Calculs d'espaces de travail à l'aide d'algorithmes de détection de collision : comparaison de deux méthodes

Stéphane Brunet et Luc Baron
Département de Génie Mécanique
École Polytechnique de Montréal
C.P. 6079, succ. C.V., Montréal (QC), H3C 3A7
stephane.brunet@polymtl.ca, luc.baron@polymtl.ca

Résumé

Cet article présente l'utilisation de deux méthodes de détection de collision dans le calcul d'espaces de travail de manipulateurs parallèles. La première utilise des polyèdres convexes pour décrire précisément la géométrie du manipulateur tandis que la seconde fait appel à un modèle filaire qui représente de manière approchée les formes des membrures. Un manipulateur parallèle sphérique 3-RRR à trois degrés de liberté est étudié comme exemple. Les résultats du calcul de l'espace de travail en ignorant puis en tenant compte des collisions entre les membrures sont ensuite discutés.

Abstract

This paper presents the integration of two collision detection methods into the workspace computation of parallel manipulators. The first one uses convex polyhedra in order to describe precisely the manipulator geometry whereas the second consists in a wireframe model. As an example, a three-degree-of-freedom 3-RRR spherical parallel manipulator is studied. Results of the workspace computation while ignoring and considering collisions between legs are then discussed.

1 Introduction

Lors de l'étude de manipulateurs parallèles, la taille et la forme de l'espace de travail Cartésien sont des variables importantes. Le calcul analytique de l'espace de travail d'un manipulateur sériel est une tâche aisée mais il en est tout le contraire dans le cas des manipulateurs parallèles.

Bien que fournissant une solution approchée, une méthode numérique peut avantageusement remplacer un calcul analytique pour les robots parallèles. Cet article présente une autre utilisation des 2^k-arbres dans le calcul des espaces de travail. Lorsque la discrétisation de l'espace est suffisamment fine, il est possible d'obtenir des résultats tout à fait convaincants.

Dans le processus de création et de sélection des manipulateurs parallèles, le concepteur doit faire face à deux problèmes majeurs caractéristiques de cette famille de robots : la petite dimension de l'espace de travail et les nombreuses possibilités d'obstruction entre les membrures. De nombreux articles traitant du calcul des espaces de travail s'arrêtent souvent à une résolution strictement mathématique basée sur le modèle géométrique inverse, sans tenir compte des obstructions [3, 4].

Cet article présente deux techniques de détection de collision pour le calcul des espaces de travail réels. La première fait appel à des modèles polyédriques convexes tandis que la deuxième est basée sur un simple modèle filaire du manipulateur.

2 Décomposition spatiale à l'aide de 2^k-arbres

Les *quadtree* et *octree* sont deux structures de données en arbre connues du domaine de la conception assistée par ordinateur. Celles-ci peuvent entre autre servir de modèle surfacique ou volumique approché ou bien de méthode de génération de maillage. En robotique, elles permettent de calculer numériquement l'espace de travail des manipulateurs lorsque le modèle mathématique est trop compliqué à résoudre analytiquement [1, 3, 4].

Ces structures de données font partie de la famille des 2^k-arbres dont la caractéristique principale est que chaque nœud de l'arbre peut avoir jusqu'à 2^k nœuds enfants. La constante k correspond à la dimension de l'espace utilisée. Ainsi, les *quadtree* et *octree* correspondent respectivement à $k = 2$ et $k = 3$. Traditionnellement, les nœuds des 2^k-arbres ont aussi une couleur qui leur est associée : *blanc* lorsqu'ils n'appartiennent pas à la portion de l'espace décrite, *noir* lorsqu'ils sont totalement inclus et *gris* lorsqu'ils sont partiellement inclus.

La figure 1 est un exemple de *quadtree* de profondeur 2 représentant la lettre L. La structure de donnée correspondante est présentée sur la figure 2. Le lecteur devrait noter que les branches de l'arbre ne vont pas nécessairement à une profondeur maximale. En effet, les quatre nœuds enfants blancs sous le nœud parent, blanc lui aussi, ont été fusionnées afin d'alléger la structure. Ainsi il ne reste que les noeuds essentiels.

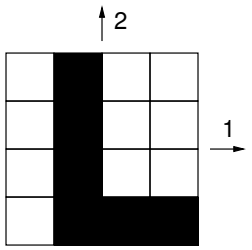

FIG. 1 – *Lettre 'L' représentée à l'aide d'un quadtree*

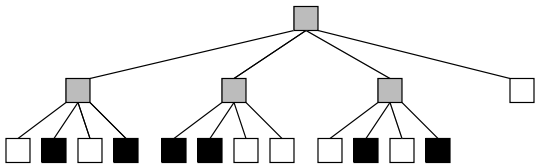

FIG. 2 – *Représentation de l'arborescence pour l'exemple de la figure 1*

Il existe plusieurs méthodes pour représenter les 2^k-arbres en C++. La plus directe serait de créer un objet *nœud* ayant un pointeur sur le nœud parent et 2^k pointeurs sur les nœuds enfants. Toutefois, une représentation explicite de l'arbre consomme énormément de mémoire.

Une autre méthode consiste à ne stocker que les nœuds-feuilles, c'est-à-dire les nœuds n'ayant

aucun enfant. Afin de conserver le lien entre les noeuds et de connaître quelle portion de l'espace ils représentent, une méthode pour les référencer est nécessaire. La technique utilisée dans le mémoire de Chablat [1] et reprise ici semble provenir d'un rapport de Morton [2]. Celui-ci décrit un ordre de balayage logique et ordonné des nœuds qui permet de définir un *chemin* unique pour chaque nœud. Pour chaque direction, une puissance de deux est assignée. Par exemple, pour un *quadtree*, 2^0 et 2^1 correspondent respectivement aux directions 1 et 2, comme présenté sur la figure 3. Ainsi, pour une profondeur donnée, il est possible d'associer un *chemin* vers chacun des nœuds en concaténant les puissances de deux.

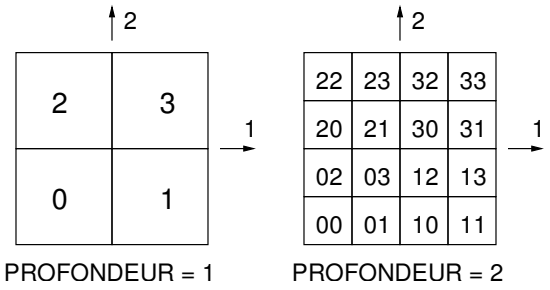

FIG. 3 – *Numérotation des nœuds dans un quadtree*

Comme chaque nœud est identifié et positionné dans l'espace par un chemin unique, il est possible de ne stocker que les nœuds feuilles dans un conteneur garantissant aucune duplication des données. Ainsi, pour l'exemple de la figure 1, il serait possible de ne stocker que les nœuds noirs dans un conteneur `set` de la librarie standard du C++. Ainsi, beaucoup d'espace mémoire est économisé et des profondeurs d'arbre plus grandes peuvent être utilisées.

En utilisant un conteneur associatif tel que `map` de la librairie standard du C++, il est aussi possible d'associer des données aux nœuds stockés. Par exemple, dans le cas d'une application en robotique, il pourrait être possible de conserver des données locales telles que le nombre de solutions ou le degré d'isotropie à la position indiquée.

3 Détection de collision

Beaucoup de techniques de détection de collision ont été développées ces dernières années. Un des objectifs premiers des recherches effectuées dans ce domaine est d'obtenir des algorithmes peu gourmands en ressources matérielles, tant au niveau de la mémoire que du temps de calcul. La solution idéale serait d'utiliser directement les équations mathématiques des courbes et surfaces utilisées dans le modèle 3D dessiné sur un logiciel de CAO. Mais un calcul de distance sur ces éléments est un problème compliqué à résoudre et demandant un temps de calcul non négligeable, ce qui rend la technique inutilisable pour des applications temps-réel. Parmi toutes les techniques disponibles, deux ont été sélectionnées pour les raisons mentionnées dans les prochains paragraphes.

3.1 Modèle polyédrique

Tous les logiciels de CAO sont obligés d'utiliser des modèles basés sur des triangles ou des rectangles pour l'affichage. En effet, les cartes graphiques sont conçues pour travailler avec des triangles et des rectangles pour effectuer les rendus réalistes d'objets en 3D. Il existe plusieurs fichiers d'échange de données basés sur ces modèles facettisés qui approchent la forme réelle décrite par les équations mathématiques : le VRML, langage balisé pour la réalité virtuelle, et le STL, fichier d'échange pour la stéréolithographie.

Une technique de détection de collision utilisant de tels fichiers est donc tout à fait appropriée car n'importe quel logiciel de CAO est utilisable pour générer des modèles 3D se rapprochant beaucoup des formes réelles. C'est la raison principale qui a guidé le choix de cette première méthode.

Pour décrire un solide avec des facettes triangulaires ou rectangulaires, il est indispensable de savoir de quel côté de chaque triangle la matière se trouve. Il faut donc associer une orientation aux triangles et avoir un polyèdre complètement fermé. Des algorithmes rapides et robustes sont disponibles pour calculer la distance entre deux polyèdres convexes comme, par exemple, celui de

Lin-Canny [6]. Malheureusement, la plupart des modèles polyédriques créés sont non-convexes et une technique de conversion doit alors être utilisée pour mettre à profit ces algorithmes performants.

Dans le cadre de cet article, une technique développée à l'Université de Caroline du Nord et disponible sous la forme d'une librairie C++ (nommée SWIFT++) a été utilisée [5]. Elle est basée sur une décomposition du modèle polyédrique en surfaces convexes et sur une structure de données hiérarchique de polyèdres convexes.

La première étape consiste à décomposer le modèle polyédrique en plusieurs surfaces convexes. L'algorithme utilisé pour cette tâche consiste à parcourir un graphe contenant les sommets et les côtés de chaque triangle pour chercher des ensembles de triangles connectés et orientés de manière à former une surface convexe. Ensuite, la structure de données hiérarchique est construite en séparant les différentes surfaces de manière itérative et en créant à chaque étape un polyèdre convexe associé au sous-ensemble de surfaces en question. De manière pratique, cet algorithme mène à un ensemble de polyèdres convexes parmi lesquels celui correspondant à la racine de la structure englobe le modèle au complet de manière très imprécise. À mesure que la structure est parcourue, les polyèdres sont plus nombreux, plus petits et correspondent de mieux en mieux au modèle initial. Lors de la construction des polyèdres convexes, un attribut est associé à chacune des facettes. Trois états sont possibles : la facette appartient au modèle initial, la facette a été générée et se trouve en dehors du modèle ou la facette a été générée et se trouve à l'intérieur du volume décrit par le modèle.

L'intérêt d'utiliser cette hiérarchie est d'accélérer le calcul de collision ou de distance. Il est en effet rapide de déterminer de façon itérative si deux objets s'intersectent en allant chercher au besoin une description plus détaillée de la géométrie à l'intérieur des structures hiérarchiques. Par exemple, si les deux polyèdres de la racine de l'arbre ne s'intersectent pas, il n'est pas nécessaire d'utiliser les polyèdres plus nombreux et contenant plus de facettes pour savoir que les deux objets ne s'intersectent pas. Par ailleurs, si les deux polyèdres s'intersectent et que les triangles impliqués

appartiennent au modèle initial ou sont inclus dans le volume, alors il n'est pas non plus nécessaire de chercher plus loin car les objets s'intersectent. Tant qu'il n'est pas possible d'en arriver à cette conclusion, c'est à dire lorsque les polyèdres s'intersectent avec une ou deux surfaces à l'extérieur du volume, il est nécessaire d'avancer dans les structures hiérarchiques pour tester les polyèdres convexes approchant plus précisément le modèle initial.

Ainsi, bien que la quantité de mémoire nécessaire pour stocker les structures hiérarchiques est importante, un gain appréciable de vitesse est réalisé car un nombre limité de calculs est nécessaire pour vérifier la collision entre deux objets. L'intérêt d'utiliser une technique rapide de détection de collision est important dans le cadre de calculs d'espaces de travail car beaucoup de positions différentes doivent être testées.

3.2 Modèle filaire

Lorsque les objets sont très élancés, comme par exemple des tiges droites ou courbées, un modèle hiérarchique peut rapidement devenir très volumineux. D'autres techniques permettent d'approcher plus grossièrement la géométrie pour améliorer l'usage des ressources matérielles tout en ne compromettant pas trop la précision du résultat. La plus simple consiste à effectuer un modèle filaire de l'objet comme si c'était un maillage d'éléments-poutre en éléments finis (voir fig. 4). À chaque segment, une épaisseur approchant les dimensions de l'objet est associée (fig. 5).

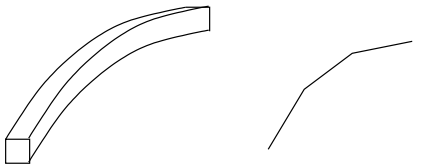

FIG. 4 – *Modèle réel et modèle filaire associé*

La distance entre deux objets correspond alors à calculer la distance entre les deux segments les plus rapprochées, qui est un calcul géométrique rapide. Lorsque la distance minimale est inférieure

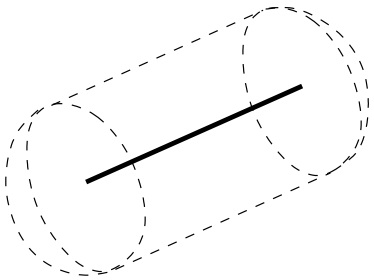

FIG. 5 – *Représentation de la matière entourant un segment du modèle filaire*

aux dimensions associées aux segments impliqués, il y a alors collision entre les deux objets. Cette technique a été développée car elle est extrèmement simple. De plus, il n'est plus nécessaire de créer la géométrie dans un logiciel de CAO grâce à la simplicité du modèle.

4 Une application en robotique

Pour illuster l'intrégration des 2^k-arbres avec la détection de collision, un manipulateur parallèle sphérique à trois degrés de liberté est présenté. La figure 6 présente la topologie de la famille des manipulateurs parallèles 3-RRR. Trois jambes constituées de deux membrures relient la base 0 à l'effecteur 3. Toutes les liaisons sont des rotoïdes et seulement celles attachées à la base sont actionnées. Ainsi, aucun actionneur n'est en mouvement lors des déplacements de l'effecteur. L'inertie est alors réduite et le manipulateur peut donc avoir une grande rapidité de déplacement.

Lorsque les axes des liaisons rotoïdes sont concourants en un point, des robots sphériques à trois degrés de liberté en rotation sont obtenus (voir fig. 7). Ceux-ci sont particulièrement intéressants dans des applications nécessitant des grandes vitesses de rotation et une grande précision. Par exemple, l'*œil agile* de Gosselin [9] est un prototype permettant l'orientation d'une caméra vidéo.

La figure 7 présente un manipulateur ayant beaucoup de symétries et des membrures distales et proximales de même dimension, c'est-à-dire $\beta = \gamma$ sur la figure 8. L'étude de l'espace de travail de ce manipulateur est basé sur le modèle géométrique inverse. Pour une même posture de l'effecteur,

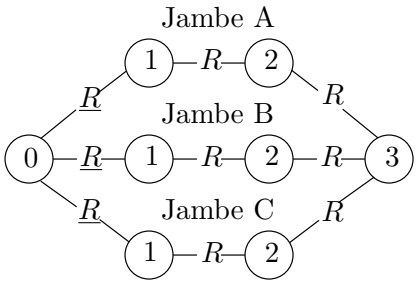

FIG. 6 – *Topologie des manipulateurs parallèles 3-RRR*

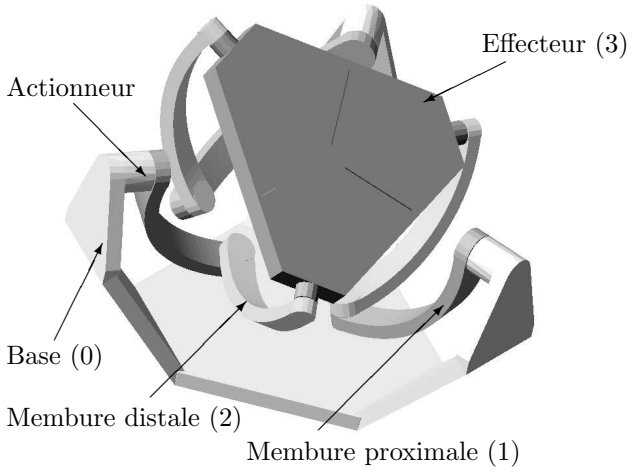

FIG. 7 – *Un manipulateur sphérique 3-RRR*

chaque jambe du manipulateur peut être dans la position *coude en haut* ou *coude en bas*, sauf dans les positions singulières, c'est-à-dire lorsqu'une jambe est complètement dépliée ou repliée. Ainsi, le modèle géométriques inverse peut avoir un maximum de huit solutions [7] d'où l'existance de huit modes d'opération (tableau 1). Le calcul de l'espace de travail est effectué dans l'espace des angles d'Euler Z-X-Z représentant l'orientation de l'effecteur. Bien que les angles d'Euler puissent causer des problèmes de singularités mathématiques, ces dernières sont évitées grâce à la structure particulière des 2^k-arbres. La portion de l'espace discrétisé s'étend de $-\pi$ à $+\pi$ dans chacune des directions. L'appartenance d'un nœud à l'arbre est déterminé par l'existence d'une ou plusieurs solutions du modèle géométrique inverse pour une position correspondant au centre de la portion cubique de l'espace pointée par le nœud.

Une comparaison visuelle entre les résultats obtenus à l'aide des deux méthodes de détection

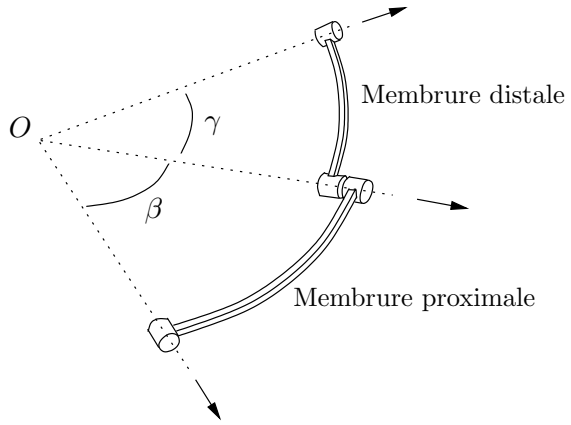

FIG. 8 – *Géometrie des jambes – membures proximale et distale*

de collision et obtenus sans tenir compte des collisions, est maintenant présentée. Le manipulateur choisi est totalement symétrique avec $\beta = \gamma = 65°$ pour toutes ses membrures. Afin de faciliter la comparaison, seules les solutions du premier mode d'opération sont considérées. La figure 9 correspond à la forme de l'espace parcouru par le manipulateur dans ce mode, sans tenir compte des collisions tandis que les figures 10 et 11 montrent les résultats obtenus avec les deux méthodes de détection de collision présentées précédemment. Le cadre gris correspond aux dimensions maximales que l'espace de travail peut avoir. Chaque case représente une solution possible de modèle géométrique inverse. Visuellement, il ne semble pas avoir de différences majeures quant à la forme des espaces de travail calculés avec les deux méthodes. Par ailleurs, en combinant les solutions menant à des collisions avec celles sans collision, l'espace de travail de la figure 9 est retrouvé. Enfin, la quantité de solutions théoriques menant à des collisions est suffisamment importante pour être non négligeable.

Des résultats quantitatifs sont présentés dans le tableau 2. Les valeurs présentées correspondent au pourcentage de l'espace total (le cadre gris des figures 9 à 11) atteignable par le manipulateur, pour chaque mode d'opération. Le manipulateur étudié étant symétrique, les valeurs obtenues sont communues à plusieurs modes d'opération. Les deux méthodes de détection de collision mènent à des résultats quasiment identiques (2 % de différence maximum). Pour le manipulateur étudié,

l'erreur maximale commise en négligeant les collisions entre les membrures est d'environ 14 %. Elle est obtenue pour les modes 1 et 8, c'est-à-dire lorsque toutes les jambes sont dans la même posture (*coude haut* ou *coude bas*).

Le manipulateur symétrique avec $\beta = \gamma = 90°$ est maintenant considéré. En théorie, l'espace pourrait être entièrement décrit par ce manipulateur car, pour chaque jambe, l'extrémité reliée à l'effecteur est capable de se positionner de manière quelconque sur la sphère décrite par les deux premiers angles d'Euler (fig. 12). Ce serait donc théoriquement le manipulateur optimal. Le tableau 3 présente les résultats comparatifs des différentes méthodes présentées. De la même manière que précédemment, les deux méthodes de détection de collision donnent des résultats quasi identiques. Par ailleurs, l'erreur maximale obtenue en négligeant les collisions dépasse 30 % ! Enfin, ces résultats prouvent qu'un simple modèle filaire permet de faire une bonne approximation de l'espace de travail réel en tenant compte des obstructions. De plus, dans le cas du manipulateur sphérique, la nécessité d'utiliser la détection de collision en vue de maximiser l'espace de travail réel est évidente. En effet, plus les membrures sont longues, plus l'espace de travail est grand, mais plus les chances de collision sont élevées.

5 Conclusion

Le calcul d'espaces de travail de manipulateurs parallèles doit se faire en tenant compte des obstructions. Dans le cas du manipulateur sphérique, ignorer les collisions conduit à une erreur de plus de 30%, ce qui est loin d'être négligeable.

Utiliser un modèle polyédrique permet de calculer précisément l'espace de travail réel du manipulateur en générant une géométrie qui se rapproche grandement de la forme prévue pour le manipulateur. Toutefois, il est indispensable d'utiliser un logiciel de CAO pour générer un modèle facettisé qui sera ensuite converti en une hiérarchie de polyèdres. Beaucoup d'étapes intermédiaires

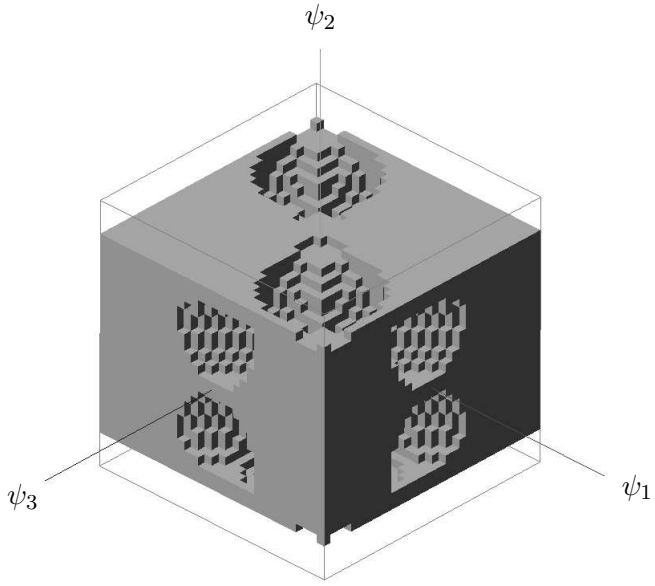

FIG. 9 – *Espace de travail pour le premier mode d'opération sans tenir compte des collisions*

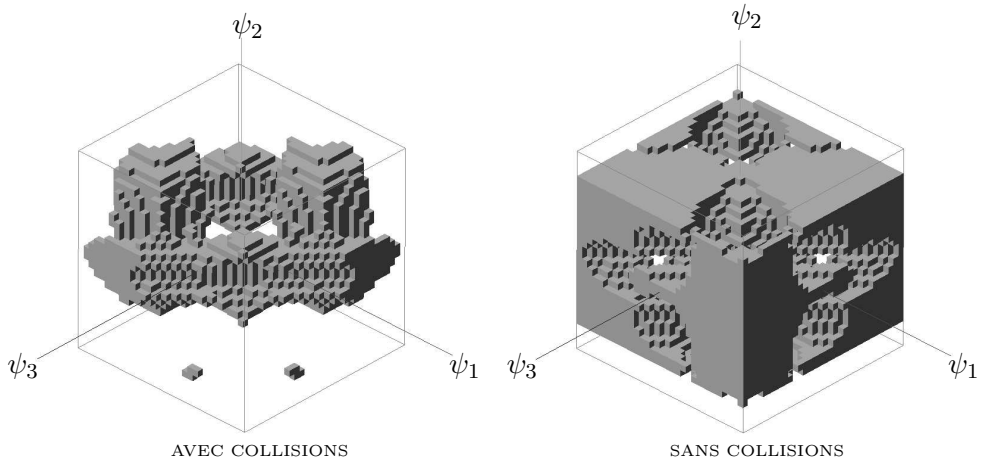

FIG. 10 – *Modèle polyédrique – solutions du premier mode d'opération*

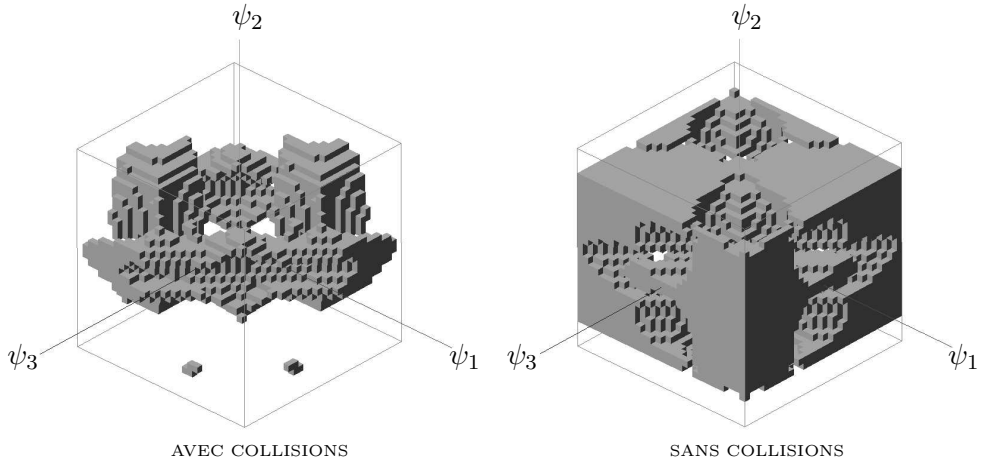

FIG. 11 – *Modèle filaire – solutions du premier mode d'opération*

Mode	Jambe A	Jambe B	Jambe C
1	Coude bas	Coude bas	Coude bas
2	Coude haut	Coude bas	Coude bas
3	Coude bas	Coude haut	Coude bas
4	Coude haut	Coude haut	Coude bas
5	Coude bas	Coude bas	Coude haut
6	Coude haut	Coude bas	Coude haut
7	Coude bas	Coude haut	Coude haut
8	Coude haut	Coude haut	Coude haut

TAB. 1 – *Les huit modes d'opération*

Mode d'opération	Sans détection de collision	Modèle polyédrique	Modèle filaire
1 et 8	56,1 %	40,2 %	42,2 %
2 et 7	56,1 %	47,5 %	48,6 %
3, 4, 5 et 6	56,1 %	51,0 %	51,6 %

TAB. 2 – *Pourcentage de l'espace total – Résultats comparatifs pour* $\beta = \gamma = 65°$

Mode d'opération	Sans détection de collision	Modèle polyédrique	Modèle filaire
1 et 8	100 %	52,9 %	57,1 %
2 et 7	100 %	70,5 %	73,4 %
3, 4, 5 et 6	100 %	71,2 %	73,7 %

TAB. 3 – *Pourcentage de l'espace total – Résultats comparatifs pour* $\beta = \gamma = 90°$

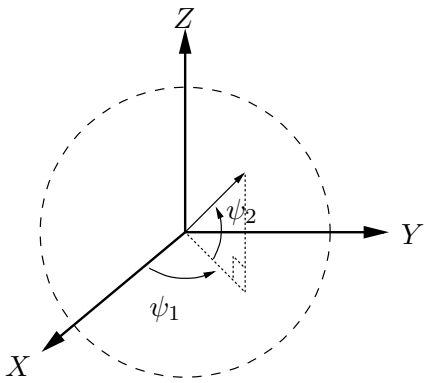

FIG. 12 – *La sphère décrite par les deux premiers angles d'Euler*

sont donc nécessaires mais le résultat est probant.

Lorsque la précision de calcul n'est pas fondamentale, un modèle filiaire peut convenir tout à fait. Par exemple, dans les phases préliminaires du design d'un manipulateur, la géométrie n'est pas connue de manière exacte et change souvent. Générer un modèle filaire n'est pas compliqué et ne nécessite pas de logiciel particulier. Par ailleurs, dans l'exemple présenté, l'erreur occasionné par l'utilisation de ce modèle est très faible. C'est donc une méthode qui peut avantageusement remplacer la précédente dans les phases préliminaires de conception.

Remerciements

Merci au groupe de recherche *Gamma* de l'Université de Caroline du Nord pour avoir mis à disposition gratuitement la librairie de détection de collision SWIFT++ (url : `www.cs.unc.edu/~geom/SWIFT++/`).

Les auteurs remercient par ailleurs le CRSNG (subvention RGPIN-203618) et le FCAR (subvention NC-66861) pour leur aide financière.

Références

[1] Chablat, D., 1988, "Domaines d'unicité et parcourabilité pour les manipulateurs pleinement parallèles", Ph.D. Thesis, École Centrale de Nantes.

[2] Morton, G.M., 1966, "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing", Technical Report, IBM Ltd, Ottawa, Canada.

[3] Baron, L., 2001, "Workspace-Based Design of Parallel Manipulators of Star Topology with a Genetic Algorithm", *Proceedings of ASME Design Engineering Technical Conferences*, DETC/DAC-21026, Pittsburgh, USA.

[4] Wenger, P., Chablat, D., 2000, "Kinematic Analysis of a New Parallel Machine Tool : the Orthoglide", *Advances in Robot Kinematics*, J. Lenarčič and M.M. Stanišić (Eds.), Kluwer Academic Publishers, pp. 305-314 .

[5] Ehmann, S.A., Lin, M.C., 2001, "Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition", EUROGRAPHICS 2001, Volume 20, number 3.

[6] Lin, M.C., Canny, J.F, 1991, "A fast algorithm for incremental distance computation", *Proceedings of IEEE Int. Conf. on Robotics and Automation* 1991.

[7] Gosselin, C., and Angeles, J., 1989, *The Optimum Kinematic Design of a Spherical Three-Degree-of-Freedom Parallel Manipulator*, ASME Journal of Mechanical Design, Vol. 111, No. 2, pp. 202-207.

[8] Gosselin, C., and Lavoie, E., 1993, *On the development of Spherical 3-DOF Parallel Manipulators*, Int. Journal of Robotic Research, Vol. 12, No. 4, pp. 394-402.

[9] Gosselin, C., St-Pierre, E. and Gagné, M., 1996, *On the development of the agile eye: mechanical design, control issues and experimentation*, IEEE Robotics and Automation Society Magazine, Vol. 3, No. 4, pp. 29-37.