une distance inférieure à une fonction $c(n)$ où n est le nombre de configurations dans le graphe. La fonction $c(n)$ décroit avec n, le nombre moyen de connections testées étant proportionnel à $\log(n)$. De manière similaire, dans la variante k-PRM*, les $k(n)$ plus proches configurations voisines sont choisies, $k(n)$ étant une fonction décroissante avec le nombre de configurations dans le graphe. L'algorithme PRM, bien que complet en probabilité, n'est pas asymprotiquement optimal. Les variantes PRM* et k-PRM* sont à la fois complètes en probabilité et asymptotiquement optimale [12].

5.2.1.b Phase de requêtes

Une requête de planification est définie par une configuration initiale et une configuration finale. Pour un robot donné et un espace des configurations $\mathcal{CS}$ donné, une fois qu'un graphe R a été construit, de nombreuses requêtes peuvent être effectuées à moindre coût.

Supposons que R ne contient qu'une seule composante connexe. Pour une requête définie par une configuration initiale $\mathbf{q}_{\text{init}}$ et une configuration finale $\mathbf{q}_{\text{goal}}$, les étapes sont les suivantes.

- L'algorithme tente de relier $\mathbf{q}_{\text{init}}$ au graphe. Si cela réussit, la configuration reliée à $\mathbf{q}_{\text{init}}$ est notée $\widetilde{\mathbf{q}_{\text{init}}}$. Le chemin simple reliant $\mathbf{q}_{\text{init}}$ et $\widetilde{\mathbf{q}_{\text{init}}}$ est noté $\mathbf{p}_i$. Si cette étape échoue, la requête échoue.
- De même avec $\mathbf{q}_{\text{goal}}$. L'algorithme tente de relier $\mathbf{q}_{\text{goal}}$ au graphe. Si cela réussit, la configuration reliée à $\mathbf{q}_{\text{goal}}$ est notée $\widetilde{\mathbf{q}_{\text{goal}}}$. Le chemin simple reliant $\widetilde{\mathbf{q}_{\text{goal}}}$ et $\mathbf{q}_{\text{goal}}$ est noté $\mathbf{p}_g$. Si cette étape échoue, la requête échoue.
- Une successions de points de passage permettant de relier $\widetilde{\mathbf{q}_{\text{init}}}$ et $\widetilde{\mathbf{q}_{\text{goal}}}$ au sein du graphe est calculée en utilisant l'algorithme de Dijkstra. Le planificateur local permet de reconstruire les chemins entre ces points, et en les concaténant, on obtiens un chemin $\mathbf{p}$ entre $\widetilde{\mathbf{q}_{\text{init}}}$ et $\widetilde{\mathbf{q}_{\text{goal}}}$.
- Le chemin obtenu en concaténant $\mathbf{p}_i$, $\mathbf{p}$ et $\mathbf{p}_g$ relie donc $\mathbf{q}_{\text{init}}$ et $\mathbf{q}_{\text{goal}}$ et est renvoyé en tant que solution de la requête.

Différentes stratégies permettent de créer les chemins $\mathbf{p}_i$ et $\mathbf{p}_g$ le plus rapidement possible, afin d'avoir un temps de réponse minimal pour chaque requête.

Si R contient plusieurs composantes connexes, comme c'est souvent le cas, les étapes ci-dessus sont répétées pour chaque composante connexe, dans un ordre choisi, jusqu'à ce que l'algorithme réussisse pour une composante ou échoue pour toutes. L'ordre des composantes connexes peut dépendre par exemple d'une distance aux configurations $\mathbf{q}_{\text{init}}$ et $\mathbf{q}_{\text{goal}}$.

L'algorithme PRM est utilisé dans la Section 5.4.2 afin d'étudier la connexité de l'espace de travail. La version implémentée dans le logiciel HPP est la version k-PRM*.

5.2.2 Rapidly-Exploring Random Tree (RRT)

L'algorithme Rapidly-exploring Random Tree (RRT) proposé par LaValle [8] est un algorithme probabiliste de diffusion. Comme pour l'algorithme PRM, RRT nécessite

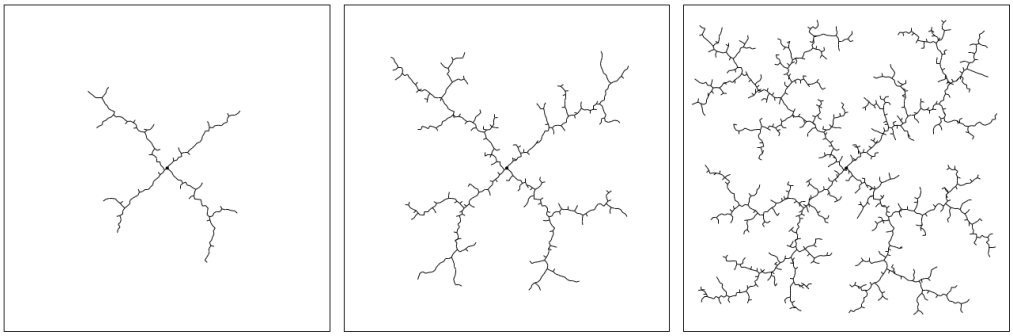


Figure 5.2 – Visualisation de différentes étapes de génération d'un RRT selon [8]

Algorithme 5 Generation of a Rapidly-exploring Random Tree (RRT)

1: **function** BUILDRRT($\mathbf{q}_{\text{init}}, K$)
2: $\mathcal{T}.init(\mathbf{q}_{\text{init}})$
3: **for** $i = 1 \dots K$ **do**
4: $\mathbf{q}_{\text{rand}} \leftarrow$ RANDOM_VALID_CONFIGURATION($\mathcal{CS}$)
5: $\mathbf{q}_{\text{near}} \leftarrow$ NEAREST_NEIGHBOR($\mathbf{q}_{\text{rand}}, \mathcal{T}$)
6: $\mathbf{q}_{\text{new}} \leftarrow$ EXTEND($\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{rand}}, \mathcal{T}$)
7: **if** TRYANDCONNECT($\mathbf{q}_{\text{new}}, \mathbf{q}_{\text{goal}}$) **then**
8: **return** success
9: **end if**
10: **end for**
11: **end function**

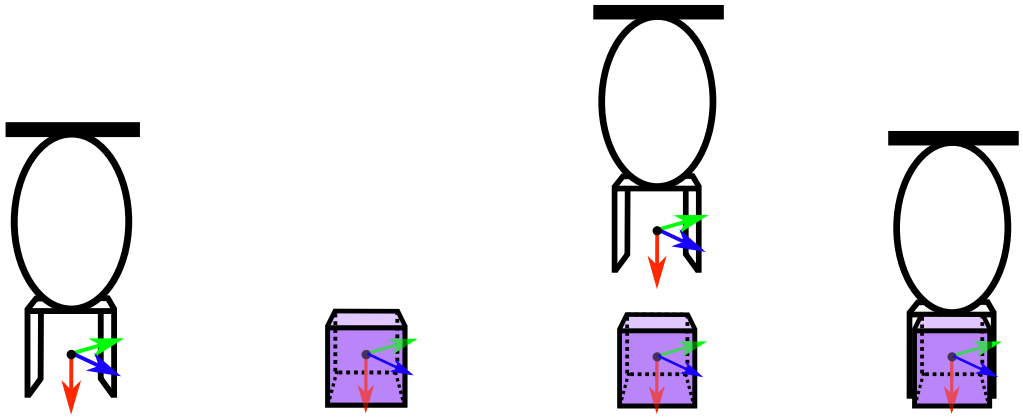
de disposer d'une méthode de tirage de configurations aléatoires, d'un planificateur local ainsi que d'une méthode de validation de chemins et d'une métrique pour déterminer la proximité des configurations. La génération d'un arbre RRT est montrée sur la figure 5.2 et décrite dans l'algorithme 5.

L'arbre $\mathcal{T}$ est initialisé avec la configuration initiale du problème. Ensuite, pour chaque itération, une configuration aléatoire $\mathbf{q}_{\text{rand}}$ dans $\mathcal{CS}$ est tirée (ligne 4). On cherche ensuite $\mathbf{q}_{\text{near}}$ la configuration de l'arbre la plus proche de la nouvelle configuration (ligne 5). Ce plus proche voisin existe toujours puisque $\mathcal{T}$ contient au moins $\mathbf{q}_{\text{init}}$ et n'est pas vide. L'arbre est ensuite étendu, grâce à un planificateur local, de $\mathbf{q}_{\text{near}} \in \mathcal{CS}_{\text{free}}$ vers $\mathbf{q}_{\text{rand}} \in \mathcal{CS}$ et retourne la nouvelle configuration obtenue $\mathbf{q}_{\text{new}} \in \mathcal{CS}_{\text{free}}$ (ligne 6). Après chaque itération, l'algorithme tente de relier $\mathbf{q}_{\text{new}}$ obtenue à la configuration finale $\mathbf{q}_{\text{goal}}$ et s'arrête quand ce teste réussit (ligne 7).

Une méthode classique d'extension est de considérer le *chemin direct* reliant $\mathbf{q}_{\text{near}} \in \mathcal{CS}_{\text{free}}$ et $\mathbf{q}_{\text{rand}} \in \mathcal{CS}$. Ce *chemin direct* est testé et la première partie valide $\mathbf{p}_{\text{valid}}$ est retournée. On ajoute ensuite le chemin $\mathbf{p}_{\text{valid}}$ à la liste des chemins de $\mathcal{T}$ et la configuration finale de $\mathbf{p}_{\text{valid}}$ à la liste des configurations de $\mathcal{T}$. Au lieu de générer un *chemin direct*, d'autres planificateurs locaux (*local planners*) peuvent être utilisés.

L'algorithme RRT présente les avantages suivants.

- L'algorithme RRT permet de gérer des systèmes avec de nombreux degrés de liberté.
- Les arbres RRT possèdent des propriétés intéressantes. L'extension d'un RRT est biaisé en faveur des endroits inexplorés de $\mathcal{CS}$, et la distribution de config-



(a) Préhenseur (b) Poignée (c) Pré-prise (d) Prise

Figure 5.3 – Schéma d'un robot avec préhenseur et d'un objet avec poignée

urations dans un RRT tend vers la distribution d'échantillonnage.

Une variante très courante de RRT est l'algorithme Bi-RRT [148, 149], dans lequel deux arbres de configurations sont construits, un à partir de la configuration initiale et l'autre à partir de la configuration finale. Cette méthode permet d'obtenir des temps de calcul plus faibles.

5.3 Planification de manipulation

Afin de planifier des mouvements de manipulation pour RPCs, nous utilisation la solution au problème de manipulation développée par [94], qui se base sur la création d'un graphe des contraintes représentant les possibilités de manipulation du problème.

5.3.1 Préhenseurs, poignées et surfaces de contact

Une *prise* est définie comme la situation où un objet est attrapé par l'organe terminal d'un robot, la transformation relative de l'objet par rapport à l'organe terminal restant constante au court de la prise. Cette transformation relative définissant la prise doit être renseignée à travers les modélisations du robot et de l'objet. Des *préhenseurs* et des *poignées* sont définis pour le robot et l'objet, respectivement. Un préhenseur et une poignée définissent chacun un repère attaché à leur corps de référence, comme sur la figure 5.3. Par convention, l'axe X de ce repère définit la direction d'approche de la prise. Une valeur de *distance de dégagement* est spécifiée pour chaque préhenseur ou poignée, correspondant à la taille approximative de l'objet. Un plan orthogonal à l'axe X, placé le long de l'axe X à la distance de dégagement de l'origine du repère, ne doit pas entrer en collision avec le corps. Cette valeur permet de définir une *pré-prise*, une configuration où le robot s'est approché de l'objet à attraper, les axes X des repères du préhenseur et de la poignée sont alignées, et le préhenseur à la distance de dégagement de la poignée. Cette pré-prise facilite la manipulation, en définissant une configuration généralement plus facile à atteindre (plus loin de l'obstacle) que la configuration de prise.

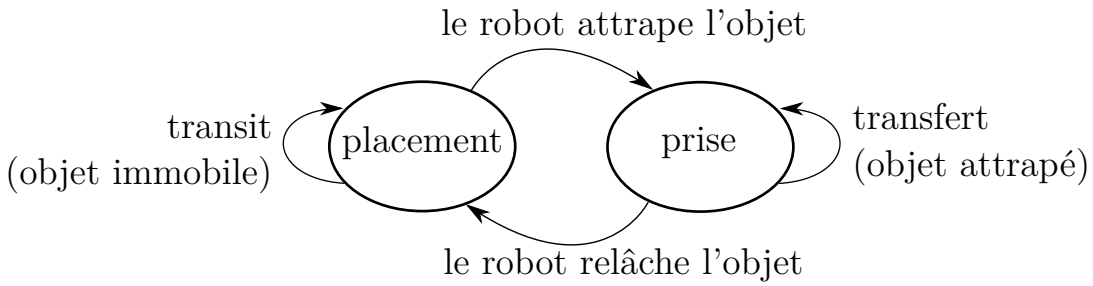


Figure 5.4 – Graphe des contraintes simple pour la manipulation d'un objet

Il est également nécessaire de définir des *surfaces de contact* afin de compléter la modélisation apportée par les préhenseurs et les poignées. Une surface de contact est définie par un polygone plan convexe. Typiquement, des surfaces de contacts sont définies sur les faces des objets, ainsi que sur les différents plans horizontaux sur lesquels les objets doivent être posés comme le sol, une table, etc.

Les données définissant les préhenseurs, poignées et surfaces de contact doivent être fournies par l'utilisateur dans les fichiers de modélisation du robot, de l'environnement et des objets.

5.3.2 Graphe des contraintes

Les relations de prise entre les différents préhenseurs et poignées peut se représenter sous la forme d'un *graphe des contraintes*, dont la figure 5.4 montre un exemple que la section 5.3.2.b décrit plus en détails.

5.3.2.a Définitions

La planification de manipulation est une partie de l'algorithmique du mouvement caractérisée par un certain nombre de contraintes. L'espace des configurations du système composé de robots et d'objets est divisé en états, chacun caractérisé par un ensemble de contraintes non-linéaires. Par exemple, lorsqu'un robot tient un objet avec un préhenseur, la position de l'objet est définie par la position du préhenseur. Tous les autres objets doivent être en équilibre statique. Chaque état est donc caractérisé par un sous-ensemble de paires (préhenseur, objet) qui définit quelle contraintes s'appliquent dans cet état. Les états du systèmes peuvent être connectés par des transitions qui correspondent à des trajectoire faisables du système entre les états, pour donner une structure appelée *graphe des contraintes*.

Graphe des contraintes

Le **graphe des contraintes** représente les contraintes de manipulation d'un problème de manipulation. Les sommets du graphe sont appelés les *états* et les arêtes sont appelées des *transitions*.

État du graphe des contrainte

Chaque sommet S du graphe des contraintes est un *état* défini par un ensemble de contraintes et un ensemble de transitions sortantes. Une configuration $\mathbf{q}$ est dans l'état S si et seulement si $\mathbf{q}$ satisfait la contrainte.

Transition du graphe des contraintes

Chaque arête T du graphe des contraintes est une *transition* définie par une fonction de paramétrisation $\mathbf{f}$, un état d'origine, un état destination, et un état de transition. La transition relie l'état d'origine à l'état destination. L'état de transition représente l'état dans lequel sont les chemins admissibles de la transition.

Chemin admissible d'une transition

Un chemin $\mathbf{p} \in C([0,1], \mathcal{CS})$ est un chemin admissible d'une transition T de fonction de paramétrisation f si et seulement si :

- $\mathbf{p}(0)$ est dans l'état d'origine de T,
- $\forall t \in [0,1], \mathbf{p}(t)$ est dans l'état de transition de T,
- $\forall t \in [0,1], \mathbf{p}(t)$ a la même paramétrisation qu'au paramètre zéro, c'est-à-dire $\mathbf{f}(\mathbf{p}(t)) = \mathbf{f}(\mathbf{p}(0))$.

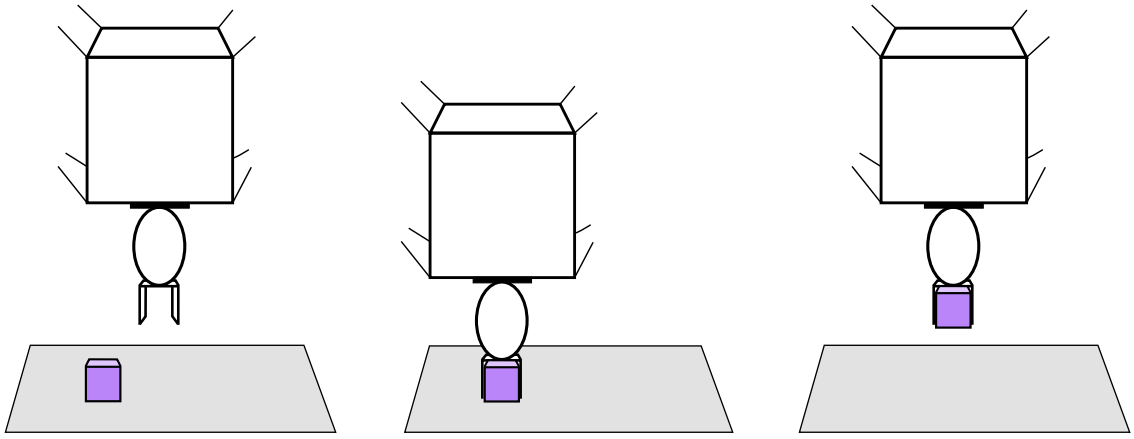
L'état de transition d'une transition T est souvent le même que son état d'origine ou destination, mais peut également être différent.

Lorsqu'une configuration respecte les contraintes de plusieurs états, il existe une incertitude sur la définition de l'état de cette configuration. S'il y a inclusion d'un état dans un autre, alors l'état le plus petit est choisi pour être l'état de la configuration. Si ce n'est pas la cas, l'utilisateur doit spécifier un ordre de priorité selon lequel l'état de la configuration sera choisi.

Projeter une configuration sur un état consiste à appliquer les contraintes de l'état sur cette configuration. Ceci est fait grâce à une méthode de projection, par exemple l'algorithme de Newton-Raphson. Tirer aléatoirement une configuration dans un certain état revient à tirer une configuration aléatoire dans l'espace des configurations, puis à la projeter sur l'état voulu.

5.3.2.b Graphe des contraintes de manipulation

Considérons un exemple de manipulation simple avec un seul objet posé sur le sol, manipulable par un RPC doté d'un organe terminal préhenseur, comme représenté sur la figure 5.5. La figure 5.4 montre le graphe des contraintes correspondant. Le graphe possède deux états : l'état *placement* — l'objet est posé sur le sol — et l'état *prise* — l'objet est pris dans le préhenseur du robot. Les mouvements de *transit*



(a) État *placement* (b) État *prise* et état *placement* (c) État *prise*

Figure 5.5 – Cas simple de manipulation avec un seul objet

qui boucle sur l'état *placement* correspondent aux déplacement du robot tandis que l'objet reste à sa place. Pendant un tel mouvement de *transit*, la fonction de paramétrisation de la transition garde la même valeur tout au long du mouvement, correspondant à la position de l'objet. Lorsque le robot se déplace pour aller attraper l'objet, il passe de l'état *placement* à une configuration se trouvant à la fois dans *placement* et dans *prise*, où l'objet est à la fois attrapé par le robot et en même temps encore posé sur le sol. Le robot peut alors se déplacer tout en gardant l'objet attrapé, qui se détache du sol : ces mouvements sont des mouvements de *transfert*.

La transition *transit* se trouve dans l'état *placement* et la transition *transfert* se trouve dans l'état *prise*. La transition de *placement* à *prise* se trouve dans l'état *placement* car le long d'un tel mouvement, l'objet demeure posé sur le sol : l'état de la transition est son état de d'origine. Lors de la transition inverse, allant de *prise* à *placement*, l'objet reste également posé sur le sol pendant que le robot s'en détache : cette transition se trouve donc également dans l'état *placement*, qui est son état destination.

Notons $\mathcal{CP} \subset \mathcal{CS}$ l'ensemble des configurations correspondant à un placement valide, c'est-à-dire pour lesquelles l'objet est posé en équilibre statique sur le sol, et $\mathcal{CG} \subset \mathcal{CS}$ l'ensemble des configurations correspondant à une prise valide, c'est-à-dire pour lesquelles les repères du préhenseur du robot et de la poignée de l'objet coïncident. Résoudre un problème de manipulation revient à trouver une séquence de transits et de transferts permettant de joindre la configuration initiale à la configuration finale. Le long d'un *transit*, l'objet reste immobile *à la même position*. L'objet pouvant se trouver dans plusieurs positions différentes (typiquement une infinité), cela induit un *feuilletage* de $\mathcal{CP}$. Chaque *feuille* de $\mathcal{CP}$ correspond à une position de l'objet sur le sol. De même, si la prise d'un objet par un robot peut se faire de plusieurs manières (avec une rotation possible si l'objet est cylindrique par exemple), chaque position de prise définit une feuille de $\mathcal{CG}$. Les notions de feuilletage et de feuilles de l'espace des configurations, analysées par [150, 151, 152], sont des notions importantes au sein du problème de manipulation.

5.3.3 Algorithme Manipulation-RRT

L'algorithme Manipulation-RRT (ou M-RRT) proposée par [94] est une variante de l'algorithme RRT adaptée pour fonctionner avec un graphe des contraintes. Il est important de noter que l'algorithme M-RRT utilise deux graphes : le graphe des contraintes comme défini dans la section 5.3.2, ainsi qu'un graphe de configurations que l'algorithme construit au fil des itérations.

Comme pour l'algorithme RRT basique, M-RRT cherche à relier une configuration initiale $\mathbf{q}_{\text{init}}$ et une configuration finale $\mathbf{q}_{\text{goal}}$. Le graphe des configurations est initialisé avec $\mathbf{q}_{\text{init}}$ et pour chaque itération, une configuration aléatoire est tirée et l'algorithme cherche à étendre le graphe des configurations, jusqu'à pouvoir relier $\mathbf{q}_{\text{goal}}$ au graphe. Les étapes de chaque itération pour M-RRT sont les suivantes :

- Une configuration aléatoire $\mathbf{q}_{\text{rand}}$ est tirée.
- Pour chaque composante connexe du graphe des configurations :
  - la configuration la plus proche $\mathbf{q}_{\text{near}}$ est trouvée.
  - L'état de $\mathbf{q}_{\text{near}}$ est déterminé. Une transition sortante de cet état est choisie selon une distribution de probabilité déterminée à l'avance — typiquement, la distribution uniforme.
  - La configuration $\mathbf{q}_{\text{near}}$ est étendue de manière contrainte suivant cette transition vers une configuration $\mathbf{q}_{\text{new}}$. Le procédé d'extension utilise un projecteur de configuration comme défini dans la section 1.2.2 et un projecteur de chemins afin de créer un chemin valide respectant les contraintes de la transition. [94] a proposé deux types de projecteurs de chemins garantissant la continuité du chemin.
- Le planificateur local est utilisé pour essayer de relier les nouvelles configurations $\mathbf{q}_{\text{new}}$ obtenues pour chaque composantes connexes entre elles.
- Si $\mathbf{q}_{\text{init}}$ et $\mathbf{q}_{\text{goal}}$ sont dans la même composante connexe, le problème est résolu. Sinon, l'algorithme démarre une nouvelle itération.

Lorsque les état de prises et de placements possèdent des feuilletages, le problème de *feuilletage croisé* apparaît. Il est impossible d'échantillonner aléatoirement un espace de volume nul telles que sont les feuilles. La probabilité que des arbres de RRT émanant de l'état placement se rejoignent sur la même feuille de prise est donc nulle. Une solution apportée par [94] consiste à rajouter des transitions de feuilletage croisé au graphe des contraintes. Ces transitions s'ajoutent entre les états *placement* et *prise* parallèlement aux transitions normales, et permettent de générer de nouvelles configurations dans des feuilles qui ont été déjà atteintes par des composantes du graphe. Par la suite, nous considérons que les graphes de contraintes sont dotées de telles transitions, mais nous ne les faisons pas apparaître dans les graphes.

L'algorithme M-RRT est utilisé dans le Chapitre 6 afin de réaliser des mouvements de manipulation pour le robot CoGiRo.

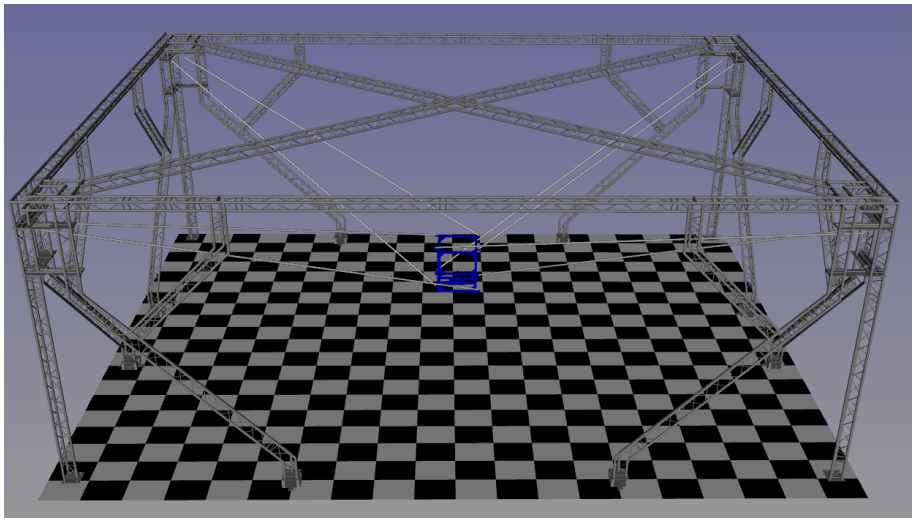


Figure 5.6 – Configuration initiale du RPC CoGiRo

5.4 Problème des composantes connexes

5.4.1 Présentation du problème

Les algorithmes de planification présentés dans la section 5.2 tirent des configurations aléatoirement dans un espace défini afin de construire petit à petit un graphe de chemins valides pour le robot. L'algorithme tente de relier chaque nouvelle configuration tirée aux configurations du graphe en utilisant un planificateur local. Au cours de simulations de planification de mouvements d'un RPC, il a été remarqué que certaines configurations aléatoires tirées, bien que valides du point de vue des collisions et des tensions, ne réussissent pas à être reliées au graphe.

Considérons la position initiale de la plateforme mobile $\mathbf{q}_{\text{init}}$ (Figure 5.6), qui est la configuration de démarrage du robot. Nous définissons les deux types de configurations valides suivantes.

Configuration atteignable

Une configuration valide $\mathbf{q}$ est dite atteignable par rapport à une position initiale $\mathbf{q}_{\text{init}}$ s'il existe un chemin valide reliant $\mathbf{q}$ et $\mathbf{q}_{\text{init}}$.

Configuration non-atteignable

Une configuration valide $\mathbf{q}$ est dite non-atteignable par rapport à une position initiale $\mathbf{q}_{\text{init}}$ s'il n'existe aucun chemin valide reliant $\mathbf{q}$ et $\mathbf{q}_{\text{init}}$.

Pour visualiser ces configurations non-atteignables, imaginons la plateforme d'un robot tel que CoGiRo, tournant sur elle-même selon un axe vertical. La plateforme alterne des courts intervalles de collisions entre câbles et des intervalles de validité plus longs. Après avoir passé une collision, la plateforme se trouve dans une configuration non-atteignable.

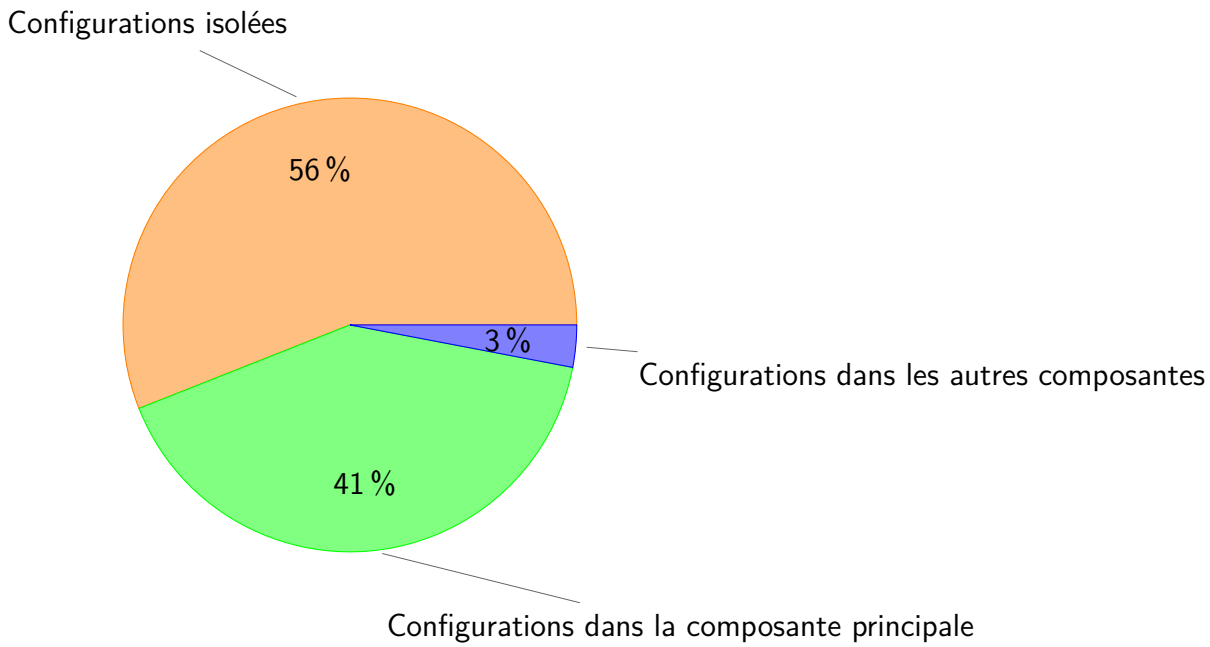


Figure 5.7 – Diagramme circulaire de la répartition des configurations dans les différentes composantes connexes

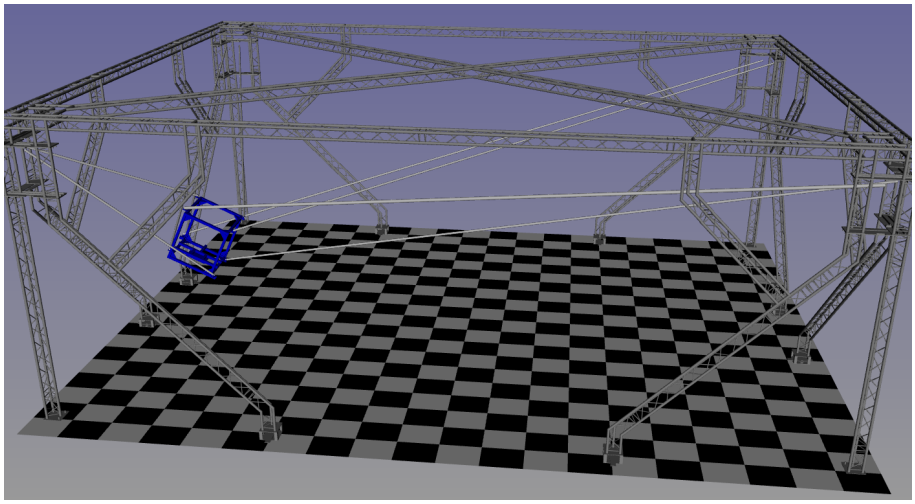
Cela met en évidence l'existence de différentes composantes connexes de l'espace de travail : au sein d'une composante connexe, toutes les configurations sont atteignables entre elles, mais deux configurations situées dans deux composantes connexes différentes ne sont pas atteignables entre elles. La composante contenant la configuration initiale est appelée la composante principale. Cela veut dire qu'une partie de l'espace de travail, valide pour les collisions et les tensions, n'est en réalité pas faisable car ses configurations sont non-atteignables. Réduire le nombre de configurations tirées aléatoirement qui sont non-valides et non-atteignables peut aider à améliorer les performances des planificateurs de mouvements. Ce problème est lié au problème de détermination de l'espace de travail sans collision qui a déjà été traité pour une orientation constante de la plateforme mobile [59].

Il est à noter que les configurations valides représentent une faible part des configurations tirées aléatoirement. Considérons un espace de travail réduit : la position x, y, z de la plateforme est limitée à cube de 1 m de côté au milieu de l'espace de travail total tandis que les rotations ne sont pas limitées. Réduire ainsi l'espace de travail permet de s'affranchir des collisions avec le sol, le plafond et les murs de la structure fixe. Pour $N = 500000$ configurations tirées aléatoirement dans cet espace réduit, 0.9% des configurations sont valides vis-à-vis des collisions. On observe donc que même sans la présence d'objets dans l'espace de travail, les auto-collisions sont des obstacles propres au RPC qui rendent nécessaire l'utilisation d'un planificateur de mouvements comme RRT.

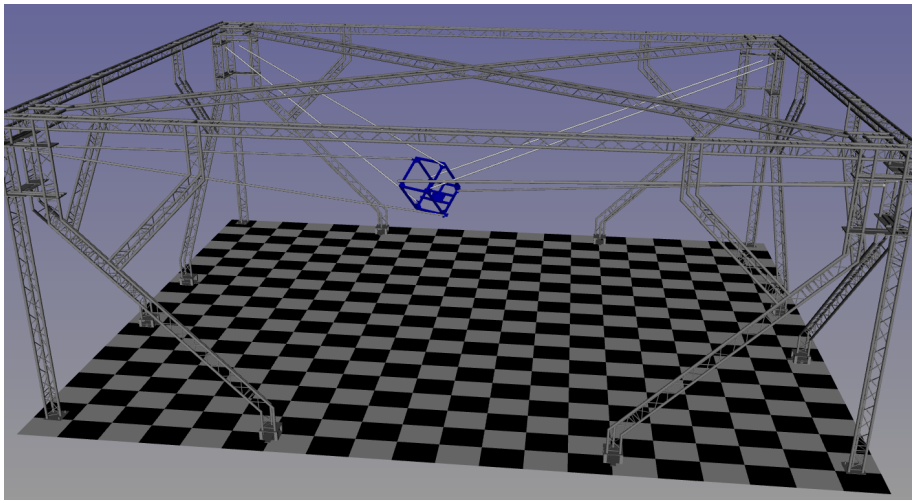
5.4.2 Étude de l'espace de travail

Une étude de l'espace de travail est nécessaire pour juger de l'importance du problème des composantes connexes.

Considérons la plateforme mobile du robot CoGiRo (sans bras robotique ni autre



(a) Configuration isolée



(b) Configuration parallèle

Figure 5.8 – Deux configurations non-atteignables

chargement). Nous analysons l'espace de travail en prenant en compte la détection de collisions et la validation des tensions des câbles. L'objectif de cette analyse statistique de l'espace de travail est de créer un graphe approximant la connectivité de $\mathcal{CS}_{\text{free}}$. Puisque le robot CoGiRo a été conçu de telle sorte à optimiser son espace de travail, nous nous attendons qu'une majorité des configurations de $\mathcal{CS}_{\text{free}}$ se trouvent dans une seule et même composante.

Pour obtenir une approximation du nombre et de la taille des différentes composantes connexes, nous utilisons l'algorithme PRM. La configuration initiale du robot est visible sur la figure 5.6, elle correspond à la position de démarrage du robot. Seule la phase de construction du graphe est lancée, sans phase de requête, l'objectif étant d'obtenir un graphe approximant la connectivité de $\mathcal{CS}_{\text{free}}$. L'algorithme s'arrête une fois que son graphe contient un nombre N donné de configurations.

Considérons premièrement l'espace de travail total. Avec un total de $N = 30000$ configurations dans le graphe, nous obtenons 16984 différentes composantes connexes. La figure 5.7 montre la répartition des configurations. La quasi-totalité (99.8%) des composantes connexes ne contiennent chacune qu'une seule configura-

tion. Nous appelons ces configurations des *configurations isolées*. Elles représentent 56% des configurations totales. Parmi les autres composantes, une composante regroupe 41% des configurations, dont la configuration initiale : c'est la composante principale. Les 3% de configurations restantes sont réparties dans différentes composantes de tailles moyennes (moins de 200 configurations).

Nous observons que les configurations isolées se trouvent en très grande majorité près des bords de l'espace de travail — quand le robot est proche des murs, du sol ou du plafond (Figure 5.8a). Ces configurations reflètent la non-convexité de $\mathcal{CS}_{\text{free}}$, elles pourraient être atteignables si l'espace de travail était plus grand.

Les composantes de tailles moyennes correspondent à des rotations inaccessibles depuis la configuration initiale. Ce sont des configurations atteignables entre elles, mais pour lesquelles certains câbles sont « croisés » par rapport à la configuration initiale (Figure 5.8b). Elles représentent un faible pourcentage de l'espace de travail. Nous appelons ces configurations des *configurations parallèles*.

5.4.3 Solutions aux problèmes des composantes connexes

Plusieurs pistes sont possibles face au problème des composantes connexes.

Une première solution serait de caractériser mathématiquement les configurations non-atteignables. Déterminer si deux configurations ne peuvent pas être reliées sans que des câbles n'entrent en collision permettrait de rejeter les configurations non-atteignables. Nous n'avons aujourd'hui pas réussi à trouver une solution à ce problème pour le cas général où l'orientation de la plateforme n'est pas constante.

Une deuxième solution est de créer un nouveau tireur de configurations aléatoires qui, à la différence du tireur de configurations actuel, tire davantage de configurations valides et atteignables. Nous proposons un tireur utilisant une liste pré-calculée de configurations valides et atteignable. Cette liste peut être obtenue grâce à une des méthodes de planification de mouvement probabiliste, comme PRM, et en sélectionnant toutes les configurations de la composante principale.

Pour tirer une nouvelle configuration, une configuration $\mathbf{q}$ aléatoire est sélectionnée parmi la liste de configurations disponibles, selon une distribution uniforme. Une configuration $\mathbf{q}_{\text{new}}$ est alors tirée aléatoirement selon une loi de distribution gaussienne autour de $\mathbf{q}$.

Un tel tireur de configuration aléatoire possèdent quelques inconvénients. D'abord, une phase de calcul hors-ligne est nécessaire pour chaque nouveau robot ou arrangement des câbles sur la plateforme mobile. D'autre part, ce tireur nécessite de stocker un grand nombre de configurations dans un fichier. Garder ce fichier en mémoire est coûteux en terme de mémoire, et relire ce fichier à chaque appel est coûteux en temps de calcul, même si le reste des calculs (tirage d'une configuration, construction d'un vecteur vitesse aléatoire) se fait rapidement. Enfin, ce tireur aléatoire ne garantit pas de couvrir uniformément ou complétement l'ensemble des configurations atteignables. Ses performances dépendent de la liste des configurations utilisée et des paramètres de la loi de distribution gaussienne utilisée. Cependant, malgré ces inconvénients, un tireur de configuration aléatoire amélioré permettrait de tirer moins de configurations non-valides et d'améliorer les performances des planificateurs de mouvements pour les RPCs. Davantage de travaux sont prévus pour développer et tester ce nouveau tireur de configurations.

5.5 Conclusion

Pour la planification de mouvement et de manipulation, les RPCs peuvent être considérés de la même manière que les robots classiques. Ce chapitre a présenté deux méthodes de planification de mouvement fréquemment utilisées en robotique : l'algorithme PRM et l'algorithme RRT. Afin de générer des mouvements de manipulation, il faut modéliser un certain nombre de caractéristiques du robot et de l'environnement afin de définir des prises, c'est-à-dire des états où le robot attrape un objet. L'algorithme RRT est étendu en Manipulation-RRT afin de gérer les différents états de prise possibles. Cet algorithme sera l'algorithme utilisé dans le chapitre suivant pour résoudre les problèmes de manipulation avec le robot CoGiRo.

La Section 5.4 présente un problème lié à l'existence de plusieurs composantes connexes dans l'espace de travail sans collision d'un RPC. En effet, pour deux configurations sans collision, il est possible qu'il n'existe aucune trajectoire sans collision qui les relie. L'existence de différentes composantes connexes peut ralentir la planification aléatoire de mouvements, en introduisant des configurations inaccessibles depuis la configuration initiale dans le graphe des configurations. Plusieurs pistes de solutions sont évoquées. Cependant, pour le RPC CoGiRo, la composante principale, qui contient la configuration initiale, est suffisamment large par rapports aux autres composantes pour que le ralentissement soit négligeable.

Chapitre 6

Résultats expérimentaux

Sommaire

6.1 Introduction

Les chapitres précédents ont présenté tous les outils nécessaires à la génération de mouvements valides pour RPCs. Le présent chapitre va mettre en oeuvre ces outils sur plusieurs scénarios de manipulation. Ces scénarios vont permettre d'appliquer la méthode de validation continue proposée dans le Chapitre 2 avec la détection de collision du Chapitre 3 et la validation de tension du Chapitre 4.

Les scénarios sont conçus et testés sur le RPC CoGiRo. Pour chaque scénario, des graphes des contraintes sont construits et l'algorithme M-RRT présenté dans le Chapitre 5 est utilisé pour résoudre le problème. La méthode de validation continue de tension développée dans cette thèse ne permet pas de gérer le cas d'un bras robotique fixé à la plateforme. Ainsi, pour les scénarios 1 et 3, les tensions sont validées discrètement tandis que l'absence de collision est validée continûment.

Pour chaque scénario, les propriétés géométriques de manipulation sont présentées, c'est-à-dire les informations des préhenseurs, des poignées et des surfaces de contact pour chaque robot et chaque objet.

Chaque section de ce chapitre présente un scénario différent. La Section 6.2 présente un scénario de peinture de fuselage, la Section 6.3 un scénario de déplacement de palettes, et la Section 6.4 un scénario de mouvements autour d'une rampe. Enfin, la Section 6.5 conclut le chapitre en discutant les résultats obtenus.

Contributions de cette thèse :

- Formulation du problème de manipulation et création du graphe des contraintes pour chacun des trois scénarios présentés.
- Génération de trajectoires et étude statistique des temps de calculs pour chaque scénario.
- Application sur le RPC physique CoGiRo pour le scénario de mouvements autour d'une rampe.

6.2 Scénario 1 : peinture de fuselage

6.2.1 Présentation du scénario

Grâce à leurs grands espaces de travail, les RPCs ont un grand potentiel d'applications dans différents domaines industriels, notamment l'aéronautique. À l'heure actuelle, la peinture des fuselages d'avions se fait majoritairement manuellement. Les ouvriers, portés par des plateforme mécanisées, projettent la peinture avec des outils sur le fuselage (Figure 6.1). Cette tâche est pénible et longue pour les ouvriers.

Avec le scénario présenté dans cette section, nous simulons la peinture d'un fuselage par le RPC CoGiRo. Un cylindre de 4 m de long et 80 cm de diamètre est placé au milieu de l'espace de travail et un bras robotique Yaskawa SIA20 est fixé sous la plateforme mobile du RPC CoGiRo, comme montré sur la figure 6.2. L'objectif est que l'organe terminal du bras robotique suive des trajectoires circulaires autour du cylindre, en restant à une distance fixe, comme schématisé sur la figure 6.3 afin de couvrir une certaine partie de la surface. On considère que le robot doit maintenir son organe terminal à 20 cm du cylindre, et que chaque passe du robot permet de peindre une largeur de 20 cm du fuselage. Le robot doit donc faire $N = 20$ mouvements, c'est-à-dire 10 allers-retours. Après chaque mouvement, nous considérons que le robot arrête de peindre, se déplace pour se mettre en place pour le prochain mouvement, et recommence à peindre en même temps qu'il se met à bouger.

Il n'existe pas dans HPP d'outil de suivi de trajectoire de l'organe terminal ou de couverture de surface. La section 6.2.3 détaille la solution, qui consiste à utiliser un préhenseur virtuel placé au bout du bras robotique et des poignées articulées le long du fuselage. Le problème de peinture de fuselage est ainsi transformé en problème de manipulation.

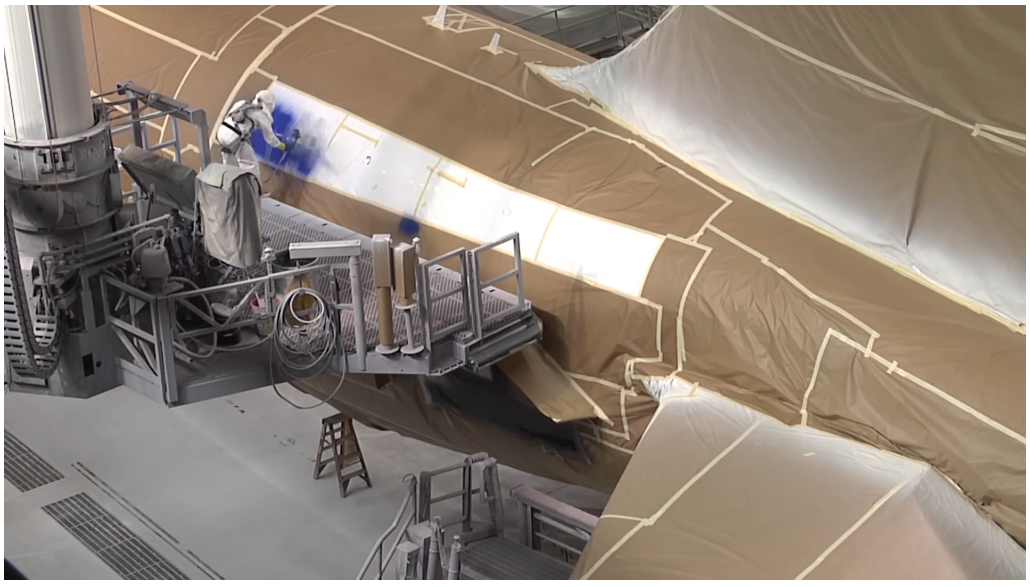


Figure 6.1 – Peinture d'un fuselage d'avion par un ouvrier [9]

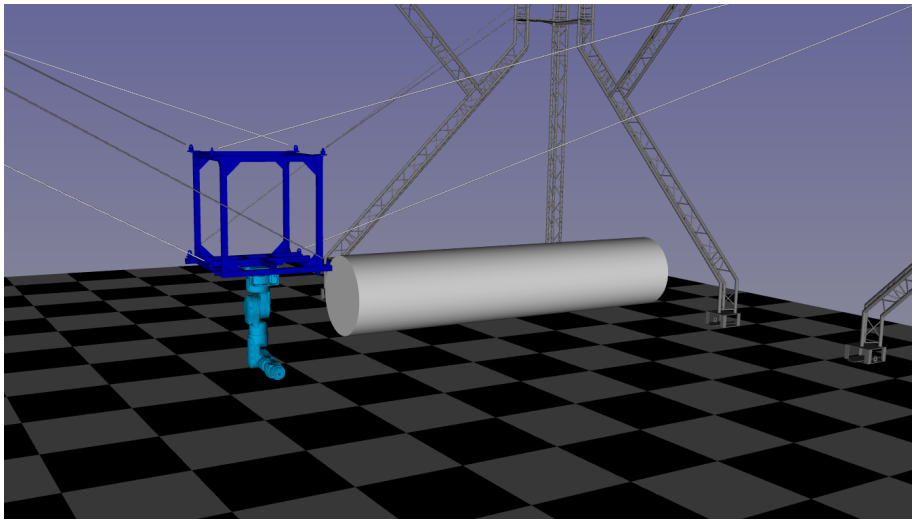


Figure 6.2 – Configuration initiale du scénario de peinture de fuselage, visualisé dans l'interface d'HPP

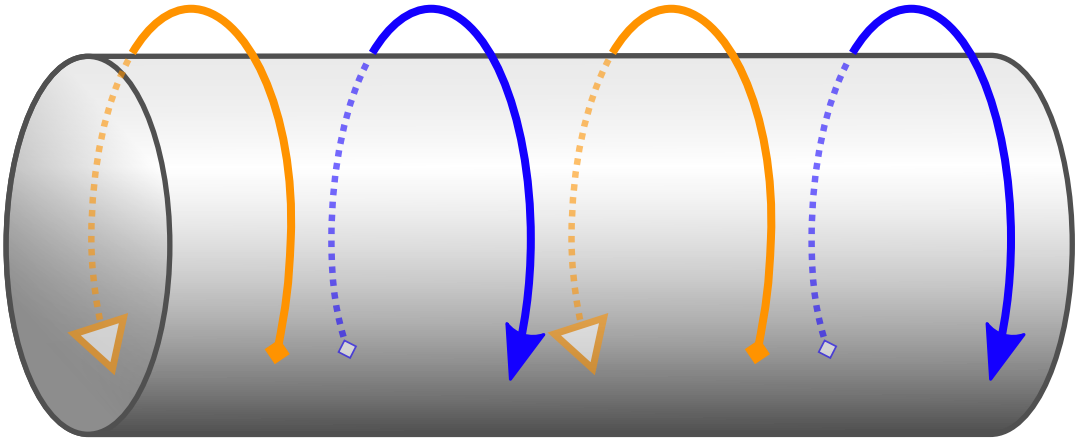


Figure 6.3 – Schéma des mouvements souhaités pour couvrir le fuselage

6.2.2 Propriétés géométriques de manipulation

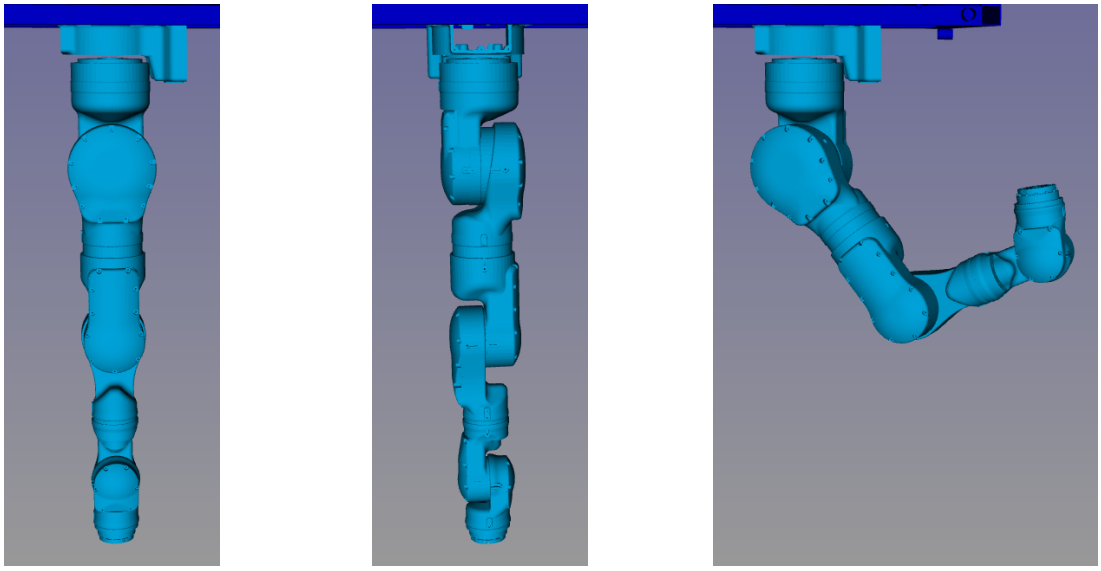
6.2.2.a Plateforme CoGiRo et bras Yaskawa SIA20

Un bras Yaskawa SIA20 est fixé en-dessous de la plateforme mobile du RPC CoGiRo (Figure 6.4). Le bras SIA20 est un bras robotique à 7 degrés de liberté, c'est-à-dire 7 articulations successives. Il peut porter une charge utile de 20 kg à son extrémité.

Un préhenseur est défini au bout de l'organe terminal du bras (Figure 6.5). Dans ce scénario, l'outil de peinture n'est pas modélisé mais pourra être facilement rajouté à l'organe terminal du bras robotique.

6.2.2.b Fuselage

Afin de résoudre le problème de couverture de la surface du cylindre comme un problème de manipulation, des éléments sont rajoutés au modèle du fuselage. Le robot doit suivre le cylindre selon des mouvements circulaires comme montré sur la figure 6.3. Nous considérons N mouvements distincts, chacun consistant en un tour non-complet du cylindre dans le plan vertical, dans un sens ou dans l'autre. Pour



(a) Vue de face (b) Vue de côté (c) Bras plié

Figure 6.4 – Le bras robotique Yaskawa SIA20

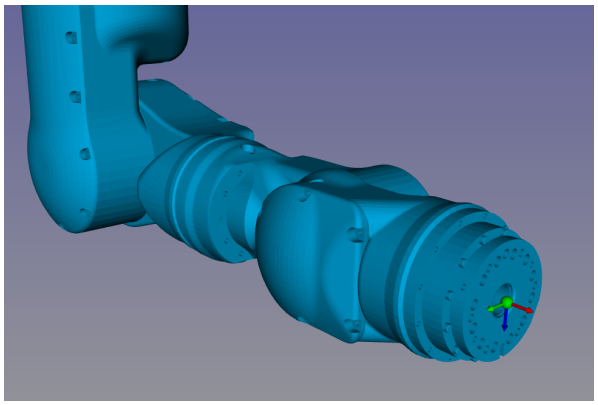


Figure 6.5 – Préhenseur attaché à l'organe terminal du bras SIA20

chaque mouvement $i \in [1, N]$, nous définissons les éléments suivants.

- Un corps virtuel i, sans modèle de collision, appelé *rail*. Par la suite, chaque rail est visualisé par une sphère rose.
- Une articulation pivot autour de l'axe central du cylindre reliant le corps du cylindre et le rail i. Le rail se situe à 20 cm de la surface du cylindre et peut donc pivoter autour du cylindre.
- Une poignée sur le rail i, pouvant être attrapée par le préhenseur du bras robotique.
- Un préhenseur sur le rail i.
- Deux poignées sur le cylindre : une à chaque extrémité du mouvement, au début du mouvement et à la fin du mouvement. Ces poignées peuvent être attrapées par le préhenseur du rail afin de « fixer » le rail à ces positions. Ces poignées sont visualisées par des sphères bleues.

La figure 6.6 schématise ces éléments. Le mouvement commence en (1) quand le préhenseur du robot attrape la poignée du rail, et le préhenseur du rail attrape la

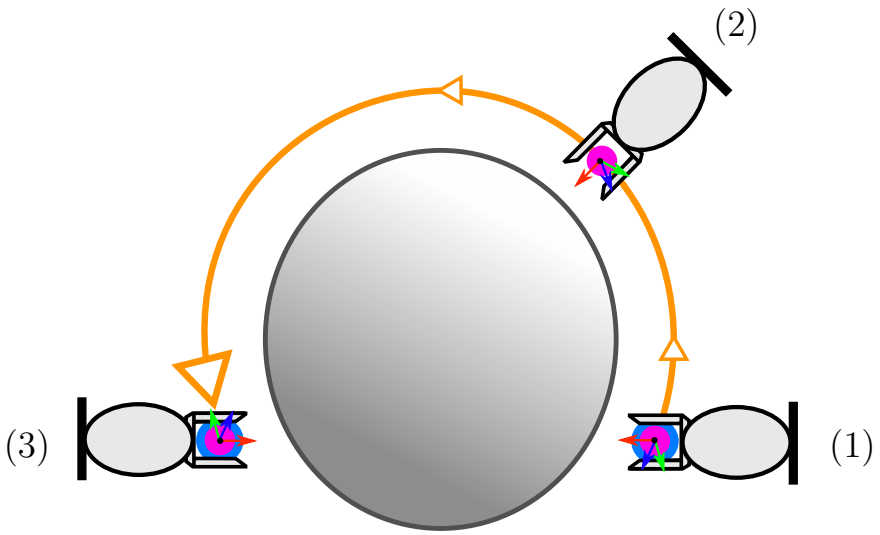


Figure 6.6 – Vue de la tranche du cylindre pour le mouvement i

poignée de début du mouvement. Le robot bouge ensuite, en gardant le rail attrapé, ce dernier faisant alors le tour du cylindre. À la fin du mouvement, le préhenseur du rail attrape la poignée de fin du mouvement. Le robot relâche la poignée du rail.

6.2.3 Graphe des contraintes

Nous avons donc un problème composé de 20 mouvements, chacun nécessitant l'interaction de 2 préhenseurs (robot et rail) et 3 poignées (rail, début du mouvement et fin du mouvement). Nous avons en tout 21 rails et 60 poignées. Vu la nature combinatoire du problème, si aucune contrainte n'est spécifiée, la résolution brute est trop coûteuse en temps pour être possible. La seule tâche de générer le graphe des contraintes complet — permettant à chaque préhenseur d'attraper chaque poignée — est hors de portée de nos capacités informatiques.

Nous divisons donc le problème en sous-problèmes que nous définissons manuellement. Les sous-problèmes sont résolus un à un successivement, chaque configuration finale d'un sous-problème étant la configuration initiale du suivant. Une fois tous les sous-problèmes résolus, il suffit de concaténer les solutions obtenues afin d'obtenir une solution globale.

Chaque mouvement, dans un sens ou dans l'autre autour du cylindre, constitue un sous-problème, pour lequel nous définissons un graphe des contraintes (Figure 6.7). Les noms des états définissent les paires préhenseur-poignée en prise. L'état *libre* correspond à l'état où aucun préhenseur n'attrape de poignée. Entre chaque paire d'états reliés par une transition, il n'y a qu'une seule action de prise ou de relâchement.

La configuration initiale de chaque mouvement i se situe dans l'état *rail-i attrape fuselage/début-i* tandis que la configuration finale se situe dans l'état *rail-i attrape fuselage/fin-i* . Pour chaque mouvement, la configuration initiale du problème est le configuration finale du mouvement précédent — pour le premier mouvement, la configuration initiale globale est utilisée. La configuration finale est tirée aléatoirement dans l'état *rail-i attrape fuselage/fin-i*. L'algorithme M-RRT est ensuite utilisé pour résoudre le problème. Si l'algorithme ne trouve pas de trajectoire valide, une nouvelle configuration de l'état *rail-i attrape fuselage/fin-i* est tirée.

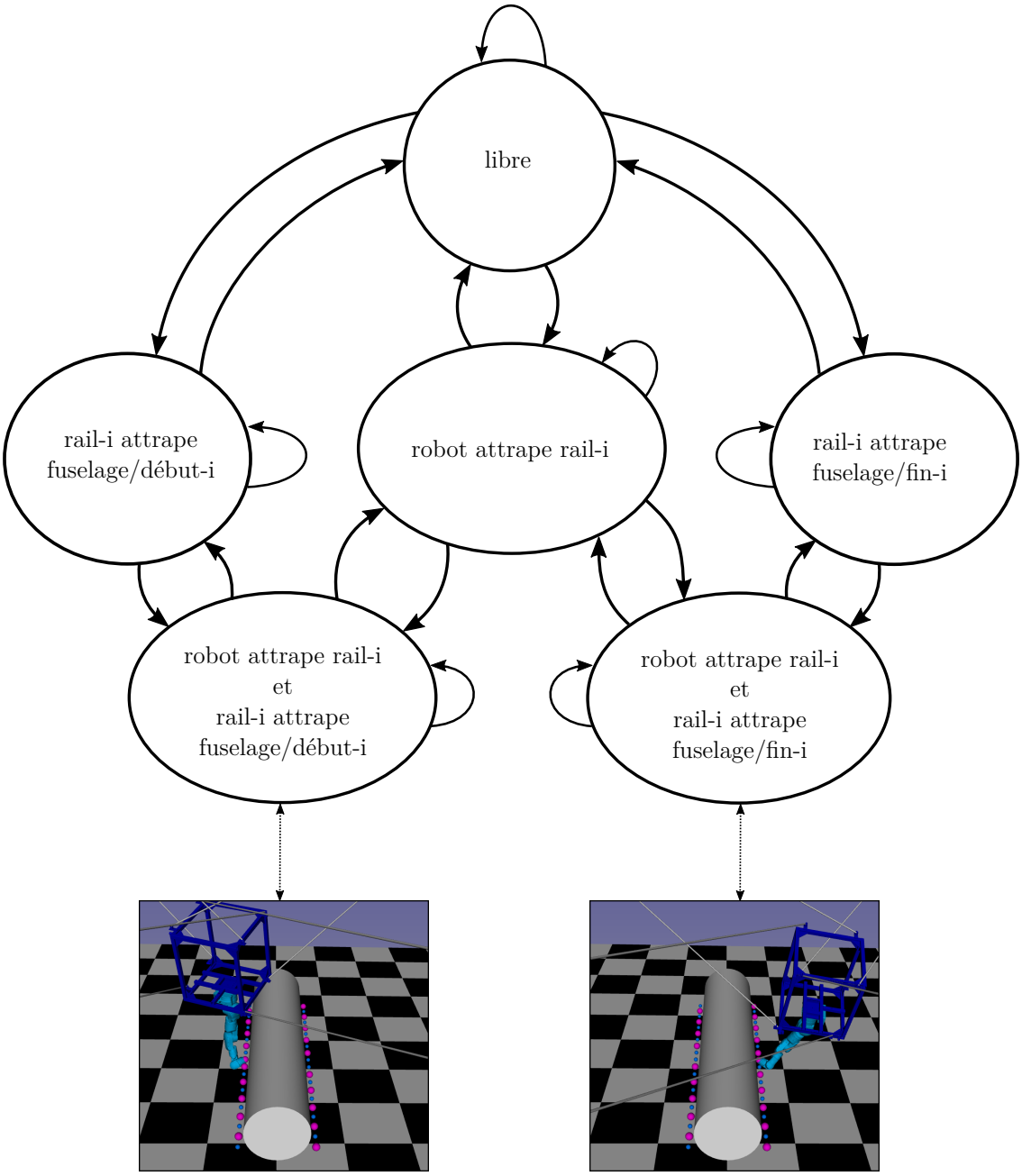


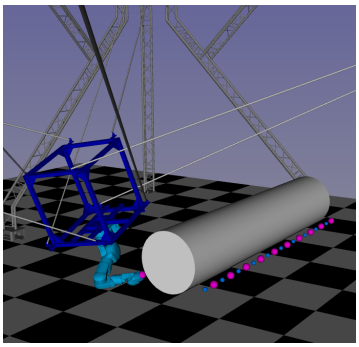
Figure 6.7 – Graphe des contraintes pour le mouvement i du scénario de peinture de fuselage. Deux configurations correspondant à deux états du graphe sont montrées pour $i = 10$.

6.2.4 Résultats

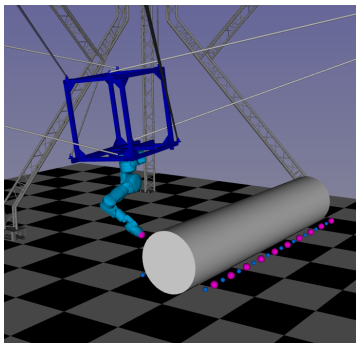
Le tableau 6.1 montre les temps de calcul pour plusieurs générations de trajectoire pour le mouvement 1 du scénario (première rotation autour du cylindre). Chaque trajectoire est générée en utilisant l'algorithme M-RRT et une méthode de validation hybride : la validation continue pour les collisions proposée dans la section 3.4 et une validation discrète pour les limites des tensions, en utilisant la méthode de décalage d'hyperplans sur un ensemble de configurations échantillonnées le long du

Nombre de trajectoires	Min.	Moy.	Max.	Écart type
20	18.2	50.56	130.12	30.51

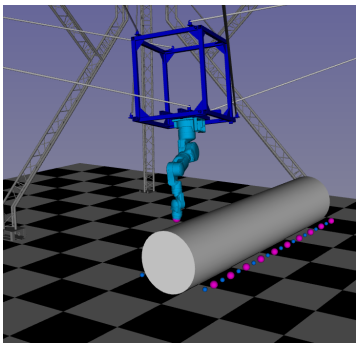
Tableau 6.1 – Temps de calcul en secondes pour $N_{\text{tests}} = 20$ planifications de trajectoire pour un mouvement du scénario de peinture de fuselage



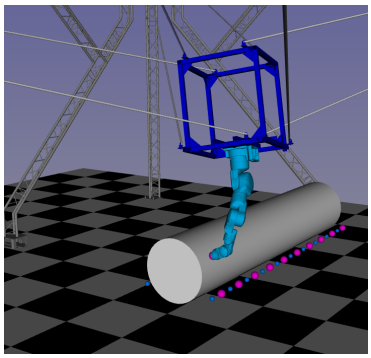
(a) robot attrape rail-0 et rail-0 attrape fuselage/début-0



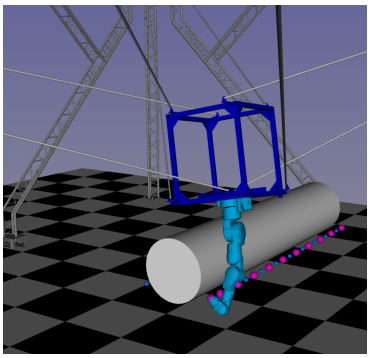
(b) robot attrape rail-0



(c) robot attrape rail-0



(d) robot attrape rail-0



(e) robot attrape rail-0 et rail-0 attrape fuselage/fin-0

Figure 6.8 – Configurations le long du premier mouvement de peinture du fuselage

chemin. Il faut noter qu'il est possible que pour un sous-problème, la configuration initiale et la configuration finale tirée aléatoirement soient difficilement joignables à cause des limites articulaires du bras robotique, ou bien non-joignables si elles sont dans deux composantes connexes différentes du graphe des configurations. Nous avons mis en place deux solutions : d'une part, nous fixons un temps de calcul maximal $T^{\max}$ par essai. Si le temps maximal est atteint, l'algorithme s'arrête, tirent une nouvelle configuration finale, et lance à nouveau l'algorithme M-RRT avec la nouvelle configuration finale. Ici, au vu des temps obtenus, nous avons choisi $T^{\max} = 200s$. D'autre part, l'avis de l'utilisateur est demandé à chaque tirage de nouvelle configuration. L'utilisateur visualise la configuration et peut choisir de l'utiliser pour la résolution du problème, ou de l'ignorer et d'en tirer une autre. Cette étape est rapide et permet d'obtenir des trajectoires satisfaisantes.

La figure 6.8 montre certaines configurations du robot au cours du mouvement. Ce scénario n'a pas encore été testé sur le robot réel. Le calcul de chaque mouvement se fait en environ 20 secondes.

6.3 Scénario 2 : déplacement de palettes

6.3.1 Présentation du scénario

Les RPCs peuvent porter des charges lourdes au sein d'un vaste espace de travail, ce qui les rend particulièrement adaptés à des utilisations en manutention et en logistique. Le scénario détaillé dans cette section est une tâche de manipulation de palettes chargées comme ce qui peut être attendu d'un RPC en entrepôt. La figure 6.9 montre les différentes positions des palettes. Les palettes, numérotées de 1 à 3, sont initialement posées sur le sol (1). Le robot doit les empiler les unes sur les autres en tour (2) ou en pyramide (3). Chaque configuration est définie par les positions dans l'espace de chaque palette.

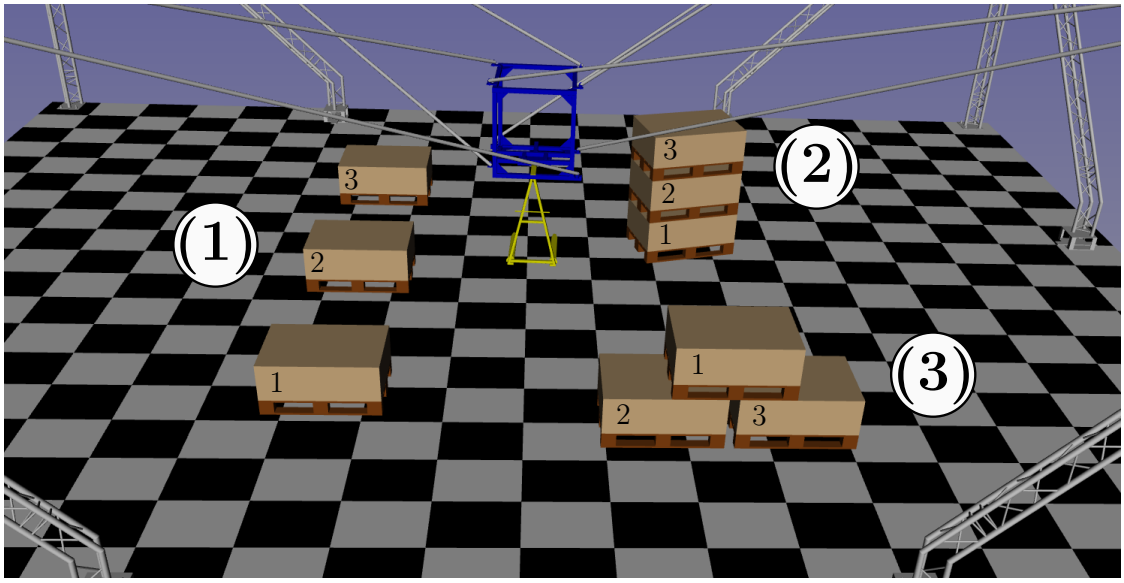


Figure 6.9 – Configurations des palettes : configuration initiale (1), configuration empilée (2), configuration en pyramide (3)

Ce scénario est un problème de manipulation de palettes. Certains éléments du problème peuvent également être modélisés par les outils de planification de manipulation : nous considérons les positions relatives des palettes empilées comme des contraintes de manipulation.

6.3.2 Propriétés géométriques de manipulation

6.3.2.a Plateforme CoGiRo

Pour ce scénario, la plateforme du RPC CoGiRo est dotée d'un lève-palette. Un préhenseur est défini entre les deux branches du lève-palette, tel que montré sur la figure 6.10.

6.3.2.b Palettes

Les palettes utilisées sont des Palettes Europe « EPAL », un type de palette particulièrement répandu en manutention dans le monde et en Europe [153]. Chaque palette est surmontée d'une boîte lestée de 40 cm pour une masse totale de 110 kg.

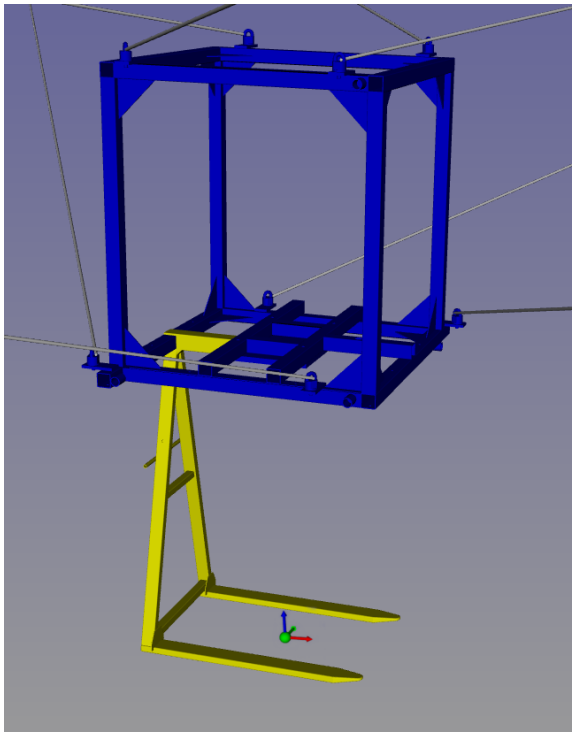
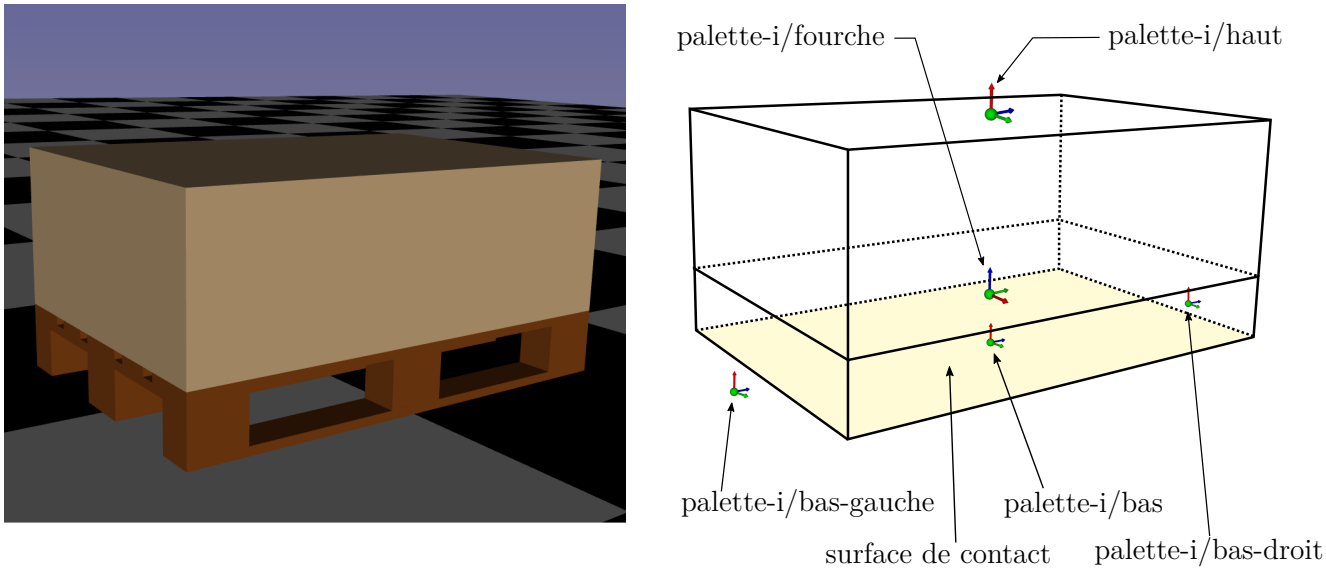


Figure 6.10 – La plateforme mobile de CoGiRo dotée d'un lève-palette. Un préhenseur est défini au milieu des branches du lève-palette.

Afin de modéliser le lève-palette attrapant la palette ainsi que le placement relatif des palettes lorsqu'elles sont empilées, nous définissons les éléments suivants pour chaque palette i (Figure 6.11).

- Une poignée est placée dans la palette afin d'être attrapée par le préhenseur du lève-palette attaché à CoGiRo. Nous nommons cette poignée « palette-i/fourche ».
- Un préhenseur est placé en haut de la boîte de la palette, nommé « palette-i/haut ».
- Une poignée est placée au bas de la palettes, de telle sorte que si elle est attrapée par le préhenseur « haut » d'une autre palette, les deux palettes soient empilées. Cette poignée est nommée « palette-i/bas ».
- Deux poignées sont placées en bas de la palette, de chaque côté (en dehors du modèle physique de la palette), de telle sorte que si les préhenseurs de 2 palettes côte à côte attrapent chacun une de ces poignées, la troisième palette soit empilée à cheval sur les deux palettes. Ces poignées sont nommées « palette-i/bas-gauche » et « palette-i/bas-droit ».
- Une surface de contact est définie sur la surface inférieure de la palette.

Il est à noter que pour ce scénario, il aurait été possible de définir des surfaces de contacts sur le dessus des boîtes des palettes, afin de pouvoir modéliser le placement d'une palette sur une autre dans un empilement. Nous avons choisi d'utiliser plutôt des paires de préhenseurs et poignées, car ces derniers permettent d'imposer une position relative unique entre les deux palettes, ce qui n'est pas possible avec les surfaces de contact.



(a) Vue de face

(b) Vue de côté

Figure 6.11 – Une palette i surmontée d'une boîte avec son préhenseur et ses poignées

6.3.2.c Environnement

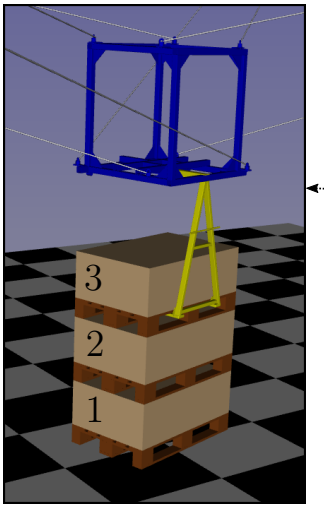
Considérons le repère orthonormé fixe de référence $\mathscr{F}_0$ posé au sol. L'environnement de ce scénario se compose du sol. Une surface de contact est définie au sol en $z = 0$ afin de pouvoir définir la contrainte de placement au sol d'une palette dotée d'une surface de contact.

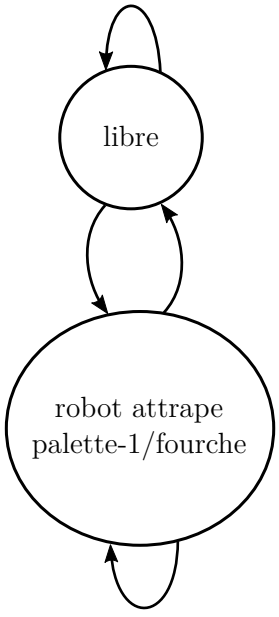
6.3.3 Graphes des contraintes

Avec une configuration initiale et une configuration finale données, nous planifions la liste des palettes qui doivent être successivement manipulées. le mouvement de chaque palette, nous déterminons quelles prises entre en jeu afin de déterminer le graphe des contraintes correspondant.

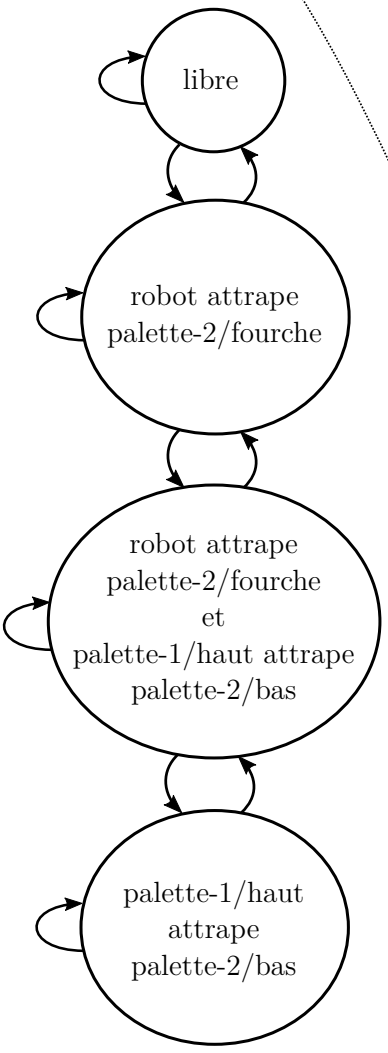
Pour le mouvement d'empilage des palettes en tour, passant de la configuration (1) à la configuration (2) de la figure 6.9, le robot manipule successivement les palettes 1, 2 et 3. Pour la manipulation de la palette i, la paire cogiro + palette-i/fourche est nécessaire. Pour la manipulation palette 2, la paire palette-1/haut + palette-2/bas est également nécessaire. La manipulation de la palette 3 requiert la paire palette-1/haut + palette-2/bas ainsi que la paire la paire palette-2/haut + palette-3/bas. La figure 6.12 montre les trois graphes des contraintes correspondants. L'état *libre* n'existe pas pour la manipulation de la palette 3 car l'empilement des palettes 1 et 2 est imposé.

Pour le mouvement d'empilage des palettes en pyramide, passant de la configuration (1) à la configuration (3), la manipulation des palettes 1 et 2 ne requiert que la paire robot + palette-i/fourche. Pour la manipulation de la palette 3, les paires palette-1/haut + palette-3/bas-gauche et palette-1/haut + palette-3/bas-droit permettent de modéliser la position en pyramide. Les graphes correspondant se construisent de la même manière que pour le mouvement d'empilage en tour. Ce scénario sera bientôt testé sur le RPC CoGiRo.

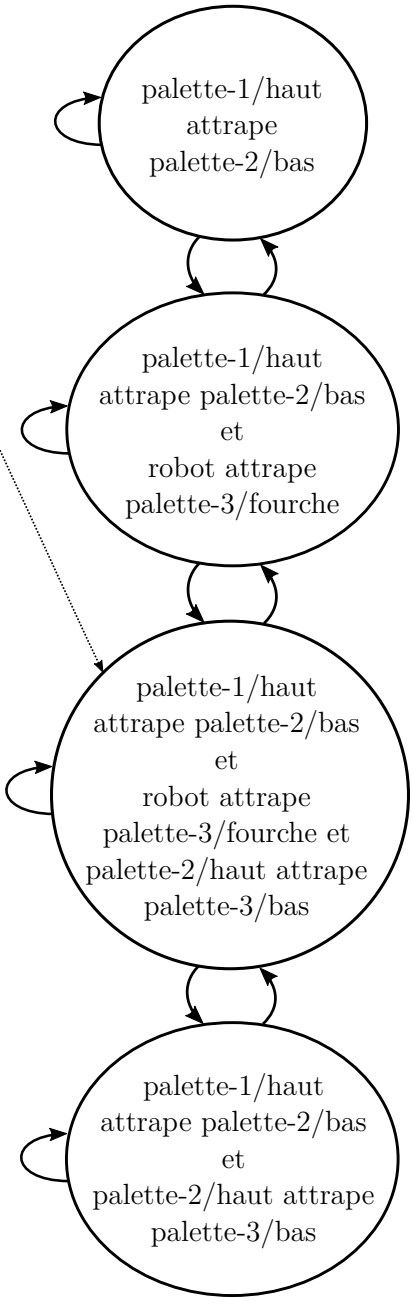




(a) Palette 1



(b) Palette 2



(c) Palette 3

Figure 6.12 – Graphes des contraintes pour la manipulation de chaque palette pour l'empilage en tour. Une visualisation d'un des états est montrée.

Nombre de trajectoires	Min.	Moy.	Max.	Écart type
20	197.09	410.81	707.47	165.43

Tableau 6.2 – Temps de calcul en secondes pour $N_{\text{tests}} = 20$ planifications de trajectoire pour le scénario d'empilage des palettes en tour

6.3.4 Résultats

Considérons le problème d'empilage en tour, avec pour la configuration initiales les palettes disposées sur le sol avec différentes orientation (figure 6.13).

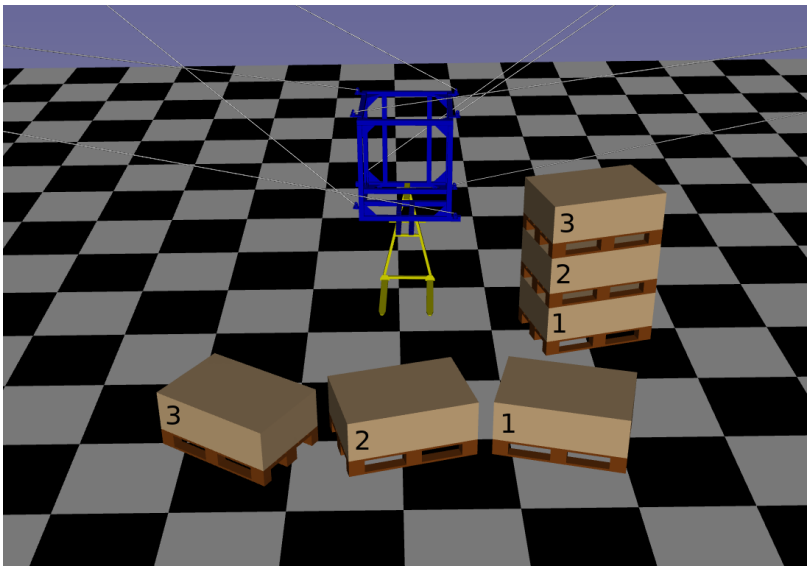


Figure 6.13 – Configurations des palettes : configuration initiale sur le sol et configuration objectif empilée en tour

Pour résoudre le problème, une trajectoire est générée en utilisant l'algorithme M-RRT et la méthode de validation continue globale pour RPCs présentée dans cette thèse, qui prend en compte les collisions et les limites des tensions. Le tableau 6.2 montre les temps de calculs pour $N_{\text{tests}} = 20$ générations de trajectoires pour l'empilage. Une trajectoire générée a été testée sur le robot CoGiRo en situation réelle. La figure 6.14 montre quelques images de la démonstration, en indiquant les états des graphes des contraintes correspondant. Le mouvement total d'empilage dure 3 minutes et 30 secondes.

(a) état « libre »

(b) état « robot attrape palette-1/fourche »

(c) état « robot attrape palette-2/fourche »

(d) état « robot attrape palette-3/fourche et palette-1/haut attrape palette-2/bas »

Figure 6.14 – Démonstration du scénario d'empilage de palettes par le robot CoGiRo

6.4 Scénario 3 : mouvements autour d'une rampe

6.4.1 Présentation du scénario

Utiliser un bras robotique fixé sur un RPC permet d'utiliser des outils et de réaliser des tâches variées. Dans ce scénario, une rampe est placée au milieu de l'espace de travail. La tâche du robot est de venir placer son organe terminal à plusieurs endroits autour de la rampe afin de simuler des actions de perçage ou de vissage. La rampe est visualisée sur la figure 6.15.

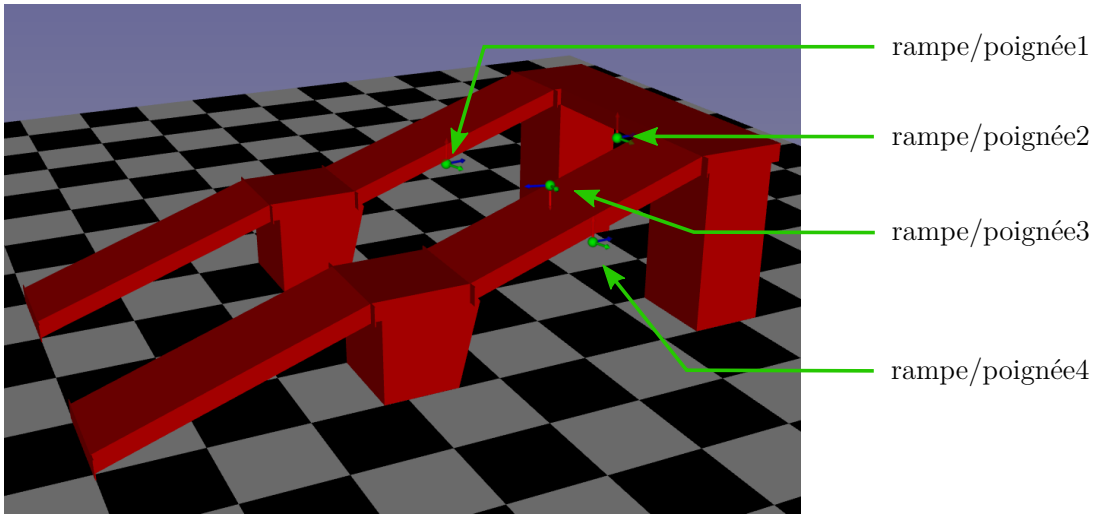


Figure 6.15 – Visualisation de la rampe et de ses poignées

6.4.2 Propriétés géométriques de manipulation

Le bras robotique Yaskawa SIA20 est fixé sous la plateforme du RPC CoGiRo, et un préhenseur est défini sur l'organe terminal du SIA20 comme dans la Section 6.2.2.a.

Des poignées nommée « rampe/poignéeN » pour N=1,2,3,4 sont définies sur le dessus et le dessous de la rampe afin de définir les différentes positions que l'on souhaite que le robot atteigne pendant le mouvement. La figure 6.15 montre ces poignées.

6.4.3 Graphe des contraintes

Le problème de mouvements autour de la rampe est assez simple pour ne pas avoir besoin d'être divisé en sous-problèmes. En effet, il n'y a qu'un seul préhenseur — celui du bras robotique. La figure 6.16 montre le graphe des contraintes du problème. Il y a un état *libre*, puis un état par poignée. Le nombre d'état du graphe est donc linéaire par rapport au nombre de poignées. Pour chaque poignée, une configuration valide dans l'état de prise correspondant est aléatoirement tirée. Le robot commence dans l'état *libre*, transite vers la configuration de l'état correspondant à la prise de la première poignée, retourne dans l'état *libre*, transite vers la configuration de l'état correspondant à la prise de la deuxième poignée, etc.

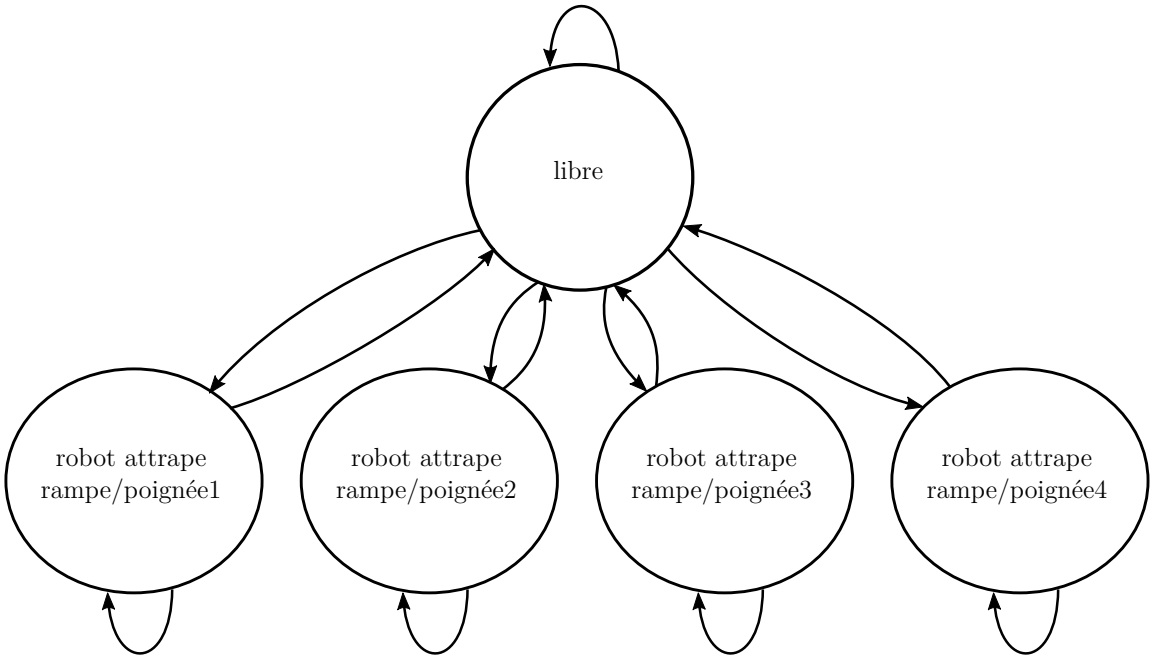


Figure 6.16 – Graphe des contraintes pour le scénario de mouvements autour d'une rampe

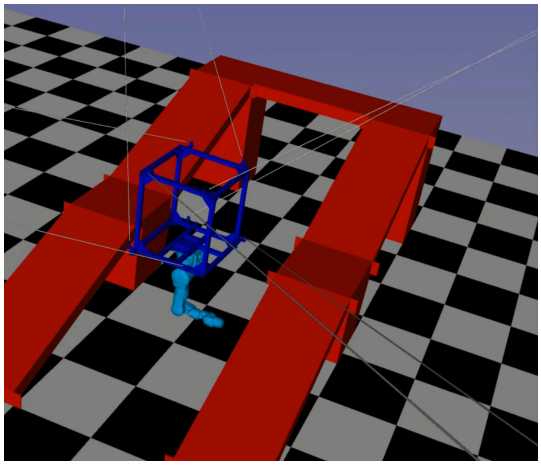
Nombre de trajectoires	Min.	Moy.	Max.	Écart type
20	14.99	40.33	73.62	14.60

Tableau 6.3 – Temps de calcul en secondes pour $N_{\text{tests}} = 20$ planifications de trajectoire pour le scénario de mouvements autour d'une rampe

6.4.4 Résultats

Le mouvement a été généré avec le logiciel HPP, testé en simulation et sur le robot CoGiRo réel. Le tableau 6.3 montre les temps de calculs. Chaque trajectoire est générée en utilisant l'algorithme M-RRT et une méthode de validation hybride : la validation continue pour les collisions proposée dans la section 3.4 et une validation discrète pour les limites des tensions, en utilisant la méthode de décalage d'hyperplans sur un ensemble de configurations échantillonnées le long du chemin.

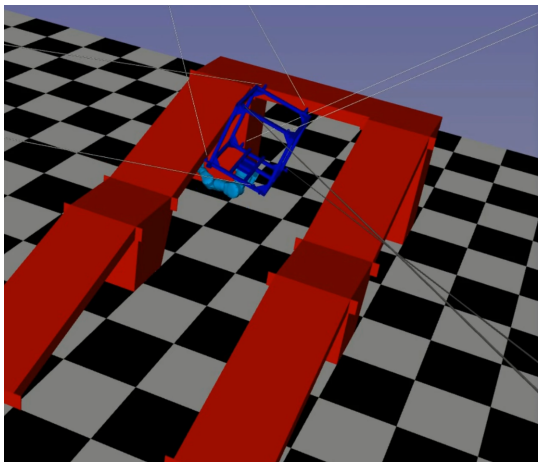
La figure 6.17 montre la position initiale du mouvement. La figure 6.18 montre les différentes configurations obtenues au cours du mouvement. Chaque configuration correspond à une prise d'une des poignées de la rampe par le préhenseur du bras robotique.



(a) CoGiRo vu dans l'interface d'HPP

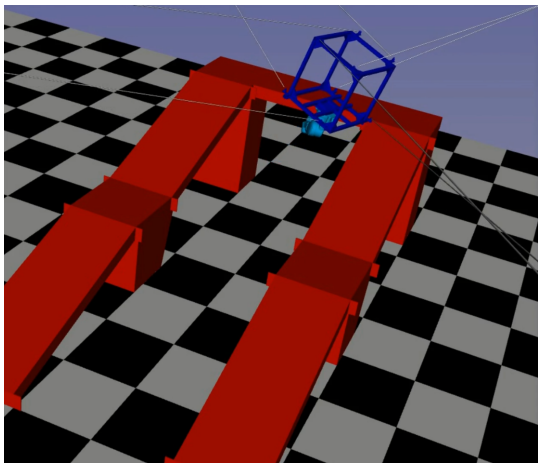
(b) Le RPC CoGiRo équipé du bras robotique Yaskawa SIA20

Figure 6.17 – Configuration initiale du RPC CoGiRo et du bras robotique Yaskawa SIA20



(a) Poignée 1 (HPP)

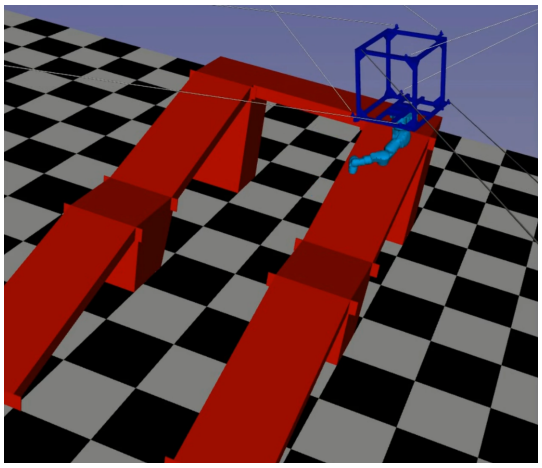
(b) Poignée 1 (réel)



(c) Poignée 2 (HPP)

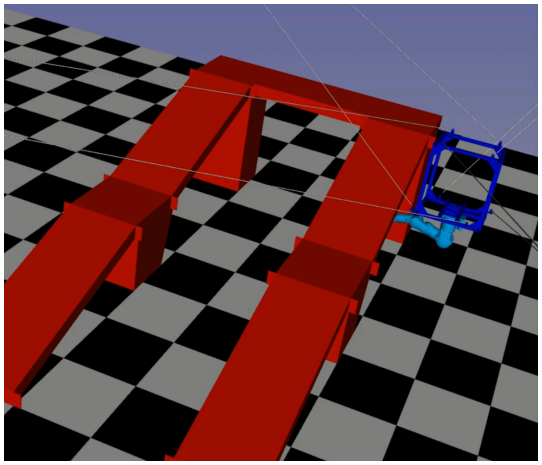
(d) Poignée 2 (réel)

Figure 6.18 – Les différentes configurations du robot CoGiRo pour le scénario de mouvements autour d'une rampe. Chaque configuration correspond à la prise d'une poignée par le préhenseur virtuel du bras robotique.



(e) Poignée 3 (HPP)

(f) Poignée 3 (réel)



(g) Poignée 4 (HPP)

(h) Poignée 4 (réel)

Figure 6.18 – (suite) Les différentes configurations du robot CoGiRo pour le scénario de mouvements autour d'une rampe. Chaque configuration correspond à la prise d'une poignée par le préhenseur virtuel du bras robotique.

6.5 Conclusion

Les trois scénarios présentés dans ce chapitre utilisent la méthode de validation continue de trajectoire et les calculs associés présentés dans les précédents chapitres de cette thèse. Grâce à l'implémentation de la méthode dans le logiciel HPP, la planification de mouvement se fait de manière rapide, de l'ordre de quelques minutes même pour des problèmes complexes. Les mouvements obtenus ont pu être reproduits relativement facilement sur le robot en situation réelle et les résultats montrent l'applicabilité de notre méthode.

La planification automatique de mouvement permet de replanifier facilement le mouvement en cas de changement dans le scénario ou dans l'environnement. En effet, une alternative de la planification automatique est de calculer manuellement les différents points successifs à atteindre, puis faire une interpolation entre ces points. Quelques changements légers (par exemple, modifier la position des points de perçage sur la rampe dans le scénario 3) sont facilement répercutables dans la planification automatique via HPP, mais prendraient considérablement plus de temps à recalculer manuellement. De plus, la planification automatique présentée dans cette thèse permet de réaliser des mouvements qui serait difficiles, voire impossibles à calculer manuellement, d'autant plus en prenant en compte les critères de validité vis-à-vis des tensions et des collisions, et présente donc un avantage indéniable.

Le processus complet de planification reste conséquent — modélisation du robot et de l'environnement, découpage du scénario en sous-problèmes —, mais on peut imaginer à terme une automatisation encore plus poussée du processus, afin de rendre accessible la planification de mouvement à des personnes non-expertes, et ainsi faciliter l'adoption de RPCs dans de nombreux domaines en industrie et ailleurs.

Conclusion

Synthèse

L'objectif de cette thèse était d'appliquer des algorithmes de planification à des robots parallèles à câbles. Durant mes travaux, l'intérêt de pouvoir valider continûment des trajectoires s'est imposé afin de garantir la faisabilité des trajectoires dans un contexte industriel.

Après avoir rappelé l'état de l'art en robotique parallèle à câbles et en planification de mouvement dans le Chapitre 1, un algorithme général de validation continue a été présenté au Chapitre 2 de cette thèse. Cet algorithme est facilement implémentable et hautement versatile, puisqu'il peut prendre en compte n'importe quel type de critère et s'appliquer à n'importe quel type de robot. Les deux critères de validation nécessaires pour les RPCs sont l'absence de collision et la faisabilité de la trajectoire vis-à-vis des tensions des câbles.

Le Chapitre 3 présente la modélisation d'un RPC et le problème de détection de collision pour RPCs. Le critère d'absence de collision pour l'algorithme de validation continue est détaillé. Chaque paire d'objets est testée continûment vis-à-vis des collisions, dont les paires comportant des câbles. Cette méthode s'applique également au cas d'un RPC doté d'un bras robotique pour la manipulation. Les performances obtenues en temps de calcul rivalisent avec les performances d'une méthode discrétisée. Cependant, qu'elles soit discrétisées ou continues, ces méthodes de détection de collision supposent que la modélisation de chaque objet est connue à l'avance. Chaque objet doit être modélisé manuellement, ou automatiquement à l'aide d'une méthode de modélisation d'objets 3D grâce à des caméras ou des lidars.

Dans le Chapitre 4, les contraintes de tensions des câbles sont présentées. L'extension de la méthode de décalage d'hyperplans permet de définir le critère de validation continue en prenant en compte les limites minimales et maximales des tensions des câbles. Les temps de calcul pour la validation continue des tensions sont relativement élevés, mais permettent tout de même de valider ou d'invalider une trajectoire en temps raisonnable.

Les méthodes présentées dans les Chapitres 2, 3 et 4 constituent ensemble une méthode de validation continue complète pour RPCs. Ces méthodes ont été implémentées dans la plateforme logicielle open-source HPP, qui possède tout un ensemble d'algorithmes de planification de mouvements et de méthodes de modélisation de manipulation, comme présenté dans le Chapitre 5. Différents scénarios sont présentés dans le Chapitre 6. L'interface graphique ainsi que l'interface en Python de HPP permettent une prise en main rapide de l'outil. Générer un mouvement revient à modéliser les paires d'objets impliqués dans la manipulation, et à créer un graphe des contraintes correspondant. Le processus entier de génération de mouvements de manipulation pour RPCs devient ainsi simple et rapide.

Perspectives

Amélioration de l'algorithme de validation continue

Plusieurs améliorations pourraient être apportées à l'algorithme de validation continue.

Tout d'abord, il serait intéressant d'étendre la définition d'une configuration d'un RPC afin de prendre en compte la distribution de tensions, en incluant par exemple les indices des câbles tendus. Par rapport à la validation de tension, il serait judicieux d'étendre les calculs au cas où le torseur des efforts requis est variable.

La validation continue de collision présente des temps de calcul suffisamment faibles pour rendre son usage intéressant. Au contraire, la validation continue de tension présente des temps de calculs élevés qui pourraient être réduits en améliorant la précision des calculs de majorants, ce qui pourrait être fait en utilisant des notions d'analyse par intervalle.

Enfin, l'ensemble de cette thèse se place sous l'hypothèse d'un mouvement quasi-statique du système. Plusieurs travaux ont étudié les mouvements dynamiques des RPCs [154, 155, 156]. Il serait intéressant d'étendre les calculs de validation continue des collisions et des tensions des Sections 3.4 et 4.5 aux cas où la dynamique de la plateforme et des objets n'est pas négligée. Cela permettrait de générer des mouvements qui ne sont pas réalisables sous l'hypothèse des mouvements quasi-statiques.

Optimisation des trajectoires vis-à-vis des tensions

Bien que la méthode de validation continue soit exacte et puisse valider une trajectoire concernant les collisions et les tensions des câbles, elle ne garantit pas que la trajectoire soit adaptée pour être effectuée sur le robot physique. Par exemple, on pourrait souhaiter garder le robot le plus loin possible de ses limites d'efforts. Les trajectoires pourraient donc être améliorées en utilisant une méthode de planification basée sur des coûts à optimiser afin par exemple de maximiser la distance du torseur des efforts aux frontières du AWS.

Intégration à un planificateur de tâches

La méthode proposée dans cette thèse requiert que l'utilisateur définisse un graphe des contraintes pour chaque scénario de manipulation. Pour des problèmes simples, l'algorithme M-RRT réussit à trouver des solutions. Pour des problèmes complexes comme ceux présentés dans le chapitre 6, il est nécessaire de guider la recherche du planificateur de mouvements en séparant les problèmes en sous-problèmes, chacun avec un graphe des contraintes contenant un nombre réduit d'états. Cela pourrait être évité en intégrant la méthodologie de manipulation utilisée dans cette thèse dans un planificateur de tâches symbolique [110, 157, 158]. Cela faciliterait d'autant plus la définition de nouveaux problèmes et faciliterait l'usage en industrie d'un planificateur comme HPP.

Intégration à des méthodes de planification dynamique

Dans les méthodes présentées dans cette thèse, les objets de l'environnement sont supposés entièrement connus, et fixes sauf s'ils sont manipulés par le robot. Dans un monde plus réaliste, le robot évolue toujours dans un environnement partiellement inconnu et dynamique. Inclure les éléments de planification développés dans cette thèse dans un framework plus général de planification de mouvement dynamique permettrait de résoudre une gamme plus variée de problèmes, et notamment de pouvoir gérer la présence d'humains dans l'espace de travail. Utiliser des caméras permettrait d'appliquer aux RPCs des méthodes d'asservissement visuel, que ce soit en situation "eye-in-hand" avec une caméra fixée sur la plateforme mobile, ou dans la situation "eye-to-hand" avec des caméras dans l'environnement observant la plateforme. De plus, l'utilisation de marqueurs, comme les marqueurs AprilTag très populaires en robotique, simplifierait le repérage de la plateforme et des objets à manipuler. La position de la plateforme mobile pourrait être déterminée plus précisément en couplant l'estimation provenant de la longueur des câbles avec l'estimation venant d'une caméra et de marqueurs. Sachant que les incertitudes sur la localisation de la plateforme mobile constituent une des limitations majeures à l'utilisation des RPCs, l'amélioration de la localisation permettrait de faciliter le déploiement des RPCs en industrie

Bibliographie

[1] Marc Gouttefarde, Jean-François Collard, Nicolas Riehl, and Cédric Baradat. Geometry selection of a redundantly actuated cable-suspended parallel robot. *IEEE Transactions on Robotics*, 31(2):501–510, 2015. v, 3, 16, 84

[2] Vidéo « Cable robot CoGiRo and Yaskawa SIA20F: Path Following – ANR project DexterWide ». `https://youtu.be/g0_OqK-ZWdU`, 2019. v, 5

[3] Ying Mao and Sunil Kumar Agrawal. Design of a cable-driven arm exoskeleton (CAREX) for neural rehabilitation. *IEEE Transactions on Robotics*, 28(4):922–931, 2012. v, 11, 12

[4] Yousef Baba Zadeh Bedoustani. *Design and control of a cable-driven 6-DOF loading simulator*. PhD thesis, École de technologie supérieure, 2015. v, 13

[5] Vidéo "IAAC & Tecnalia - On Site Robotics". `https://www.youtube.com/watch?v=_j7Ku3udHhk`, 2017. v, 13

[6] Site internet du projet Tensile Truss. `https://www.nist.gov/image/tensiletrussinsidechernobylnscchnppjpg`, 2020. v, 14, 15

[7] Ellips Masehian and Davoud Sedighizadeh. Multi-objective robot motion planning using a particle swarm optimization model. *Journal of Zhejiang University SCIENCE C*, 11(8):607–619, 2010. vi, 89

[8] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998. vi, 2, 20, 83, 91, 92

[9] Vidéo « Painting the ANA Starwars R2D2 Boeing 787 ». `https://www.youtube.com/watch?v=RYyPvXIgKX0`, 2015. vi, 105

[10] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, (5):293–308, 2001. 2

[11] Carole Nissoux, Thierry Siméon, and J-P Laumond. Visibility based probabilistic roadmaps. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)*, volume 3, pages 1316–1321. IEEE, 1999. 2

[12] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. 2, 20, 23, 90, 91

[13] Sepanta Sekhavat, P Švestka, Jean-Paul Laumond, and Mark Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. In *Algorithms for Robotic Motion and Manipulation: 1996 Workshop on the Algorithmic Foundations of Robotics*, pages 79–96, 1997. 2

[14] M. Campana, F. Lamiraux, and J. P. Laumond. A gradient-based path optimization method for motion planning. *Advanced Robotics*, 30(17-18):1126–1144, 2016. 2, 24

[15] Marc Gouttefarde, Jean-François Collard, Nicolas Riehl, and Cédric Baradat. Geometry Selection of a Redundantly Actuated Cable-Suspended Parallel Robot. *IEEE Transactions on Robotics*, 31(2):501–510, February 2015. 3, 13

[16] Site internet du projet CoGiRo. `http://www.lirmm.fr/cogiro/`, 2013. 3

[17] Julien Alexandre dit Sandretto, Gilles Trombettoni, and David Daney. Confirmation of hypothesis on cable properties for cable-driven robots. *New Trends in Mechanism and Machine Science, Mechanisms and Machine Science*, 7, 01 2013. 5

[18] M. Gouttefarde and S. Krut. Characterization of parallel manipulator available wrench set facets. In Jadran Lenarcic and Michael M. Stanisic, editors, *Advances in Robot Kinematics: Motion in Man and Machine*, pages 475–482, Dordrecht, 2010. Springer Netherlands. 5, 70, 71

[19] LL Cone. Skycam: An aerial robotic system. *BYTE, October*, pages 122–132, 1985. 8

[20] Roger Bostelman, James Albus, Nicholas Dagalakis, Adam Jacoff, and John Gross. Applications of the NIST robocrane. In *Proceedings of the 5th International Symposium on Robotics and Manufacturing*, volume 5, 1994. 8

[21] Albus James, Bostelman Roger, and Dagalakis Nicholas. The NIST robocrane. *Journal of Robotic Systems*, 10(5):709–724, 1993. 8

[22] Imme Ebert-Uphoff and Philip A Voglewede. On the connections between cable-driven robots, parallel manipulators and grasping. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4521–4526. IEEE, 2004. 9

[23] M. Gouttefarde and C. M. Gosselin. Analysis of the wrench-closure workspace of planar parallel cable-driven mechanisms. *IEEE Transactions on Robotics*, 22(3):434–445, 2006. 9, 16

[24] Lorenzo Gagliardini, Stéphane Caro, Marc Gouttefarde, and Alexis Girin. Discrete reconfiguration planning for cable-driven parallel robots. *Mechanism and Machine Theory*, 100:313–337, 2016. 10

[25] Lorenzo Gagliardini, Marc Gouttefarde, and Stéphane Caro. Design of reconfigurable cable-driven parallel robots. In *Mechatronics for Cultural Heritage and Civil Engineering*, pages 85–113. Springer, 2018. 10

[26] Dinh Quan Nguyen and Marc Gouttefarde. Study of reconfigurable suspended cable-driven parallel robots for airplane maintenance. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1682–1689. IEEE, 2014. 10

[27] Dinh Quan Nguyen, Marc Gouttefarde, Olivier Company, and François Pierrot. On the analysis of large-dimension reconfigurable suspended cable-driven parallel robots. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 5728–5735. IEEE, 2014. 10

[28] Jonathan Fink, Nathan Michael, Soonkyum Kim, and Vijay Kumar. Planning and control for cooperative manipulation and transportation with aerial robots. *The International Journal of Robotics Research*, 30(3):324–334, 2011. 10

[29] Koushil Sreenath and Vijay Kumar. Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots. *rn*, 1(r2):r3, 2013. 10

[30] Carlo Masone, Heinrich H Bülthoff, and Paolo Stegagno. Cooperative transportation of a payload using quadrotors: A reconfigurable cable-driven parallel robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1623–1630. IEEE, 2016. 10

[31] Yaser N Alothman. *Optimal Control of Multiple Quadrotors for Transporting a Cable Suspended Payload*. PhD thesis, University of Essex, 2018. 10

[32] Khaled Youssef and Martin J-D Otis. Reconfigurable fully constrained cable driven parallel mechanism for avoiding interference between cables. *Mechanism and Machine Theory*, 148:103781, 2020. 10

[33] William E Gordon and TC Kavanaugh. Arecibo observatory. 11

[34] Xiang Cui, Weihai Chen, Xin Jin, and Sunil K Agrawal. Design of a 7-DOF cable-driven arm exoskeleton (CAREX-7) and a controller for dexterous motion training or assistance. *IEEE/ASME Transactions on Mechatronics*, 22(1):161–172, 2016. 11

[35] Xin Jin, Xiang Cui, and Sunil K Agrawal. Design of a cable-driven active leg exoskeleton (c-alex) and gait training experiments with human subjects. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5578–5583. IEEE, 2015. 11

[36] Damiano Zanotto, Giulio Rosati, Simone Minto, and Aldo Rossi. Sophia-3: A semiadaptive cable-driven rehabilitation device with a tilting working plane. *IEEE Transactions on Robotics*, 30(4):974–979, 2014. 11

[37] Giulio Rosati, Paolo Gallina, and Stefano Masiero. Design, implementation and clinical tests of a wire-based robot for neurorehabilitation. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 15(4):560–569, 2007. 11

[38] Philipp Miermeister, Maria Lächele, Rainer Boss, Carlo Masone, Christian Schenk, Joachim Tesch, Michael Kerger, Harald Teufel, Andreas Pott, and Heinrich H Bülthoff. The cablerobot simulator large scale motion platform based on cable robot technology. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3024–3029. IEEE, 2016. 12

[39] Zhaokun Zhang, Zhufeng Shao, Liping Wang, and Albert J Shih. Optimal design of a high-speed pick-and-place cable-driven parallel robot. In *Cable-Driven Parallel Robots*, pages 340–352. Springer, 2018. 13

[40] L Barbazza, F Oscari, S Minto, and G Rosati. Trajectory planning of a suspended cable driven parallel robot with reconfigurable end effector. *Robotics and Computer-Integrated Manufacturing*, 48:1–11, 2017. 13

[41] Alan M Lytle and Kamel S Saidi. NIST research in autonomous construction. *Autonomous Robots*, 22(3):211–221, 2007. 13

[42] Andreas Pott, Hendrick Mütherich, Werner Kraus, Valentine Schmidt, Philipp Miermeister, and Alexander Verl. IPAnema: a family of cable-driven parallel robots for industrial applications. In *Cable-Driven Parallel Robots*, pages 119–134. Springer, 2013. 13

[43] Cyril Alias, Ilia Nikolaev, Eduardo Garduño Correa Magallanes, and Bernd Noche. An overview of warehousing applications based on cable robot technology in logistics. In *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pages 232–239. IEEE, 2018. 13

[44] Nicolò Pedemonte, Tahir Rasheed, David Marquez-Gamez, Philip Long, Étienne Hocquard, Francois Babin, Charlotte Fouché, Guy Caverot, Alexis Girin, and Stéphane Caro. Fastkit: A mobile cable-driven parallel robot for logistics. In *Advances in Robotics Research: From Lab to Market*, pages 141–163. Springer, 2020. 13

[45] Bin Zi, Ning Wang, Sen Qian, and Kunlong Bao. Design, stiffness analysis and experimental study of a cable-driven parallel 3D printer. *Mechanism and Machine Theory*, 132:207–222, 2019. 13

[46] Eric Barnett and Clément Gosselin. Large-scale 3D printing with a cable-suspended robot. *Additive Manufacturing*, 7:27–44, 2015. 13

[47] Jean-Baptiste Izard, Alexandre Dubor, Pierre-Elie Hervé, Edouard Cabay, David Culla, Mariola Rodriguez, and Mikel Barrado. Large-scale 3D printing with cable-driven parallel robots. *Construction Robotics*, 1(1-4):69–76, 2017. 13

[48] Patrick L Swaim, Clark J Thompson, and Perry D Campbell. The Charlotte (TM) intra-vehicular robot. 1994. 14

[49] J Perret and L Dominjon. The Inca 6D: a commercial stringed haptic system suitable for industrial applications. In *Joint Virtual Reality Conference, Springer Tracts in Advanced Robotics*, 2009. 14

[50] Vidéo "NASA's Virtual Reality Laboratory". `https://www.youtube.com/watch?v=LHbhVJ_nP68`, 2015. 14

[51] J-P Merlet. MARIONET, a family of modular wire-driven parallel robots. In *Advances in Robot Kinematics: Motion in Man and Machine*, pages 53–61. Springer, 2010. 14

[52] A. Pott. Determination of the cable span and cable deflection of cable-driven parallel robots. In C. Gosselin, P. Cardou, T. Bruckmann, and A. Pott, editors, *Cable-Driven Parallel Robots*, pages 106–116. Springer, 2017. 15

[53] A. Martin, S. Caro, and P. Cardon. Geometric determination of the cable-cylinder interference regions in the workspace of a cable-driven parallel robot. In C. Gosselin, P. Cardou, T. Bruckmann, and A. Pott, editors, *Cable-Driven Parallel Robots*, pages 117–127. Springer, 2017. 15

[54] M. Fabritius, C. Martin, and A. Pott. Calculation of the collision-free printing workspace for fully-constrained cable-driven parallel robots. In *Proc. ASME International Design Engineering Technical Conferences*, number DETC2018-85961, Québec city, Québec, Canada, 2018. 15

[55] Laurent Blanchet. *Contribution à la modélisation de robots à câbles pour leur commande et leur conception*. PhD thesis, Université de Nice Sophia-Antipolis, 2015. 15, 16

[56] J.-P. Merlet and D. Daney. Legs interference checking of parallel robots over a given workspace or trajectory. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 757–762, Orlando, Florida, may 2006. 15, 16

[57] S. Lahouar, E. Ottaviano, S. Zeghoul, L. Romdhane, and M. Ceccarelli. Collision free path-planning for cable-driven parallel robots. *Robotics and Autonomous Systems*, 57:1083–1093, 2009. 15, 24

[58] J.-P. Merlet. Analysis of the influence of wires interference on the workspace of wire robots. In J. Lenarčič and C. Galletti, editors, *Advances in Robot Kinematics*, pages 211–218, Dordrecht, The Netherlands, 2004. Springer. 16

[59] S. Perreault, P. Cardou, C. Gosselin, and M. Otis. Geometric determination of the interference-free constant-orientation workspace of parallel cable-driven mechanisms. *ASME Journal of Mechanisms and Robotics*, 2(3), 2010. 16, 99

[60] L. Blanchet and J.-P. Merlet. Interference detection for cable-driven parallel robots (CDPRs). In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM '14)*, pages 1413—1418, 2014. 16

[61] D. Q. Nguyen and M. Gouttefarde. On the improvement of cable collision detection algorithms. In T. Bruckmann and A. Pott, editors, *Cable-Driven Parallel Robots*, pages 29–40. Springer, 2014. 16

[62] Guillaume Barrette and Clément Gosselin. Determination of the dynamic workspace of cable-driven planar parallel mechanisms. *Journal of Mechanical Design*, 127:242–248, 03 2005. 16

[63] Alessandro Berti, Jean-Pierre Merlet, and Marco Carricato. Workspace analysis of redundant cable-suspended parallel robots. In Andreas Pott and Tobias Bruckmann, editors, *Cable-Driven Parallel Robots*, pages 41–53, Cham, 2015. Springer International Publishing. 16

[64] G. Abbasnejad, J. Eden, and D. Lau. Generalized ray-based lattice generation and graph representation of wrench-closure workspace for arbitrary cable-driven robots. *IEEE Transactions on Robotics*, 35(1):147–161, 2019. 16

[65] Jean-Pierre Merlet. On the workspace of suspended cable-driven parallel robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 841–846. IEEE, 2016. 16

[66] Ethan Stump and Vijay Kumar. Workspaces of cable-actuated parallel manipulators. *Journal of Mechanical Design*, 128:159–167, 01 2006. 16

[67] X. Diao and O. Ma. Workspace analysis of a 6-DOF cable robot for hardware-in-the-loop dynamic simulation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4103–4108, Oct 2006. 16

[68] J. T. Bryson, X. Jin, and S. K. Agrawal. Optimal design of cable-driven manipulators using particle swarm optimization. *ASME Journal of Mechanisms and Robotics*, 8(4), 2016. 16

[69] Reza Babaghasabha, Mohammad A Khosravi, and Hamid D Taghirad. Adaptive robust control of fully-constrained cable driven parallel robots. *Mechatronics*, 25:27–36, 2015. 16

[70] Xiaoqiang Tang and Zhufeng Shao. Trajectory generation and tracking control of a multi-level hybrid support manipulator in FAST. *Mechatronics*, 23(8):1113–1122, 2013. 16

[71] Salih Abdelaziz, Laurent Barbé, Pierre Renaud, Michel de Mathelin, and Bernard Bayle. Control of cable-driven manipulators in the presence of friction. *Mechanism and Machine Theory*, 107:139–147, 2017. 16

[72] Sana Baklouti, Eric Courteille, Philippe Lemoine, and Stéphane Caro. Vibration reduction of cable-driven parallel robots through elasto-dynamic model-based control. *Mechanism and Machine Theory*, 139:329–345, 2019. 16

[73] Jeremy Begey, Loic Cuvillon, Maximilien Lesellier, Marc Gouttefarde, and Jacques Gangloff. Dynamic control of parallel robots driven by flexible cables and actuated by position-controlled winches. *IEEE Transactions on Robotics*, 35(1):286–293, 2018. 16

[74] Motoji Yamamoto, Noritaka Yanai, and Akira Mohri. Trajectory control of incompletely restrained parallel-wire-suspended mechanism based on inverse dynamics. *IEEE transactions on robotics*, 20(5):840–850, 2004. 16

[75] Hithoshi Kino, Toshiaki Yahiro, Fumiaki Takemura, and Tetsuya Morizono. Robust PD control using adaptive compensation for completely restrained

parallel-wire driven robots: Translational systems using the minimum number of wires under zero-gravity condition. *IEEE Transactions on Robotics*, 23(4):803–812, 2007. 17

[76] Gabriel Meunier, Benoit Boulet, and Meyer Nahon. Control of an overactuated cable-driven parallel mechanism for a radio telescope application. *IEEE transactions on control systems technology*, 17(5):1043–1054, 2009. 17

[77] Marc Gouttefarde, Johann Lamaury, Christopher Reichert, and Tobias Bruckmann. A versatile tension distribution algorithm for n-DOF parallel robots driven by $n+2$ cables. *IEEE Transactions on Robotics*, 31(6):1444–1457, 2015. 17, 70

[78] Damien Gueners, B. Chedli Bouzgarrou, and Hélène Chanal. Static and dynamic analysis of a 6 dof totally constrained cable robot with 8 preloaded cables. In Andreas Pott and Tobias Bruckmann, editors, *Cable-Driven Parallel Robots*, pages 307–318, Cham, 2019. Springer International Publishing. 17

[79] Tomás Lozano-Pérez. *Spatial Planning: A Configuration Space Approach*, pages 259–271. Springer New York, New York, NY, 1990. 17

[80] Edward Pervin and Jon A Webb. Quaternions in computer vision and robotics. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1982. 18

[81] Jernej Barbic. Quaternions and rotations. *University of Southern California, CSCI*, 520, 2011. 18

[82] Osamu Takahashi and Robert J Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2):143–150, 1989. 19

[83] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *Field and service robotics*, pages 203–209. Springer, 1998. 19

[84] Frank Lingelbach. Path planning using probabilistic cell decomposition. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 1, pages 467–472. IEEE, 2004. 19

[85] Yong Koo Hwang, Narendra Ahuja, et al. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992. 19

[86] Shuzhi Sam Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, 13(3):207–222, 2002. 19

[87] John F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, USA, 1988. 19

[88] Amir R Soltani, Hissam Tawfik, John Yannis Goulermas, and Terrence Fernando. Path planning in construction sites: performance evaluation of the Dijkstra, A*, and GA search algorithms. *Advanced engineering informatics*, 16(4):291–303, 2002. 19

[89] Lydia Kavraki, Petr Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12:566 – 580, 09 1996. 20, 89

[90] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000. 20

[91] Roland Geraerts and Mark H Overmars. A comparative study of probabilistic roadmap planners. In *Algorithmic Foundations of Robotics V*, pages 43–57. Springer, 2004. 20

[92] James Kuffner and Steven M. LaValle. Rrt-connect: An efficient approach to single-query path planning. volume 2, pages 995–1001, 01 2000. 20

[93] Sébastien Dalibard, Antonio El Khoury, Florent Lamiraux, Alireza Nakhaei, Michel Taïx, and Jean-Paul Laumond. Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach. *The International Journal of Robotics Research*, 32(9-10):1089–1103, 2013. 20, 22

[94] Joseph Mirabel. *Manipulation planning for documented objects*. Theses, Institut National Polytechnique De Toulouse, February 2017. 20, 22, 93, 97

[95] Matthew Zucker, Nathan Ratliff, Anca Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher Dellin, J. Andrew (Drew) Bagnell, and Siddhartha Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *International Journal of Robotics Research*, May 2013. 21, 23

[96] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. pages 4569–4574, 05 2011. 21, 23

[97] J. Kim and J. Lee. Trajectory optimization with particle swarm optimization for manipulator motion planning. *IEEE Transactions on Industrial Informatics*, 11(3):620–631, 2015. 21

[98] Steven M LaValle, Jeffery H Yakey, and Lydia E Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 3, pages 1671–1676. IEEE, 1999. 22

[99] Terry Taewoong Um, Beobkyoon Kim, Chansoo Suh, and Frank Chongwoo Park. Tangent space RRT with lazy projection: An efficient planning algorithm for constrained motions. In *Advances in Robot Kinematics: Motion in Man and Machine*, pages 251–260. Springer, 2010. 22

[100] Mike Stilman. Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics*, 26(3):576–584, 2010. 22

[101] Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. Manipulation planning on constraint manifolds. In *2009 IEEE International Conference on Robotics and Automation*, pages 625–632. IEEE, 2009. 22

[102] Kris Hauser. Fast interpolation and time-optimization with contact. *The International Journal of Robotics Research*, 33(9):1231–1250, 2014. 22

[103] Rachid Alami, Thierry Simeon, and Jean-Paul Laumond. A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps, 1990. 22

[104] Gordon Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3(1):131–150, 1991. 22

[105] Rachid Alami, Jean-Paul Laumond, and Thierry Siméon. Two manipulation planning algorithms, 1994. 22, 23

[106] Dennis Nieuwenhuisen, A. Frank van der Stappen, and Mark H. Overmars. *An Effective Framework for Path Planning Amidst Movable Obstacles*, pages 87–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. 23

[107] Mike Stilman and James Kuffner. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research*, 27(11-12):1295–1307, 2008. 23

[108] Sébastien Dalibard, Alireza Nakhaei, Florent Lamiraux, and Jean-Paul Laumond. Manipulation of documented objects by a walking humanoid robot. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 518–523. IEEE, 2010. 23

[109] Hai-ning Wu, Martin Levihn, and Mike Stilman. Navigation among movable obstacles in unknown environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1433–1438. IEEE, 2010. 23

[110] Mike Stilman and James J Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503, 2005. 23, 124

[111] Jun Ota. Rearrangement of multiple movable objects - integration of global and local planning methodology. volume 2, pages 1962 – 1967 Vol.2, 01 2004. 23

[112] Jennifer E King, Marco Cognetti, and Siddhartha S Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3940–3947. IEEE, 2016. 23

[113] Puttichai Lertkultanon and Quang-Cuong Pham. A single-query manipulation planner. *CoRR*, abs/1509.00600, 2015. 23

[114] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 3327–3332. IEEE, 2007. 23

[115] Athanasios Krontiris and Kostas Bekris. Dealing with difficult instances of object rearrangement. 07 2015. 23

[116] Yann Labbé, Sergey Zagoruyko, Igor Kalevatykh, Ivan Laptev, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Monte-carlo tree search for efficient visually guided rearrangement planning. *IEEE Robotics and Automation Letters*, 5(2):3715–3722, 2020. 23

[117] K. Harada, T. Tsuji, and J. Laumond. A manipulation motion planner for dual-arm industrial manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 928–934, May 2014. 23

[118] Seyed Sina Mirrazavi Salehian, Nadia Figueroa, and Aude Billard. A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research*, 37(10):1205–1232, 2018. 23

[119] F Basile, F Caccavale, P Chiacchio, J Coppola, and C Curatella. Task-oriented motion planning for multi-arm robotic systems. *Robotics and Computer-Integrated Manufacturing*, 28(5):569–582, 2012. 23

[120] Fabrizio Caccavale and Masaru Uchiyama. Cooperative manipulation. In *Springer handbook of robotics*, pages 989–1006. Springer, 2016. 23

[121] Nikolaus Vahrenkamp, Martin Do, Tamim Asfour, and Rüdiger Dillmann. Integrated grasp and motion planning. In *2010 IEEE International Conference on Robotics and Automation*, pages 2883–2888. IEEE, 2010. 23

[122] Nikolaus Vahrenkamp, Tamim Asfour, and Rudiger Dillmann. Simultaneous grasp and motion planning: Humanoid robot ARMAR-III. *IEEE Robotics & Automation Magazine*, 19(2):43–57, 2012. 23

[123] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE Transactions on Robotics*, 20(6):967–977, Dec 2004. 23

[124] Oriol Bohigas, Montserrat Manubens, and Lluís Ros. Planning wrench-feasible motions for cable-driven hexapods. *IEEE Transactions on Robotics*, 32(2):442–451, 2016. 24

[125] Philipp Tempel, Fabian Schnelle, Andreas Pott, and Peter Eberhard. Design and programming for cable-driven parallel robots in the german pavilion at the EXPO 2015. *Machines*, 3:223–241, 08 2015. 24

[126] Andreas Pott, Tobias Bruckmann, and Lars Mikelsons. Closed-form force distribution for parallel wire robots. In Andrés Kecskeméthy and Andreas Müller, editors, *Computational Kinematics*, pages 25–34, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 24

[127] C. Gosselin, Ping Ren, and S. Foucault. Dynamic trajectory planning of a two-DOF cable-suspended parallel robot. In *2012 IEEE International Conference on Robotics and Automation*, pages 1476–1481, May 2012. 24

[128] Jean-Pierre Merlet. Checking the cable configuration of cable-driven parallel robots on a trajectory. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1586–1591. IEEE, 2014. 24

[129] J. Merlet. Simulation of discrete-time controlled cable-driven parallel robots on a trajectory. *IEEE Transactions on Robotics*, 33(3):675–688, June 2017. 24

[130] Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe. Exact collision checking of robot paths. In J.-D. Boissonnat et al., editor, *Algorithmic Foundations of Robotics V, STAR 7*, pages pp 25–41. Springer, 2004. 36

[131] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation*, 2012. 37, 53, 57

[132] Diane Bury, Jean-Baptiste Izard, Marc Gouttefarde, and Florent Lamiraux. Continuous collision detection for a robotic arm mounted on a cable-driven parallel robot. *arXiv preprint arXiv:1909.10857*, 2019. 43

[133] Site internet de ROS. `https://www.ros.org/`, 2020. 46

[134] Khaled Mamou. V-HACD. `https://github.com/kmammou/v-hacd`, 2012. 49

[135] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. 53

[136] Gino Bergen. Proximity queries and penetration depth computation on 3d game objects. 01 2001. 53

[137] Gino Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–13, 1997. 53

[138] S. Gottschalk, Ming Lin, and Dinesh Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30, 10 1997. 53

[139] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998. 53

[140] Diane Bury, Jean-Baptiste Izard, Marc Gouttefarde, and Florent Lamiraux. Continuous tension validation for cable-driven parallel robots. In *IEEE IROS 2020*, 2020. 66

[141] Marc Gouttefarde. *Analysis and Synthesis of Large-Dimension Cable-Driven Parallel Robots*. Habilitation à diriger des recherches en robotique, Université de Montpellier, Novembre 2016. 196 pages. 67

[142] Per Henrik Borgstrom, Brett L Jordan, Gaurav S Sukhatme, Maxim A Batalin, and William J Kaiser. Rapid computation of optimally safe tension distributions for parallel cable-driven robots. *IEEE Transactions on Robotics*, 25(6):1271–1281, 2009. 70

[143] Tahir Rasheed, Philip Long, David Marquez-Gamez, and Stéphane Caro. Tension distribution algorithm for planar mobile cable-driven parallel robots. In *Cable-Driven Parallel Robots*, pages 268–279. Springer, 2018. 70

[144] Günter M. Ziegler. Lectures on polytopes. Technical Report TR-93-06, ZIB, Takustr. 7, 14195 Berlin, 1994. 70

[145] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998. 71

[146] Marc Gouttefarde and Cle´ ment M Gosselin. On the properties and the determination of the wrench-closure workspace of planar parallel cable-driven mechanisms. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 46954, pages 337–346, 2004. 79

[147] Jean-Pierre Merlet. On the robustness of cable configurations of suspended cable-driven parallel robots. In *14th IFToMM World Congress on the Theory of Machines and Mechanisms*, Taipei, Taiwan, October 2015. 85

[148] XZ Zang, WT Yu, L Zhang, and Sajid Iqbal. Path planning based on BI-RRT algorithm for redundant manipulator. In *2015 International Conference on Electrical, Automation and Mechanical Engineering*. Atlantis Press, 2015. 93

[149] Ahmed Hussain Qureshi and Yasar Ayaz. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015. 93

[150] André Haefliger. Feuilletages sur les variétés ouvertes. *Topology*, 9(2):183–194, 1970. 96

[151] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):729–746, 2004. 96

[152] Joseph Mirabel and Florent Lamiraux. Manipulation planning: addressing the crossed foliation issue. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4032–4037. IEEE, 2017. 96

[153] Site internet des palettes Europe. `https://www.epal-pallets.org/eu-en/load-carriers/epal-euro-pallet/`, 2020. 111

[154] Wei-Jung Shiang, David Cannon, and Jason Gorman. Dynamic analysis of the cable array robotic crane. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 4, pages 2495–2500. IEEE, 1999. 124

[155] Bin Zi, BY Duan, JL Du, and Hong Bao. Dynamic modeling and active control of a cable-suspended parallel robot. *Mechatronics*, 18(1):1–12, 2008. 124

[156] Giovanni Mottola, Clément Gosselin, and Marco Carricato. Dynamically feasible motions of a class of purely-translational cable-suspended parallel robots. *Mechanism and Machine Theory*, 132:193–206, 2019. 124

[157] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014. 124

[158] Raphaël Lallement, Juan Cortés, Mamoun Gharbi, Alexandre Boeuf, Rachid Alami, Carmelo J Fernandez-Agüera, and Iván Maza. Combining assembly planning and geometric task planning. In *Aerial Robotic Manipulation*, pages 299–316. Springer, 2019. 124

Existence du repère local d'un câble

Considérons un chemin $\mathbf{p} : [0,1] \to \mathcal{CS}$. Nous cherchons à prouver qu'il existe le long de $\mathbf{p}$ un repère orthonormé direct local $\mathscr{F}_i\,(B_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$ du câble i centré sur B_i avec $\mathbf{x}_i = \mathbf{d}_i$, et dont le vecteur de vitesse angulaire par rapport au repère global $\mathscr{F}_0$, noté ${}^0\boldsymbol{\omega}_i$, est tel que ${}^0\boldsymbol{\omega}_i{}^T \mathbf{x}_i = 0$, c'est-à-dire que ${}^0\boldsymbol{\omega}_i$ n'a pas de composante le long de $\mathbf{x}_i$. Tous les éléments dépendent du paramètre du chemin $t \in [0,1]$, mais la notation est omise pour la lisibilité.

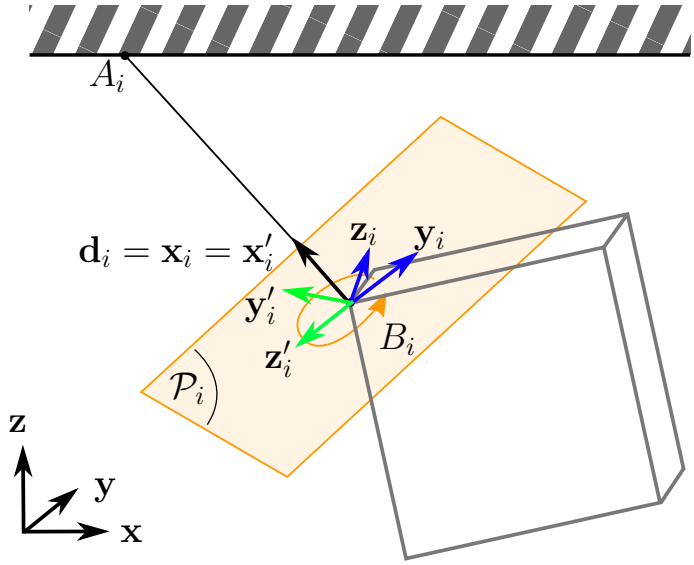


Figure 19 – Schéma du repère local d'un câble

Considérons un repère orthonormé direct $\mathscr{F}'_i\,(B_i, \mathbf{x}'_i, \mathbf{y}'_i, \mathbf{z}'_i)$ tel que

$$\mathbf{x}'_i = \mathbf{d}_i \ , \qquad \mathbf{y}'_i = \frac{\mathbf{y} \times \mathbf{d}_i}{\|\mathbf{y} \times \mathbf{d}_i\|} \quad \text{et} \quad \mathbf{z}'_i = \mathbf{x}'_i \times \mathbf{y}'_i \tag{1}$$

.

Ce repère bouge par rapport à $\mathscr{F}_0$ avec une vitesse linéaire $\mathbf{v}'_i = \mathbf{v}_{B_i}$ et une vitesse angulaire ${}^0\boldsymbol{\omega}_{i'}$.

Considérons un repère $\mathscr{F}_i\,(B_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$ tel que $\mathbf{x}_i = \mathbf{x}'_i = \mathbf{d}_i$. Le repère $\mathscr{F}_i$ tourne autour de l'axe $\mathbf{x}_i$ avec une vitesse angulaire ${}^{i'}\boldsymbol{\omega}_i$ par rapport au repère $\mathscr{F}'_i$. Nous pouvons exprimer cette vitesse angulaire sous la forme suivante.

$$ {}^{i'}\boldsymbol{\omega}_i = \dot{\theta}_i \mathbf{x}_i \tag{2} $$

Nous souhaitons calculer ${}^{i'}\boldsymbol{\omega}_i$ et donc $\dot{\theta}_i$ tel que la vitesse angulaire ${}^{0}\boldsymbol{\omega}_i$ de $\mathscr{F}_i$ par rapport à $\mathscr{F}_0$ n'ait pas de composante selon $\mathbf{x}_i$, c'est-à-dire ${}^{0}\boldsymbol{\omega}_i{}^{T}\mathbf{x}_i = 0$.

$$ \begin{aligned} {}^{0}\boldsymbol{\omega}_i{}^{T}\mathbf{x}_i = 0 &\iff \left({}^{i'}\boldsymbol{\omega}_i + {}^{0}\boldsymbol{\omega}_{i'}\right)^{T}\mathbf{x}_i = 0 \\ &\iff {}^{i'}\boldsymbol{\omega}_i{}^{T}\mathbf{x}_i = -{}^{0}\boldsymbol{\omega}_{i'}{}^{T}\mathbf{x}_i \\ &\iff \dot{\theta}_i = -{}^{0}\boldsymbol{\omega}_{i'}{}^{T}\mathbf{x}_i \end{aligned} \tag{3} $$

D'après les propriétés du produit mixte, nous pouvons écrire :

$$ \begin{aligned} {}^{0}\boldsymbol{\omega}_{i'}{}^{T}\mathbf{x}_i &= {}^{0}\boldsymbol{\omega}_{i'}{}^{T}\left(\mathbf{y}'_i \times \mathbf{z}'_i\right) \\ &= \mathbf{z}'^{T}_i \left({}^{0}\boldsymbol{\omega}_{i'} \times \mathbf{y}'_i\right) \end{aligned} \tag{4} $$

Écrivons la formule de Bour pour le vecteur $\mathbf{y}'_i$ entre les repères $\mathscr{F}_0$ et $\mathscr{F}'_i$:

$$ {}^{0}\dot{\mathbf{y}}'_i = \underbrace{{}^{i'}\dot{\mathbf{y}}'_i}_{=0} + {}^{0}\boldsymbol{\omega}_{i'} \times \mathbf{y}'_i \tag{5} $$

En combinant (3), (4) et (5), on obtient :

$$ \dot{\theta}_i = -\mathbf{z}'^{T}_i \, {}^{0}\dot{\mathbf{y}}'_i \tag{6} $$

L'équation (6) est une équation différentielle admettant une unique solution $\theta_i(t)$. Nous avons donc prouvé l'existence du repère local du câble $\mathscr{F}_i$ dont le vecteur vitesse angulaire n'a pas de composante selon $\mathbf{x}_i$.

Notons que si $\mathbf{y} \times \mathbf{d}_i = 0$, le vecteur $\mathbf{y}'_i$ tel que donné dans (1) n'est plus défini. Dans ce cas, nous considérons un autre repère orthonormé direct $\widetilde{\mathscr{F}}_i(\widetilde{\mathbf{x}_i}, \widetilde{\mathbf{y}_i}, \widetilde{\mathbf{z}_i})$ tel que :

$$ \widetilde{\mathbf{x}_i} = \mathbf{d}_i \ , \qquad \widetilde{\mathbf{y}_i} = \frac{\mathbf{z} \times \mathbf{d}_i}{\|\mathbf{z} \times \mathbf{d}_i\|} \quad \text{et} \quad \widetilde{\mathbf{z}_i} = \widetilde{\mathbf{x}_i} \times \widetilde{\mathbf{y}_i} \tag{7} $$

.

Ce repère est bien défini quand $\mathbf{y} \times \mathbf{d}_i = 0$ car $\mathbf{y}$ et $\mathbf{z}$ sont orthogonaux par définition et il est donc impossible d'avoir en même temps $\mathbf{z} \times \mathbf{d}_i = 0$. Le même raisonnement que précédemment s'applique pour prouver l'existence du repère local du câble $\mathscr{F}_i$.

Propriétés des transformations de corps rigides

Propriété 1

Soit les matrices homogènes $M_1 = \begin{pmatrix} R_1 & T_1 \\ 0\,0\,0 & 1 \end{pmatrix}$, $M_2 = \begin{pmatrix} R_2 & T_2 \\ 0\,0\,0 & 1 \end{pmatrix}$ et $M_3 = \begin{pmatrix} R_3 & T_3 \\ 0\,0\,0 & 1 \end{pmatrix}$ telles que $M_3 = M_1 M_2$. Alors :

$$\|T_3\| \leq \|T_1\| + \|T_2\| \tag{8}$$

Preuve
En effet, si $M_3 = M_1 M_2$, alors :

$$M_3 = \begin{pmatrix} R_1 R_2 & R_1 T_2 + T_1 \\ 0\,0\,0 & 1 \end{pmatrix}$$

Donc $T_3 = R_1 T_2 + T_1$. En appliquant l'inégalité triangulaire et sachant qu'une matrice de rotation a une norme unitaire, nous obtenons la propriété :

$$\|T_3\| \leq \|T_1\| + \|T_2\|$$

Propriété 2

Soit la matrice homogène $M_1 = \begin{pmatrix} R_1 & T_1 \\ 0\,0\,0 & 1 \end{pmatrix}$ et $m \in \mathbb{R}^3$ et $p \in \mathbb{R}^3$ tel que $\begin{pmatrix} p \\ 1 \end{pmatrix} = M_1 \begin{pmatrix} m \\ 1 \end{pmatrix}$. Alors :

$$\|p\| \leq \|m\| + \|T_1\| \tag{9}$$

Preuve
$\begin{pmatrix} p \\ 1 \end{pmatrix} = M_1 \begin{pmatrix} m \\ 1 \end{pmatrix} = \begin{pmatrix} R_1 m + T_1 \\ 1 \end{pmatrix} \Rightarrow p = R_1 m + T_1$. Par l'inégalité triangulaire et sachant que $\|R_1\| = 1$, on obtient la propriété $\|p\| \leq \|m\| + \|T_1\|$.

Formule pour la dérivée d'un vecteur unitaire

Dérivée d'un vecteur unitaire

Soit deux vecteurs $\mathbf{u}$ et $\mathbf{v}$ tels que $\mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$. Alors :

$$\dot{\mathbf{u}} = \frac{1}{\|\mathbf{v}\|}(I - \mathbf{u}\mathbf{u}^T)\dot{\mathbf{v}} \tag{10}$$

Preuve

Considérons $\mathbf{u}$ et $\mathbf{v}$ tels que $\mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$, et notons $l = \|\mathbf{v}\|$. Alors :

$$l^2 = \mathbf{v}^T\mathbf{v} \tag{11}$$

En différenciant, on obtient :

$$2l\dot{l} = 2\mathbf{v}^T\dot{\mathbf{v}} \implies \dot{l} = \frac{\mathbf{v}^T\dot{\mathbf{v}}}{l} \tag{12}$$

Différencions maintenant $\mathbf{u} = \frac{\mathbf{v}}{l}$:

$$\dot{\mathbf{u}} = \frac{l\dot{\mathbf{v}} - \mathbf{v}\dot{l}}{l^2} = \frac{1}{l}(\dot{\mathbf{v}} - \frac{\mathbf{v}}{l}\frac{\mathbf{v}^T\dot{\mathbf{v}}}{l}) \tag{13}$$

Notons que $\mathbf{u}\mathbf{u}^T = \frac{\mathbf{v}\mathbf{v}^T}{l^2}$. En utilisant cette expression dans (13):

$$\dot{\mathbf{u}} = \frac{1}{l}(\dot{\mathbf{v}} - \mathbf{u}\mathbf{u}^T\dot{\mathbf{v}}) \tag{14}$$

Factorisons $\dot{\mathbf{v}}$ and on obtient le résultat :

$$\dot{\mathbf{u}} = \frac{1}{l}(I - \mathbf{u}\mathbf{u}^T)\dot{\mathbf{v}} \tag{15}$$

Résumé

Les robots parallèles à câbles présentent l'intérêt d'un grand espace de travail comparativement à des robots manipulateurs classiques. En contrepartie, la présence des câbles induit des collisions possibles supplémentaires, et des contraintes de positivité sur les tensions des câbles. La plateforme logicielle HPP développée dans l'équipe GEPETTO du LAAS permet de planifier des mouvements et des tâches de manipulation pour des systèmes articulés dans des environnements encombrés par des obstacles.

Le travail réalisé au cours de cette thèse a permis d'appliquer les algorithmes existant dans la plateforme HPP aux robots parallèles à câbles. Le travail de recherche de cette thèse apporte des solutions au problème de validation continue de trajectoires pour les robots parallèles à câbles. Dans un premier temps, un algorithme de validation continue de collision est étendu pour prendre en compte toutes les paires de collisions contenant des câbles. Cet algorithme se base sur le calcul d'un majorant de la vitesse relative de deux corps afin de valider successivement des portions du chemin. Dans un deuxième temps, ce même algorithme de validation continue est étendu afin de valider les conditions de tensions des câbles.

Ensemble, la validation continue de collision et la validation continue de tensions forment une méthode de validation continue globale pour robots parallèles à câbles. Cette méthode est utilisée au sein d'algorithmes de planification probabilistes pour effectuer de la planification de mouvement pour robots à câbles. Des scénarios représentant des tâches de logistiques ou des tâches industrielles sont testés en simulation ainsi qu'en réel sur le robot CoGiRo.

Abstract

Cable-driven parallel robots have large workspaces compared to standard manipulator robots. In return, the presence of cables creates additionnal possible collisions, as well as positivity contraints on the cables tensions. The software platform HPP was developped by the GEPETTO team from the LAAS laboratory for motion planning and manipulation planning for articulated systems in cluttered environement.

This thesis extends the algorithms within HPP to cable-driven parallel robots. The present research work provides solutions to the continuous trajectory validation for cable-driven parallel robots. First, a continuous collision detection algorithm is extended to take into account all the collisions pairs which include cables. This algorithm is based on the computation of an upper bound on the relative velocity between two bodies in order to succesively validate intervals of the trajectory. Secondly, this same continuous validation algorithm is extended to handle the validation of the cable tensions.

Together, the continuous collision detection and the continous tension validation form a global continuous validation method for cable-driven parallel robots. This method is used within probabilistic planning algorithm to plan manipulation movements for cable robots. Several scenarios representing industrial tasks are tested with the robot CoGiRo both in simulation as well as on the real robot.