UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

Pipette automatisée et pince comme effecteurs d'un bras robotique collaboratif pour l'automatisation de procédés expérimentaux en biologie synthétique

Mémoire de maitrise
Spécialité : génie électrique

David Pivin

Sherbrooke (Québec) Canada

Septembre 2021

MEMBRES DU JURY

François Michaud

Directeur

Sébastien Rodrigue

Codirecteur

François Ferland

Évaluateur

Pierre-Étienne Jacques

Évaluateur

RÉSUMÉ

Les plateformes robotiques sont de plus en plus abordables et accessibles. Le laboratoire de biologie synthétique de l'Université de Sherbrooke a recours à de tels robots pour pipetter divers liquides afin d'augmenter la productivité et l'efficacité lors de l'exécution de protocoles expérimentaux simples. Par contre, pour des protocoles complexes, une intervention humaine est requise pratiquement à chacune des étapes. Par exemple, certains protocoles nécessitent plusieurs étapes successives de quelques heures chacune, mais qui au final s'échelonnent sur plusieurs jours. Les nombreux instruments de laboratoire impliqués et le besoin d'une attention constante de la part de l'expérimentateur rendent ce type d'expériences particulièrement pénibles malgré un intérêt scientifique grandissant envers de telles approches.

Ce projet de recherche vise à développer une plateforme robotique capable d'automatiser un protocole biologique complexe sans intervention humaine. Pour y arriver, l'approche consiste à concevoir une pipette automatisée et à manipuler des instruments couramment retrouvés dans les laboratoires de biologie à l'aide d'une pince robotisée. Une caractérisation du matériel et des appareils doit être faite *a priori*, définissant une structure de données ayant des zones ou éléments d'intérêt utilisés pour la planification des mouvements et des manipulation. De plus, les instruments de laboratoire sont associés à des actions spécifiques qui régissent les interactions entre éléments. Une interface graphique est disponible pour faciliter l'enregistrement de ces données. Ainsi, l'opérateur peut créer, ajouter ou exécuter un protocole complexe, avec les objets déjà caractérisés, à partir de l'interface graphique.

La plateforme robotique est utilisée pour effectuer un protocole simple de dilution en série d'échantillons liquides à l'aide de la pipette automatisée. Un protocole plus complexe permettant l'introduction d'ADN recombinant chez des bactéries par électroporation est également réalisé. Cette dernière manipulation fait appel à du matériel et des appareils spécialisés qui ne peuvent pas facilement être intégrés dans les plateformes robotiques conventionnelles.

La plateforme robotique démontrée dans le cadre de ce projet pourrait ainsi devenir un outil d'intérêt en biologie afin de d'exécuter diverses manipulations qui sont présentement jugées trop complexes ou laborieuses pour être automatisées.

Mots-clés : Planification de mouvement, Manipulation d'objets, Biologie synthétique, Automatisation de protocoles en biologie, Robotique collaborative

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE 1

INTRODUCTION

Le coût des plateformes robotiques est en baisse depuis plus de deux décennies [1]. De tels automates augmentent l'efficacité et la productivité d'une tâche à réaliser dans une multitude de domaines dont la biologie synthétique, domaine scientifique et biotechnologique émergeant qui combine biologie et principes d'ingénierie.

L'environnement d'un chercheur en biologie synthétique est généralement un laboratoire avec une température, une humidité et un éclairage contrôlé. Cet espace contient divers instruments, des produits utilisés dans les expériences et des personnes qui accomplissent leurs travaux à leur paillasse ou à des stations dédiées à certains types de manipulations. Le Laboratoire de biologie synthétique de l'Université de Sherbrooke a recours à des automates robotisés pour pipetter divers liquides, c'est-à-dire prélever un liquide à l'aide d'une pipette. Cette automatisation permet aux chercheurs de se concentrer sur l'analyse des résultats plutôt que d'effectuer des manipulations longues, répétitives et qui nécessitent une bonne précision. Cela est vrai pour un protocole simple, mais pour exécuter un protocole plus complexe une intervention humaine est requise pratiquement à chaque étape du protocole.

Plus spécifiquement, un protocole complexe nécessite plusieurs instruments : pipettes, centrifugeuse, thermocycleur, spectrophotomètre, appareil à électrophorèse, unités de séparation magnétique, etc. L'instrument le plus utilisé est sans aucun doute la pipette. Elle prélève précisément des liquides et les déplace vers divers types de contenants. Aussi, la pipette peut servir à prélever des colonies de bactéries sur une gélose d'agar, une substance nutritive favorisant ou inhibant la prolifération et le développement de certaines bactéries. De plus, elle sert à transporter les liquides vers d'autres instruments ou récipients. Les autres instruments couramment employés en laboratoire permettent d'incuber les échantillons à des températures précises, de quantifier le nombre de cellules ou la concentration de certaines macromolécules, de séparer efficacement divers types de cellules ou de molécules, etc. Hormis la pipette et les divers contenants et tubes, la majorité des instruments sont statiques, c'est-à-dire qu'ils n'ont pas besoin d'être déplacés lors d'un protocole. Il y a plusieurs réceptacles et récipients pour contenir et transporter les liquides. Certains instruments doivent interagir avec les réceptacles. Par exemple, la pipette nécessite un embout en plastique à usage unique, lors de l'aspiration et de la dispense de liquide, pour

éviter la contamination. Elle requiert aussi divers types de tubes ou bouteilles, qui varient en capacité de quelques microlitres jusqu'à plusieurs litres. Le format de ces contenants peut être individuel, en groupe de 8 ou 12, ou encore en plaque de 96 ou 384 puits.

Il existe une multitude de protocoles biologiques couramment utilisés dans les laboratoires de biologie synthétique. La presque totalité des protocoles peuvent être décomposés en une série d'étapes plus simples qui vont du pipettage vers des incubations à des températures spécifiques, en passant par diverses étapes de manutention. L'agencement de ces étapes permet d'effectuer des tâches de complexités variables. Par exemple, certains protocoles d'ingénierie génétique nécessitent plusieurs étapes successives de quelques heures chacune, mais qui au final s'échelonnent sur plusieurs jours [2, 3]. Malgré un intérêt scientifique grandissant envers ce type d'approche, les nombreux instruments de laboratoire impliqués et le besoin d'une attention constante de la part de l'expérimentateur rendent ce type d'expérience particulièrement pénible [3]. Les manipulations clées pour l'automatisation d'un grand nombre de protocoles sont le pipettage de volume variant entre 1 microlitre (µL) et 300 µL. Ces étapes doivent pouvoir s'accomplir avec précision. Les liquides prélevés doivent pouvoir être déplacés d'un type de contenant vers un autre, soit de tubes individuels d'un volume allant de 50 µL à 50 mL ou de plaques comprenant des puits de 200 µL à 2 mL. Finalement, les liquides ou les contenants doivent pouvoir être déplacés d'une station à l'autre afin de les centrifuger, de les incuber à des températures spécifiques, de mesurer une densité optique, de les électroporer ou des les étaler sur des géloses.

Figure 1.1 Eppendorf epMotion 5075 [4].

La majorité des appareils dans les laboratoires sont destinés à l'utilisation par des humains. Certains appareils, comme la plateforme robotique Eppendorf epMotion 5075 montrée à

la figure 1.1, permettent une automatisation utilisant une pipette pour le transfert et la mesure de liquides et des accessoires spécialisés. En général, l'automatisation se fait pour un faible nombre d'instruments. Par exemple, la figure 1.2 montre le robot Andrew+, de la compagnie Andrew Alliance, qui manipule directement les pipettes manuelles. Ces actions sont limitées à l'aspiration et la dispense de liquide. Cette capacité est similaire pour l'automate d'Eppendorf mais une pipette spécialisée développée par cette compagnie est utilisée plutôt qu'une pipette manuelle couramment employé par les expérimentateurs. La compagnie Opentrons offre une plateforme robotique OT2, montrée à la figure 1.3, qui pipette des liquides, les réchauffe et qui peut séparer magnétiquement les particules. Un module de thermocycleur est aussi disponible, ce qui permet de faire cinq actions. Les modules sont conçus pour fonctionner avec la plateforme robotique. Les modules et les réceptacles sont fixés sur la table de travail. Le coût de cet appareil est d'environ 10 000$. Les plateformes comme le Microlab NIMBUS de Hamilton et le Cavro Omni Flex de Tecan offrent aussi plusieurs actions. Pour ceux-ci, il faut utiliser leurs propres instruments et réceptacles, et ces appareils se vendent dans les centaines de milliers de dollars.

Figure 1.2 Andrew Alliance 1000g [5].

Habituellement, le déplacement des échantillons entre différents modules ou appareils s'effectue par un expérimentateur plutôt que par un robot. Par exemple, un protocole qui requiert une centrifugeuse ou un module d'électroporation devra déplacer les échantillons de façon manuelle, car elles ne sont pas intégrées aux solutions actuellement disponibles sur le marché ou atteignent des prix prohibitifs dépassant facilement plusieurs dizaines de milliers de dollars. Le présent projet de recherche tente de déterminer une façon de réaliser un protocole complexe avec un système robotisé sans aucune intervention humaine. L'inté-

Figure 1.3 (Gauche) Robot Opentrons OT2 [6]. (Droite) Module TempDeck et MagDeck [7].

gration avec des plateformes robotisées sur le marché n'est pas envisagée car celles-ci sont propriétaires (code source fermé) et spécifiques à une seule ou quelques tâches. De plus, leurs coûts sont prohibitifs et chaque automate est dans un environnement clos, ce qui rend l'intégration plus difficile. D'autres avenues tentent de recréer ou de modifier chaque appareil pour qu'ils soient robotisés. Toutefois, cela demande beaucoup de temps, d'argent et requiert une expertise pour certains appareils. Les options possibles sont :

- L'achat d'instruments automatisés complexes est très coûteux. L'intégration est généralement simple, il suffit de respecter les branchements et d'intégrer le protocole de communication au système. Parfois, ces appareils sont destinés à l'utilisation sur une plateforme robotique spécifique et sont donc inutilisables à moins de consacrer du temps à comprendre le protocole pour l'utiliser. L'envoi d'une commande à un objet automatisé est presque instantanée, ce qui réduit le temps d'exécution pour un protocole.
- La création d'un appareil automatisé est complexe et coûteuse. De plus, il faut avoir une expertise dans le domaine et entreprendre des démarches pour certifier le produit. Aussi, il y a des chances que l'appareil ne soit pas aussi performant qu'un appareil déjà existant. Par contre, l'intégration au système robotique se fera rapidement, car il a été conçu spécifiquement pour celui-ci.

– La modification d'un appareil existant peut être simple ou complexe. Le but de cette modification est de simplifier l'intégration à la plateforme. Cette solution intrusive peut rendre la certification obsolète et annuler la garantie de l'appareil. De plus, cela peut rendre l'appareil non fonctionnel pour certaines utilisations.
– La manipulation directe d'un appareil destiné à l'humain offre certains avantages : coût réduit versus le même appareil automatisé, certification valide et non intrusive (selon les techniques utilisées pour la manipulation). L'appareil reste donc utilisable par d'autres scientifiques. Par contre, l'intégration est plus difficile, car l'automate doit imiter le bras d'un humain pour le manipuler. Aussi, un appareil complexe peut nécessiter un système de vision pour la reconnaissance et la localisation d'items, ce qui augmente les ressources en termes de conception et de coût. C'est l'adaptation qui requiert le plus de développements d'un point de vue logiciel. Enfin, manipuler les appareils au lieu de les commander électroniquement ajoute un temps supplémentaire non négligeable à la planification et à l'exécution du robot.

Comme solution, la direction prise par ce projet de recherche consiste à explorer la possibilité qu'un bras robotique puisse manipuler des instruments types en laboratoire conçus pour être manipulés par des humains. Le chercheur devra placer les instruments nécessaires à son protocole biologique sur la station de travail. Par la suite, il devra entrer le protocole dans une interface graphique ou choisir un protocole déjà présent. Enfin, la plateforme robotique exécutera le protocole complet sans intervention humaine.

Ce document est formé des sections suivantes. Premièrement, une brève description des principaux requis d'une plateforme robotique pour une telle application est présentée et comprend la planification de mouvement et de trajectoire, le contrôle de mouvement ainsi que le contrôle en force. Ensuite, la méthodologie pour accomplir un protocole biologique sans intervention humaine est décrite. Finalement, des analyses et résultats sont présentés pour la réalisation de deux protocoles spécifiques avec le système robotique développé pendant ce projet de maitrise.

CHAPITRE 2

CONCEPTS ASSOCIÉS AU CONTRÔLE D'UN BRAS ROBOTIQUE COLLABORATIF

Il y a plusieurs types de robots sur le marché. Un robot est défini par son application, sa locomotion, son environnement et sa cinématique. Le tableau 2.1 présente les avantages et inconvénients des principaux types de robots industriels stationnaires sur le marché [8], et le tableau 2.2 présente les deux principaux types de robots manipulateurs [9, 10]. Les robots industriels effectuent des tâches telles la soudure, le déplacement de matériel, l'empaquetage d'objets, etc. La plupart du temps, ces plateformes n'effectuent qu'une seule tâche. Par contre, elles le font de manière précise, rapide et robuste. Le robot industriel a besoin d'une cage de sécurité pour s'assurer de ne pas entrer en contact avec un humain, ce qui pourrait causer des blessures importantes. Aussi, un protocole de sécurité doit être mis en place afin de minimiser les risques d'accidents. Les robots collaboratifs, de type articulé, ont la cote depuis quelques années [11]. Ces robots peuvent travailler de manière sécuritaire en collaboration avec un humain. Cela ouvre la possibilité à de nouvelles techniques comme l'apprentissage de mouvements par démonstration. Dans la majorité des cas, ils n'ont pas besoin d'une cage de sécurité. Toutefois, ils sont limités en portée, charge utile, vitesse et répétabilité vis-à-vis les robots industriels.

Les prochaines sous-sections expliquent brièvement les concepts associés au contrôle d'un bras manipulateur, dont l'analyse du mouvement et de la force, la planification de mouvement, la planification de trajectoires, le contrôle du mouvement, le contrôle en force et les architectures de contrôle en robotique.

2.1 Analyse de mouvement et de force

Avant d'aborder les concepts liés à la planification et au contrôle robotique, une introduction aux concepts de mouvement et de force est requise. Cette analyse est divisée en trois catégories, soit la cinématique, la cinématique différentielle et statique, ainsi que l'étude de la dynamique. Les concepts tels l'espace de configuration, l'espace cartésien, l'espace des phases, la matrice jacobienne et les singularités y sont traités.

Tableau 2.1 Synthèse des principaux types de robots stationnaires.

Type	**Avantages**	**Inconvénients**
Cartésien	Grande charge utile Grande vitesse Mouvement simple	Structure encombrante Orientation limité Environnement limité
Scara	Grande vitesse Mouvement simple Structure non encombrante Redéploiement rapide	Petite charge utile Orientation limité
Articulé	Structure non encombrante Pose flexible Option collaborative Redéploiement rapide	Petite charge utile Mouvement complexe Difficilement modifiable
Delta	Très grande vitesse Répétabilité Mouvement simple	Structure encombrante Orientation limité Maintenance mécanique élevé

Tableau 2.2 Types de robots manipulateurs.

Application	**Avantages**	**Inconvénients**
Industriel	Grande précision Grande répétabilité Grande charge utile Grande vitesse	Zone de sécurité (cage) Protocole de sécurité Écosystème fermé Prix élevé
Collaboratif	Prix faible Techniques d'apprentissage Travail collaboratif	Vitesse limitée Charge utile limitée Répétabilité limitée Précision limitée

2.1.1 Cinématique

La cinématique est l'étude de mouvement indépendamment des forces qui causent le déplacement. Elle s'intéresse plus spécifiquement à la géométrie d'une structure robotique, c'est-à-dire une chaîne de corps rigides connectés par des joints révolutifs et/ou prismatiques [1]. La cinématique directe du robot donne la position de l'effecteur [2] par rapport à un référentiel [3] (p.ex. la base du robot) en fonction des angles des joints [8, 13]. Quant à elle, la cinématique inverse calcule les angles des joints (domaine des joints) pour une pose dans un repère donné (domaine cartésien). Celle-ci peut retourner une solution, plusieurs solutions, une infinité de solutions ou aucune solution.

Espace de configuration

L'espace de configuration (*C-space*), aussi appelé espace des joints (*Joint-Space*), est un modèle d'espace d'états qui représente les différentes positions atteignables par le robot [14, 15, 12]. Cet espace est en fait un système de coordonnées avec une dimension par degré de liberté (DDL) du robot. Dans le *C-space*, la pose d'un robot est simplement un point tandis qu'un obstacle est une forme complexe [15]. Comme exemple, la figure 2.1 montre l'espace de travail versus l'espace de configuration pour un robot planaire avec 2 DDL et quelques obstacles. Un obstacle dans l'espace de configuration est nommé *C-obstacle* et la portion qui n'est pas en collision est appelée *C-free*. Les limites de joints sont traitées comme des obstacles.

Espace cartésien

L'espace cartésien, aussi appelé l'espace des tâches (*Task Space*) ou espace opérationnel (*Operational Space*), est un système de coordonnées qui possède six dimensions par rapport à un repère : soit x, y et z pour la position ainsi que les trois rotations (rx, ry, rz) par rapport au trois axes. Il est plus intuitif de naviguer dans cet espace que l'espace de configuration. Par contre, l'espace cartésien est sujet à des problèmes de singularité et de redondance [12].

Singularité

Une singularité se produit lorsque l'effecteur d'un robot n'est pas en mesure d'atteindre ou de traverser un certain point dans l'espace opérationnel. Il y a singularité lorsque le robot est commandé hors de la zone de travail. Près d'une singularité, un petit changement de vitesse au niveau de l'effecteur dans le domaine cartésien peut causer de grande vélocité dans le domaine des joints en dehors de la limite physique du robot [8]. Cette singularité

1. Il existe d'autres types de joints tels que les sphériques, cylindriques, universels et hélicoïdaux [12].
2. Dernier lien du robot ou outil attaché sur ce lien. Un repère est généralement associé à l'effecteur pour planifier un mouvement ou une interaction.
3. Repère cartésien orthogonal.

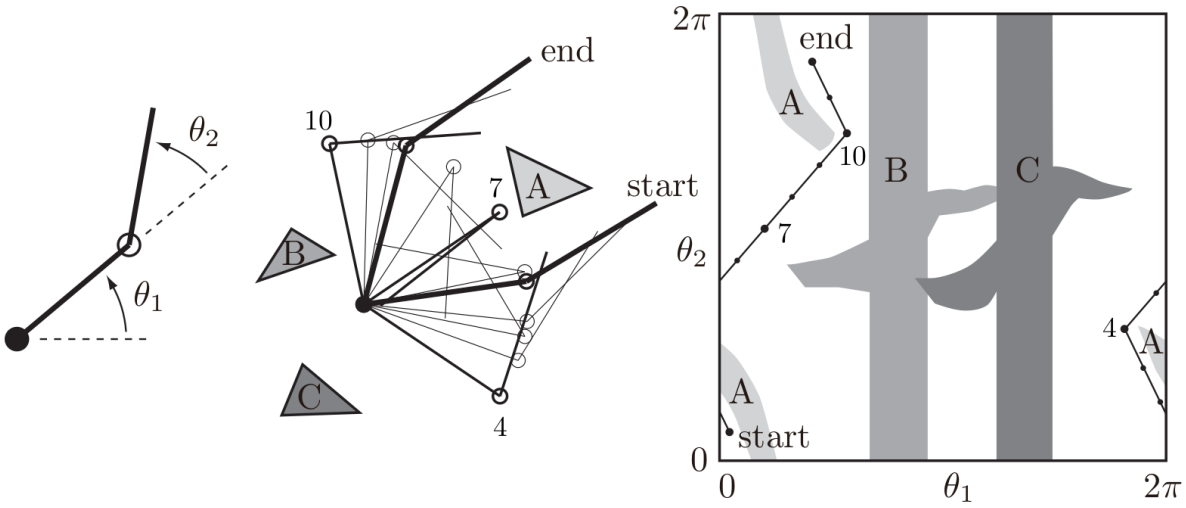


Figure 2.1 Comparaison entre l'espace cartésien et l'espace de configuration pour un robot planaire : (gauche) l'angle des joints d'un robot planaire ; (milieu) le bras naviguant avec les obstacles A, B, et C ; (droite) mouvement dans le *Cspace* avec des points intermédiaires [12].

peut être n'importe où dans l'espace de travail du robot. Mathématiquement, une singularité se produit lorsque la matrice Jacobienne est inversée et que son déterminant est près de zéro. Voici quelques exemples de singularité pour un robot articulé (figure 2.2) avec six axes rotatifs [16] :

- Frontière : la destination est hors de la zone de travail du robot.
- Poignet : les joints 4 et 6 sont alignés.
- Épaule : le centre du poignet est aligné avec le premier joint.
- Coude : le centre du poignet réside dans le même plan que le joint 2 et 3.

2.1.2 Cinématique différentielle et statique

La cinématique différentielle est l'étude de la vitesse sans la notion de force. La vitesse peut faire partie du domaine des joints ou du domaine cartésien : vitesses linéaires et angulaires de l'effecteur. La statique est l'étude des forces sans considérer le mouvement. C'est la relation entre les couples des moteurs (*Joint Space*) et la force externe appliquée à l'effecteur du robot (*Task Space*).

Matrice Jacobienne

La matrice Jacobienne est un outil mathématique important pour analyser la redondance [4], déterminer les configurations singulières, déterminer la cinématique inverse d'un algo-

4. Un robot est redondant lorsque le nombre de DDL est supérieur au nombre de variables requises pour décrire une tâche [8].

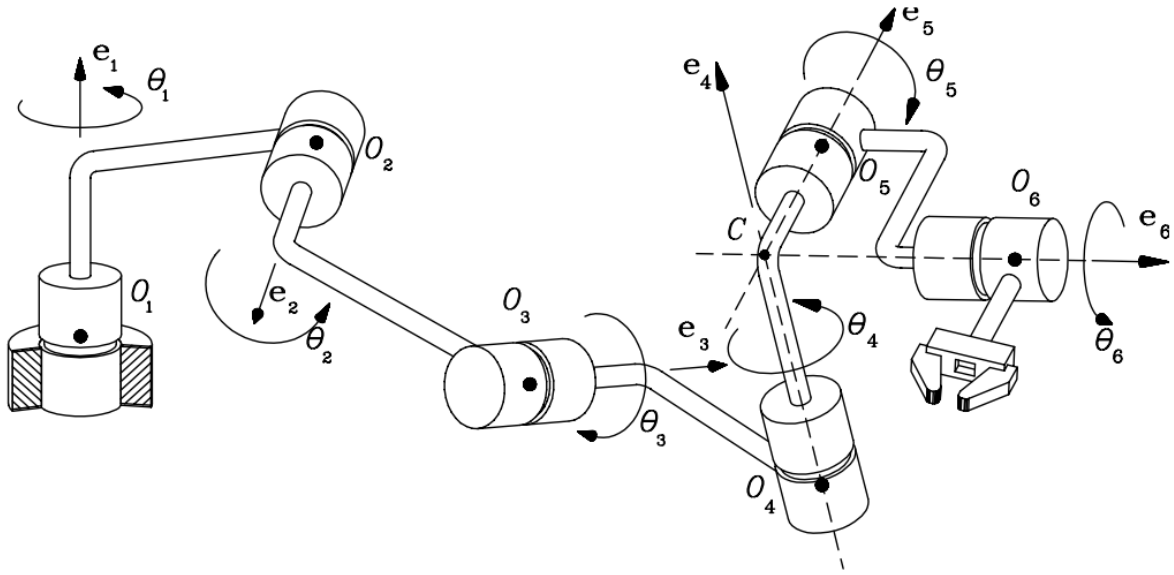


Figure 2.2 Architecture général d'un robot articulé à 6 DDL [16].

rithme, décrire la relation d'un domaine à l'autre pour la cinématique différentielle, la statique et la dynamique [8]. De plus, elle est utilisée pour concevoir des lois de commandes dans l'espace de cartésien.

Pour des robots manipulateurs redondants, où le nombre de variables dans l'espace opérationel est plus petit que le nombre de joints du robot, il existe une infinité de solution pour le calcul de la cinématique différentielle [8]. Siciliano *et coll.* [8] propose de formuler le problème en une contrainte d'optimisation linéaire pour minimiser la fonction de coût quadratique de la vitesse des joints. La solution de ce problème est la matrice *right pseudo-inverse* de la Jacobienne. Celle-ci minimise localement la norme de la vélocité des joints.

2.1.3 Dynamique

La dynamique est l'étude des forces et de ses effets sur le mouvement. Comme les robots sont des systèmes non-linéaires, ils sont sujets à l'inertie, la force centrifuge et la quantité de mouvement (*momentum*). Pour tirer le maximum du robot, il est nécessaire de tenir compte de la dynamique du robot.

Espace des phases

L'espace de configuration permet de modéliser les directions possibles pour un mouvement. Par contre, le *C-space* ne tient pas d'information quant à la dynamique du système. Par exemple, une voiture qui roule à une certaine vitesse vers un mur ne peut s'arrêter

instantanément : dû au *momentum*, la distance requise pour éviter la collision dépend de la vitesse.

L'espace des phases illustré à la figure 2.3 permet de représenter n'importe quel problème de dynamique en ajoutant une dimension de contrainte de vélocité à l'espace de configuration. Il représente donc les vélocités permises à chacun des points de l'espace de configuration. Cela permet de réduire les dérivés d'ordre supérieur, comme l'accélération, en une contrainte de dérivés de premier ordre.

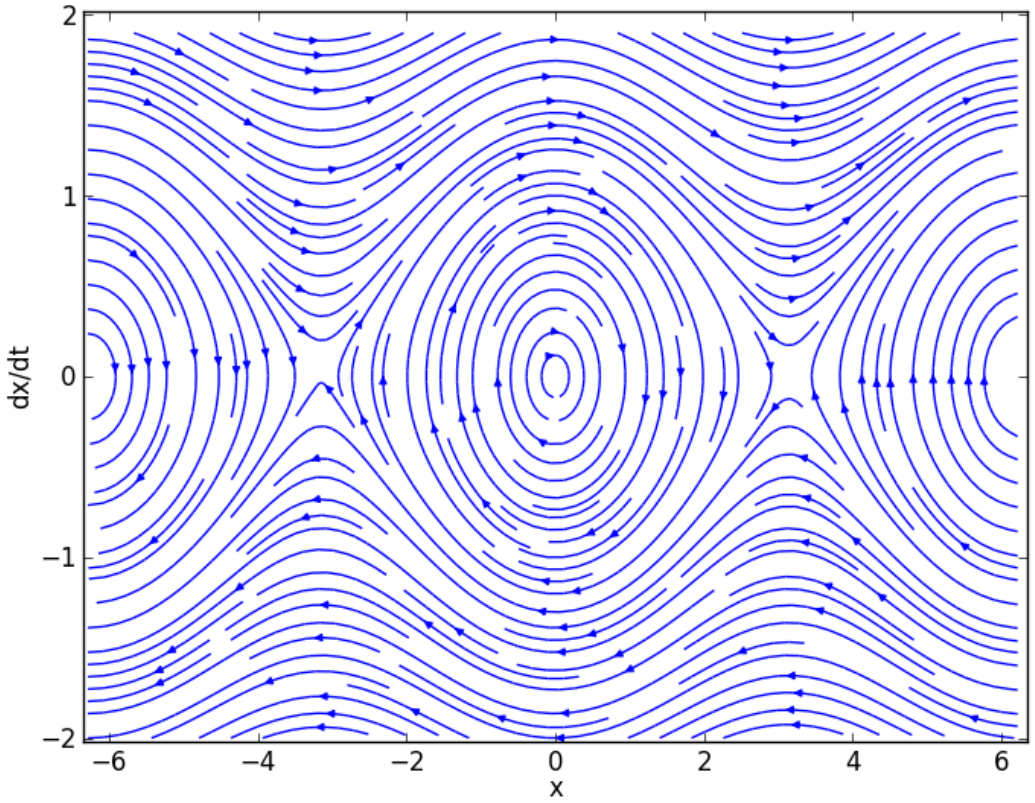


Figure 2.3 Espace des phases d'un pendule [17].

2.2 Planification de mouvement

La planification de mouvement s'intéresse à trouver la motion de chaque joint pour déplacer le robot d'un point A au point B en considérant des contraintes cinématiques, différentielles/dynamiques du robot et la collision avec l'environnement et lui-même. La planification cinématique retourne un chemin géométrique tandis qu'une planification différentielle ou dynamique retourne une trajectoire. Le tableau 2.3 présente une synthèse

des différents algorithmes de planification de mouvements, selon les concepts suivants [13, 18, 12] :

Tableau 2.3 Sommaire des algorithmes de planification de mouvements adapté de Klingensmith [18].

Algorithme	Optimal	Complet	Requête	Dimension	Répétable	Obstacle	Vitesse
Walk To	Oui	Non	Réactif	∞	Oui	Quelconque	Super Vite
Bug	Non	Oui	Réactif	2	Oui	Quelconque	Super Vite
Visibility Graph	Oui	Oui	Multiple	2-3	Oui	Polygone	Moyen
A* Lattice Grid	Oui	Résolution	Unique	2-3*	Oui	Grille	Lent
Flow Field	Non	Non	Réactif	2-3**	Oui	Quelconque	Super Vite
PRM	Probabiliste	Probabiliste	Multiple	2-100	Non	Quelconque	Lent***
RRT	Non	Probabiliste	Unique	2-100	Non	Quelconque	Vite
CHOMP	Local****	Oui	Unique	2-100	Oui	*Distance Field*	Moyen
STOMP	Local****	Oui	Unique	2-100	Non	*Distance Field*	Moyen
TrajOpt	Local****	Oui	Unique	2-100	Oui	Convexe	Vite

*Si une grille est utilisée, sinon jusqu'à 20 dimensions. **Si une grille est utilisée. Si paramétrique, jusqu'à ∞.
*** Lent si l'espace de configuration est cartographié. Vite pour les requêtes subséquentes. **** Dépend de la trajectoire initiale.

Optimal : un algorithme optimal retourne une solution optimisée avec un certain critère : temps, longueur de trajet, travail mécanique, dégagement minimum du chemin. Un planificateur est dit probabilistiquement optimal (*probabilistically optimal*) lorsque le nombre d'itérations N approche l'infini, lorsque tous les chemins existants sont disponibles. Certains algorithmes sont optimaux localement (*locally optimal*). Les optimisateurs de trajectoire vont généralement rester pris dans un minimum local pour la détection de collision [19]. Finalement, une optimalité asymptotique (*asymptotic optimality*) ne retourne pas nécessairement une solution optimale, mais s'y rapproche au fur et à mesure que les itérations augmentent [20].

Complet : un algorithme complet est capable de résoudre tous les problèmes qui ont une solution dans son domaine et dans un temps fini. De plus, il doit être capable de signaler si aucune solution n'est possible. Lorsque l'algorithme est dense [5] avec une probabilité de un, l'algorithme est dit probabilistiquement complet (*probabilistic completeness*), c'est-à-dire que la probabilité de converger vers une solution existante est de un. Il y a aussi la notion de résolution complète (*resolution complete*) lorsque la résolution de la grille de recherche est suffisamment fine pour garantir la faisabilité à trouver un chemin.

Type de requête : le type de requête (*query type*) peut être simple ou multiple. Une requête simple est utilisée lorsque l'environnement change beaucoup : la planification

5. Les échantillons sont arbitrairement proches et couvrent la quasi-totalité de l'espace de configuration lorsque le nombre d'itérations tend vers l'infini.

se fait à partir de zéro sans réutiliser des planifications du passé. La requête multiple investit du temps à bien représenter l'espace de configuration afin d'accélérer le temps de calcul pour solutionner un nouvel objectif dans une même configuration ou qui change très peu.

Dimension : certains algorithmes de planification sont efficaces et utilisés pour de faibles dimensions, tandis que d'autres sont conçus pour planifier avec beaucoup de dimensions. Le nombre de dimensions d'un système est attribué au nombre DDL du robot. Par exemple, la figure 2.4 montre deux bras robotisés de 7 DDL chacun qui manipulent une pince (1 DDL), ce qui constitue un système à 15 DDL. Un bras robotique, de 6 DDL et plus est considéré comme un système ayant beaucoup de dimensions. Plus il y a de dimensions et plus l'espace de configuration grandit.

Répétabilité : un algorithme est dit répétable lorsque, pour un même point de départ et de fin, il retourne toujours la même solution. Les algorithmes ayant des approches aléatoires sont non-répétables dû à leur nature probabiliste.

Obstacle : la représentation des obstacles est souvent indépendante de l'algorithme de planification. Toutefois, certains algorithmes requièrent une représentation spécifique.

Paramètres : certains algorithmes ont très peu de paramètres à régler et les paramètres par défaut donnent de bons résultats. D'autres ont beaucoup de réglages à effectuer et doivent être remodifiés pour des scènes différentes. Parfois, ces réglages demandent de l'intuition.

Temps de planification : le temps de planification d'un algorithme dépend de plusieurs facteurs : le nombre de dimensions du système, l'algorithme de collision et du nombre d'obstacles. Aussi, la distance minimale requise entre les points discrets va influencer le temps de planification. De plus, certains algorithmes doivent être post-traités avec un autre algorithme, sans quoi le chemin trouvé est non utilisable. Par exemple l'algorithme RRT peut trouver une solution en quelques millisecondes, mais au final prend quelques secondes pour être optimisé [18].

***Anytime* :** ce type de planificateur va continuer à chercher de meilleures solutions après en avoir trouvé une [12]. Le planificateur peut être arrêté à n'importe quel moment (*anytime*).

La planification de mouvement implique d'aborder différentes notions, comme la planification en ligne et hors-ligne, la détection de collision, la planification basée sur l'échantillonage ou sur les champs potentiels virtuels, les algorithmes d'optimisation de trajectoire

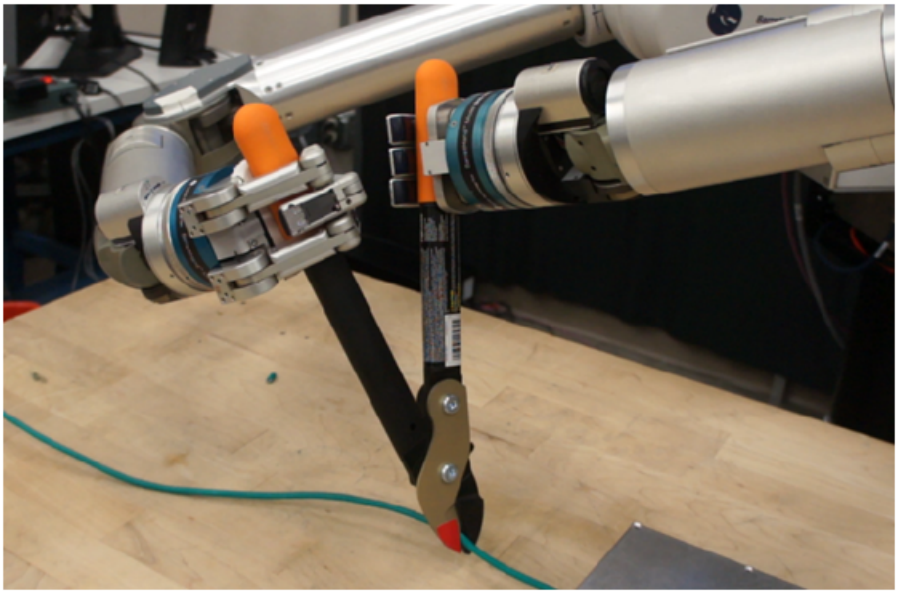


Figure 2.4 Deux bras robotiques manipulant une pince [18].

et la planification avec contraintes cartésiennes. Ces concepts sont abordés dans les prochaines sous-sections.

2.2.1 Planification en ligne versus hors-ligne

En mode hors-ligne, la planification de mouvements connecte la position initiale à la position finale. Généralement, ce type de planification est utilisé pour des environnements qui sont contrôlés ou qui changent lentement. L'environnement doit être connu à l'avance. En théorie, cette approche peut assurer une planification globale optimale en termes de distance minimale, de temps, d'énergie et de sécurité [21, 22].

Pour des environnements dynamiques, la planification en ligne est nécessaire. Elle peut s'adapter aux variations de l'environnement pour atteindre un but. Par contre, cette approche est plus complexe, exige des capteurs supplémentaires et ne génère pas une planification globale optimale [21, 22].

2.2.2 Détection de collision

La détection de collision est souvent traitée comme un module à part de la planification de mouvement. Cela permet d'avoir des algorithmes de planification qui sont indépendants des modèles géométriques (robots et obstacles). Le module de détection de collision est souvent la partie qui est la plus coûteuse en termes de temps de calcul dans la planification [13].

Pour certains types de configuration, surtout pour les robots avec beaucoup de degrés de liberté, il est peu pratique [6] de représenter l'espace de configuration [13]. En pratique,

l'algorithme de planification sonde l'espace de configuration et la détection de collision se fait dans l'espace cartésien pour chacun des points discrétisés. Des méthodes hiérarchiques et incrémentales sont utilisées pour diminuer le temps de calculs [13].

2.2.3 Planification basée sur l'échantillonage

La recherche d'un chemin se fait en sondant l'espace de configuration à l'aide d'un schéma d'échantillonnage [13]. Pour réduire la complexité exponentielle de la recherche dans l'espace de configuration, la planification sacrifie le concept de complétude et d'optimalité [13]. Lorsque l'environnement a de bonnes garanties de visibilité (*good visibility guarantees*), le taux d'approche est exponentiel [24, 25]. En pratique, le taux de convergence est généralement rapide mais avec des performances aléatoires [13, 18]. Ce type de planification est généralement utilisé pour de la planification hors ligne, car le temps de calcul est relativement long pour être implémenté en temps réel et que la recherche est probabiliste.

Les prochaines descriptions présentent les algorithmes *Probabilistic Road Map* (PRM), *Randomly Exploring Randomized Trees* (RRT) et quelques variantes dont RRTConnect et RRT* ainsi que l'algorithme AnytimePathShortening. La figure 2.5 présente respectivement l'algorithme PRM et RRT pour un problème à deux dimensions.

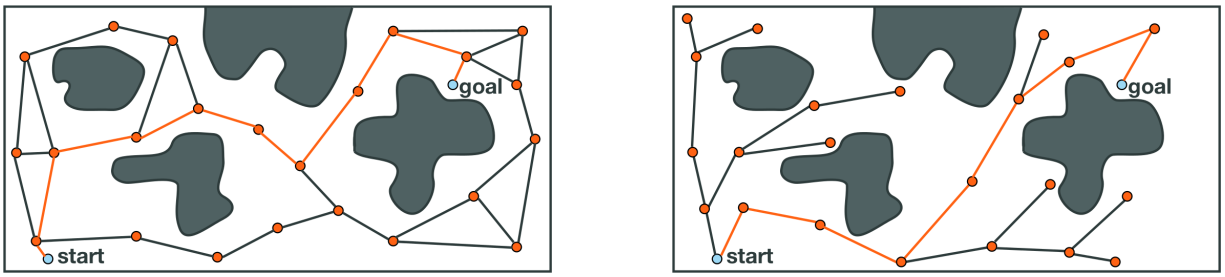


Figure 2.5 Comparaison entre PRM (gauche) et RRT (droite) [26]. Example de chemin (ligne orange) retourné pour un point de déaprt et un objectif dans l'espace de configuration 2D avec des obstacles (vert). Les points d'échantillonage (cercle orange) sont généré de façon aléatoires.

PRM

Cet algorithme crée de façon probabiliste un graphique de visibilité dans le *C-space* en utilisant des heuristiques de distance [18]. Ensuite, l'algorithme cherche dans le graphique pour trouver une solution avec A* ou Djigstra. L'algorithme est probalistiquement complet et probabilistiquement optimal. Sa force est de pouvoir résoudre des problèmes dans lesquels il y a beaucoup de dimensions. Aussi, l'algorithme permet de réutiliser le graphique pour plusieurs requêtes de manière efficace. Pour des environnements statiques,

6. Selon Pan et Manocha [23], des techniques de *machine learning* sont capables de construire efficacement l'espace de configuration.

cela permet de réutiliser le même graphique pour trouver une solution. Cet algorithme ne fonctionne pas bien lorsqu'il y a des passages étroits et il a de la difficulté à connecter des nœuds le long de contraintes de surface [15]. Il peut produire des trajets qui sont longs et inefficaces.

RRT

L'algorithme RRT cherche un mouvement sans collision qui débute à un état initial et se termine à un état final. Un arbre est généré à partir du point de départ et l'exploration se fait dans l'espace de configuration de façon randomisée avec une distribution uniforme [12]. L'algorithme peut aussi fonctionner pour des problèmes de dynamique où l'espace des phases est utilisé pour l'exploration. Les planificateurs qui sont basés sur les arbres excellent à planifier avec des dynamiques complexes dû à leur structure qui est naturellement dirigée et acyclique [26].

Cet algorithme ne requiert aucun paramètre (très peu pour les variantes), est probalistiquement complet et non-optimal. Il trouve une solution rapidement, normalement en quelques millisecondes pour un espace de configuration avec sept dimensions. Comme pour PRM, le chemin trouvé peut être long et inefficace [18].

RRTConnect

Variante de *RRT* qui génère deux arbres : au départ et à l'objectif [27]. Il est généralement plus rapide et a un pourcentage plus élevé pour ce qui est de trouver une solution par rapport à l'algorithme RRT [28]. L'algorithme ne requiert aucun paramètre.

RRT*

En rebalancant les arbres, l'algorithme devient asymptotiquement optimal et donne le chemin le plus court, donc de meilleure qualité. Contrairement aux autres algorithmes probabilistes, cette variante n'a pas besoin d'être optimisée par la suite. Par contre, il est beaucoup plus lent que RRT, voir inefficace en termes de calculs pour de hautes dimensions [29].

AnytimePathShortening

Cet algorithme exécute plusieurs algorithmes de planification en parallèle, entrelacés de techniques de raccourcissement (*shortcutting*) et d'hybridation de chemin qui sont utilisées pour optimiser localement un trajet. En pratique, celui-ci est utilisé pour obtenir un chemin optimal asymptotique [30].

2.2.4 Planification basée sur les champs potentiels virtuels

La planification basée sur les champs potentiels virtuels (*Virtual Potential Fields*) est inspiré du champ gravitationnel et du champ magnétique présent dans la nature. L'objectif est associé à un faible potentiel, tandis que les obstacles ont un potentiel élevé. Le robot

est donc poussé virtuellement vers l'objectif et repoussé par les obstacles. Le calcul se fait rapidement et peut être utilisé en ligne (temps réel). Selon Lynch et Park [12], cette méthode peut gérer des obstacles qui bougent ou qui apparaissent de façon inattendue. Toutefois, elle est très propice aux minimums locaux.

2.2.5 Optimisation de trajectoire

Cette catégorie d'algorithmes est utilisée pour deux cas de figure différents [31] :

1. Au lieu de sonder l'espace de configuration, l'algorithme se sert d'un chemin ou d'une trajectoire en ligne droite comme trajectoire de départ. Ensuite, cette trajectoire est optimisée pour respecter les contraintes d'optimisation et afin d'éviter les collisions.

2. L'algorithme utilise des chemins ou trajectoires déjà précalculées par un autre algorithme de planification de mouvement. Comme les chemins/trajectoires générés par les planificateurs randomisés ne sont pas optimaux, ils sont acheminés à un algorithme d'optimisation de trajectoire. Au final, la trajectoire est optimisée localement pour rendre le chemin ou la trajectoire de meilleure qualité.

Quelques-uns de ces algorithmes d'optimisation de trajectoire sont décrits ci-dessous, tels que *Covariant Hamiltonian optimization for motion planning* (CHOMP), l'algorithme *Stochastic Trajectory Optimization for Motion Planning* (STOMP) et un algorithme d'optimisation séquentiel convexe (TrajOpt).

CHOMP

Cet algorithme se sert du gradient covariant [19]. Avec une trajectoire naïve comme une ligne droite, CHOMP réagit au milieu environnant pour tirer la trajectoire hors de collision tout en optimisant, simultanément, des quantités dynamiques comme la vitesse et l'accélération des joints [32]. De plus, il converge vers une solution qui est lisse (*smooth*). L'algorithme est localement optimal et complet, car il s'appuie sur la trajectoire de départ, une ligne droite, comme heuristique [18]. Selon Schulman *et coll.* [29], CHOMP serait jusqu'à huit fois plus lent que RRTConnect et le taux de réussite serait plus bas.

Comme pour la plupart des optimisateurs de trajectoire, CHOMP requiert plusieurs paramètres et certains sont basés sur l'intuition d'une scène. Par exemple, CHOMP fonctionne très bien dans des environnements sans obstacle avec les paramètres par défaut. Cependant dans un environnement avec obstacles, l'algorithme va certainement rester pris dans un minimum local. Il faut alors ajuster les paramètres pour qu'il retourne un bon chemin [32].

CHOMP peut aussi être utilisé comme étape de post-traitement pour optimiser la trajectoire issue d'un autre planificateur. Au lieu de débuter avec une trajectoire en ligne droite, il est possible d'insérer une trajectoire calculée par un autre plan (p.ex. planificateur probabiliste) et optimiser la trajectoire. Cela est plus rapide, car la trajectoire de départ n'est pas en collision avec un obstacle et améliore la trajectoire.

STOMP

STOMP requiert aussi une trajectoire de départ et génère du bruit pour explorer l'espace autour de celle-ci. Une fonction de coût basée sur la combinaison des obstacles et de la qualité de la trajectoire à chaque itération peut incorporer des contraintes additionnelles telles que la limite de couple, d'énergie et d'outil. La vitesse est semblable à CHOMP, mais il peut éviter des minimums locaux dû à sa nature stochastique [33], idem pour gérer les obstacles. STOMP requiert moins de paramètres pour retourner une solution réalisable, et les paramètres demandent moins d'intuition que CHOMP. Pour naviguer dans des environnements compliqués, seuls trois ou quatre paramètres doivent être modifiés [34].

TrajOpt

TrajOpt est un algorithme d'optimisation convexe séquentiel où les contraintes non-convexe et non-égalité (*non-affine equality*) sont relâchées, approximativement linéarisées et convexifiées pour créer une fonction objective [29, 35]. Une partie importante de l'algorithme est la formulation de l'évitement d'obstacle. Dans le domaine discret, la contrainte est la différence entre la distance signée (entre un lien du robot et lui-même ou l'obstacle) et d'une valeur sécuritaire. Selon Schulman *et coll.* [29], il aurait plus de succès à trouver une solution et serait trois à neuf fois plus rapide que l'algorithme RRTConnect. Cependant pour des environnements pratiques, avec beaucoup de collisions et en partant avec une ligne droite dans l'espace des joints comme trajectoire initiale, TrajOpt a un taux de collision aussi haut que 37 % [36]. De plus, il serait plus susceptible à trouver une solution optimale localement. Aussi, TrajOpt pénalise les collisions au lieu de les interdire, donc il se peut qu'une trajectoire soit retournée avec une ou des collisions.

2.2.6 Planification avec contraintes cartésiennes

Les planificateurs abordés précédemment visent à trouver un mouvement d'un point de départ vers le point d'arrivée sans contraintes cartésiennes. Cependant, quelques planificateurs peuvent intégrer ces contraintes directement ou avec des modifications dans l'algorithme. Les planificateurs qui intègrent des contraintes cartésiennes sont appelés planificateurs cartésiens (*cartesian planners*). Les optimisateurs de trajectoire peuvent ajouter directement des contraintes de tâches dans leur fonction de coût. Les algorithmes basés sur l'échantillonnage doivent être modifiés pour ajouter des contraintes cartésiennes. Kingston

et coll. [37] proposent quelques méthodologies pour ajouter des contraintes cartésiennes aux planificateurs basés sur l'échantillonnage.

Plusieurs contraintes cartésiennes cinématiques sont nécessaires pour ce projet. L'orientation, par rapport à un effecteur, doit être contrainte pour les déplacements d'objets contenant des liquides. Cela est nécessaire pour empêcher un déversement sur la table de travail. Aussi, le déplacement d'un support à embouts de pipette doit avoir la même contrainte, car les embouts sont seulement déposés dans le support et pourraient tomber. Voici quelques contraintes cartésiennes cinématiques qui peuvent être utiles pour le projet :

- Position : limite la position d'un lien.
- Orientation : limite l'orientation du lien.
- Visibilité : limite un point sur un lien à rester visible pour un capteur.
- Joint : limite un joint à rester entre deux valeurs.

2.3 Planification de trajectoire

Certains chemins retournés par les planificateurs de mouvement sont basés sur la cinématique, c'est-à-dire qu'il retourne un chemin géométrique (en position) sans notion de temps (vitesse ou accélération). Pour exécuter cette planification, il faut ajouter une notion de temps au chemin, car tout système physique réel dépend de celle-ci [13, 12]. Pour les planificateurs qui prennent en compte les contraintes différentielles ou dynamiques du robot, la planification retournée est en fait une trajectoire et peut être utilisée telle quelle pour l'exécution du robot. La planification de trajectoire peut aussi servir à planifier un mouvement sans considérer d'obstacle. Quelques cas sont présentés par Lynch et Park [12] : une trajectoire point à point en ligne droite dans le domaine des joints et cartésiens et la génération d'une trajectoire en spécifiant des points spécifiques à atteindre (*via points*).

La section suivante présente quelques algorithmes pour ajouter une notion de temps pour un chemin retourné par une planification de mouvement.

2.3.1 Algorithmes *Time-Optimal Time Scaling*

Ces algorithmes sont utilisés afin d'ajouter une notion de temps à des chemins. Par exemple, pour des chemins précalculés par une planification cinématique, des algorithmes peuvent minimiser l'énergie consommée ou bien prévenir le déversement de liquides. Certains minimisent le temps des mouvements le long du chemin à parcourir en respectant les contraintes de vitesse et d'accélération des actionneurs du robot. Deux algorithmes maximisant la productivité du robot sont :

– *Time-Optimal Path Parameterization Based on Reachability Analysis* (TOPP-RA) : selon [38], cet algorithme est extrêmement robuste (100 % de taux de réussite) et compétitif en terme de vitesse de calcul. Cependant, il contient des sauts en accélération, ce qui inclut beaucoup de *jerk*[7].

– *Time-optimal Trajectory Generation* (TOTG) : selon [39], l'algorithme ajoute des contraintes de vitesse et d'accélération à la trajectoire. Il produit des trajectoires lisses et de façon continue pour le profil de vitesse. Certains points sur le chemin peuvent diverger du chemin de départ en respectant une tolérance. Des détections de collisions supplémentaires sont peut-être nécessaires [40]. Cet algorithme n'est pas contraint en *jerk*.

.

2.4 Contrôle de mouvement

La trajectoire retournée par la planification de mouvement est utilisée par le contrôleur. Cette trajectoire peut avoir été précalculée à l'avance (*offline planning*) ou bien calculée en temps réel (*online planning*) afin de compenser les possibles perturbations ou incertitudes. Idéalement, le contrôleur doit suivre la trajectoire planifiée avec la plus grande fidélité. Un contrôleur en boucle ouverte (*open-loop*), aussi appelé contrôleur par anticipation (*feedforward*), est le plus simple à implémenter. Il est surtout utilisé pour son effet d'anticipation et jumelé à d'autre type de contrôleur. Toutefois, il ne sera pas en mesure de suivre la trajectoire car les actionneurs sont imparfaits. Les erreurs vont s'accumuler et le robot va dévier de la trajectoire planifiée. Une loi de commande en boucle fermée est donc nécessaire [12]. Le régulateur PID est utilisé dans de nombreuses applications industrielles. La boucle de contrôle est une sommation de trois gains sur l'erreur de la consigne : un gain proportionnel (P) sur l'erreur ; l'erreur est intégrée et multipliée par le gain intégrale (I) ; l'erreur est dérivée et multipliée par le gain dérivé (D). Certains algorithmes vont n'utiliser qu'un seul paramètre ou deux. En général, ce type de compensateur ne donne pas de contrôle optimal. En effet, le compensateur à des paramètres constants et n'a pas de connaissances sur le processus. Il a aussi de la difficulté en présence de procédés non-linéaires.

L'asservissement d'un système comprend une boucle de rétroaction pour minimiser l'erreur d'une commande désirée, c'est-à-dire la trajectoire à suivre. La loi de commande peut être contrainte en position, en vitesse et/ou en accélération. De plus, elle peut être issue d'une planification dans le domaine des joints ou du domaine cartésien. Au final, il n'est possible

7. Taux de changement de l'accélération par rapport au temps.

que de contrôler ou de commander les moteurs de chaque joint du robot. Les commandes de bas-niveau (*low level*) sont disponibles en position, en vitesse et en couple, mais peuvent varier d'un robot à l'autre. Par exemple, une loi de commande peut être en position mais la commande bas-niveau appliquée au joint du robot est en couple. En général, les capteurs disponibles sont la position de chaque joint du robot, la vitesse des joints, la force à l'effecteur (domaine cartésien), et le couple appliqué à chacun des joints. Une loi de commande peut être centralisée ou décentralisée. Un contrôle centralisé tient compte de toute l'information de chacun des joints pour le contrôle de chacun, tandis qu'un contrôle décentralisé ne partage pas d'information entre les joints [12]. Dans les prochaines sous-sections, les lois de commande sont présentées via leurs commandes internes respectives, soit en position, en vitesse et en couple.

2.4.1 Commande interne en position

Lorsque les joints du robot sont contrôlés en position, il n'y a pas beaucoup de choix de contrôleurs. Généralement, un contrôleur PID est utilisé pour satisfaire la trajectoire de référence. Comme la commande n'accepte que des positions, la notion temporelle se trouve dans la fréquence des commandes envoyée. Il faut faire attention à bien choisir la séquence pour que l'exécution de la trajectoire soit fluide. Ce type de commande bas-niveau ne permet pas d'enlever les effets non-linéaires des robots. Le robot ne pourra pas être commandé avec des vitesses et accélérations maximales ainsi qu'avec des charges maximales.

2.4.2 Commande interne en vitesse

La plupart des robots commerciaux ont des contrôleurs internes en vitesse. Dans ce mode, les joints du robot sont contrôlés directement en vitesse angulaires. Comme pour la commande en position, il y a le contrôleur par anticipation et le contrôleur PID. La figure 2.6 présente le schéma de cette loi de commande. Le désavantage d'un contrôleur PID est qu'il attend une erreur avant que le joint commence à bouger [12]. En combinant le contrôleur de prédiction (la vitesse désirée) avec le contrôleur PID, cela permet de mieux suivre la trajectoire désirée. D'une part, le contrôleur n'attend pas une erreur avant de bouger, car la prédiction est utilisée. D'autre part, le PID limite l'accumulation d'erreurs. Ce contrôleur peut aussi être défini dans le domaine cartésien. La cinématique différentielle inverse est alors requise afin de passer des vélocités linéaires/angulaires cartésiennes à la vitesse des joints. Des problèmes de redondances et de singularités peuvent arriver tels que présentés à la section 2.1.1.

Normalement, la commande interne en vitesse ne permet pas d'utiliser le robot à pleine vitesse ou avec des charges maximales, car il est impossible de compenser les non-linéarités

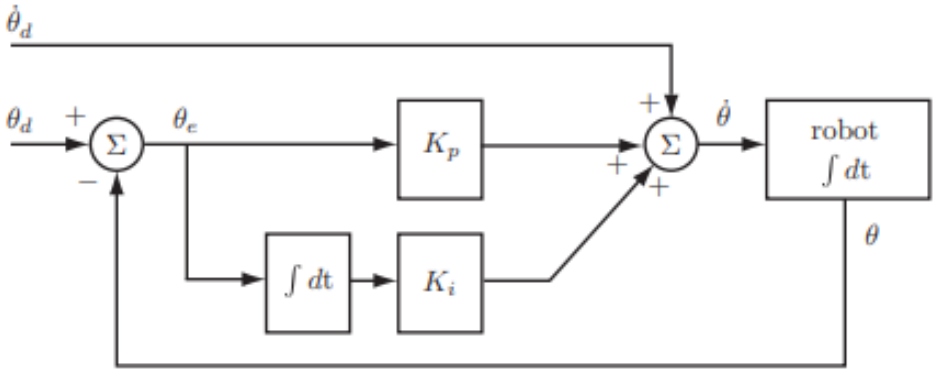


Figure 2.6 Schéma d'un contrôleur par anticipation avec PID [12].

du robot. Au cours des dernières années, quelques techniques permettent de compenser la dynamique du robot en créant un modèle virtuel dynamique basé sur la vitesse [41]. Cela ouvre la possibilité d'utiliser des contrôleurs qui requièrent normalement une commande en couple comme le contrôleur adaptatif.

2.4.3 Commande interne en couple

Contrairement aux autres commandes internes des robots (position et vitesse), la commande en torque permet de tenir compte de la dynamique du robot dans la conception de la boucle de contrôle [12]. Ainsi, il est possible de compenser les effets non-linéaires dont l'inertie, la force de coriolis et la force centrifuge. Cela permet notamment de faire bouger le robot à sa pleine capacité. La loi de commande décentralisée est appropriée lorsque la dynamique du robot est découplée, c'est-à-dire lorsque l'accélération d'un joint dépend seulement de la position, de la vitesse et du torque de ce même joint [12]. Cela est typique des robots cartésiens et lorsqu'il y a beaucoup de friction au niveau des joints. Quand la dynamique est fortement couplée, la loi de contrôle doit être centralisée afin de compenser les non-linéarités. Par contre, cela peut être complexe à calculer : un modèle exact de la dynamique n'est peut-être pas disponible ou le calcul est tout simplement trop long pour être utilisé dans la boucle de contrôle.

Contrôle par dynamique inverse

Ce contrôleur est presque identique au contrôleur par anticipation avec PID. L'accélération est donnée comme effet d'anticipation. De plus, les termes non-linéaires dus à la dynamique sont annulés et l'accélération est convertie en torque par le modèle d'inertie. Ce contrôleur peut aussi être utilisé dans l'espace cartésien avec quelques modifications.

Contrôle PD avec compensation de gravité

Une approximation du contrôleur précédant peut être utilisé pour de faible vitesse et de faible accélération [12, 8]. Cela permet de réduire la complexité du calcul.

Contrôle robuste

Ce contrôleur est conçu pour traiter les incertitudes et les perturbations imprévisibles. Dans l'espace des phases, un cône de toutes les évolutions possibles est préservé. Le contrôleur va s'assurer, dans une certaine gamme de perturbations, de ramener le système vers la trajectoire d'approche pour qu'il aie convergence malgré la perturbation. La loi de contrôle est centralisée et dans le domaine des joints.

Les techniques de contrôle robuste fournissent un taux de rejection naturel aux perturbations externes. Par contre, ils sont sensibles aux dynamiques qui ne sont pas modélisées [8]. Cette rejection naturelle est obtenue à l'aide action commutée à haute fréquence, ce qui peut être irréalisable pour la structure mécanique.

Contrôle adaptatif

Le contrôleur adaptatif vient modifier en ligne la dynamique modélisée de l'algorithme pour minimiser les paramètres d'incertitudes de cette modélisation. Cela permet en temps réel de mettre à jour ces estimations. Cette loi de commande est généralement plus douce dans le temps et la commande ne requiert pas de saut brusque. Ce contrôleur n'est pas fait pour réduire les perturbations externes. En fait, il y a dégradation du système, car le contrôleur essaie de compenser le modèle avec des perturbations qui n'ont rien à voir avec la dynamique du système. Ce contrôle est centralisé et dans le domaine des joints.

2.4.4 Redondance

Un contrôleur qui opère dans l'espace cartésien peut exploiter la redondance d'un robot pour ajouter des objectifs secondaires aux variables de joints en exploitant la pseudo-inverse de la matrice Jacobienne (section 2.1.2). Il est possible de générer un mouvement interne qui reconfigure la structure du robot sans changer la position ni l'orientation de l'effecteur. Voici quelques objectifs secondaires selon Siciliano *et coll.* [8] et Nakamura [42] : 1) le premier objectif exploite la redondance pour éloigner la configuration du robot des singularités en maximisant la mesure de manipulabilité (*manipulability measure*) ; 2) les joints peuvent être gardés le plus au centre de leur limite mécanique en position ; 3) la redondance est exploitée pour éviter des obstacles le long de la structure du robot.

2.5 Contrôle en force

Tout système mécanique réel comporte une incertitude relative de position. Par exemple, pour l'insertion d'un objet dans un réceptacle, si l'incertitude du robot est plus grande que la tolérance d'insertion de la pièce, alors il est quasi impossible d'insérer cette pièce avec un contrôle en mouvement seul. Un contrôle en force sera nécessaire pour faire ce genre de tâche, comme l'insertion d'un embout dans une pipette. Dans ce cas-ci, il faut

s'assurer qu'il y ait une certaine force d'insertion pour s'assurer que l'embout de la pipette ne tombe pas. Par exemple une erreur aussi petite qu'un millimètre peut faire en sorte que le système applique une force élevée, ce qui peut briser la plateforme robotique.

Selon Siciliano *et coll.* [8], les interactions avec l'environnement peuvent être groupées en deux catégories : le contrôle de force indirecte (*indirect force control*) et le contrôle de force direct (*direct force control*). La première catégorie applique une contrainte de force lorsque la position du robot dévie de la position désirée [43]. La seconde incorpore la force et le mouvement dans le contrôle du robot. Elle requiert un mouvement et la force désirée. Les possibilités sont :

- Le contrôle en impédance met en relation la force et la position. Elle applique un système masse-ressort (amortisseur virtuel) entre la position actuelle du robot et la position désirée. Ce contrôleur produit une certaine force à l'effecteur avec des mesures de mouvement qui correspondent à une impédance mécanique [8]. Il nécessite un robot acceptant des commandes internes en torque.
- Contrairement au contrôle en impédance, le contrôle en admittance réagit aux forces externes de l'effecteur pour contrôler le mouvement du robot. Une cible relie un système masse-ressort (amortisseur virtuel) à l'effecteur. Un capteur de force et de torque est utilisé pour recueillir les données de forces externes de l'effecteur.
- Certaines tâches requièrent une génération de force et une génération de mouvement, dans l'espace cartésien. Pour une espace de tâche avec six dimensions, chaque axe peut être contrôlé soit en force ou en mouvement. Par exemple, pour poncer une surface d'une pièce, il faut appliquer une certaine force dans un axe (position) et déplacer l'effecteur dans les cinq autres axes (position et orientation). Un contrôleur hybride de mouvement et force nécessite une bonne modélisation de contraintes naturelles et artificielles de l'environnement pour des tâches avec une surface de contact courbe [44].
- Le concept du contrôle parallèle de mouvement et force est d'utiliser la robustesse et la simplicité du contrôle en impédance avec la capacité à contrôler la position et la force comme le contrôle hybride de mouvement et de force. Deux contrôleurs sont utilisés en parallèle et le contrôleur en force a plus d'effet que le contrôleur de mouvement [44]. Ce type de contrôle est utilisé lorsqu'un modèle exact des contacts avec l'environnement n'est pas disponible. Il accommode les forces de contact imprévues durant l'exécution.

CHAPITRE 3

ROBOTISATION DE PROTOCOLES BIOLOGIQUES COMPLEXES

La plupart des robots en biologie synthétique ne font qu'une tâche spécifique. Bien que certaines plateformes ont des modules qui sont capables de réaliser deux ou trois actions, c'est insuffisant pour accomplir un protocole complexe sans intervention humaine. Actuellement, un protocole complexe nécessite une intervention humaine pratiquement entre chaque étape.

Comme indiqué dans l'introduction, le présent ouvrage se penche sur la possibilité de concevoir une plateforme robotique qui est en mesure de manipuler des instruments de laboratoire génériques utilisés en biologie synthétique dans le but de réaliser un protocole complexe sans intervention humaine. Afin d'atteindre cet objectif, plusieurs sous-objectifs sont requis. Premièrement, il faut déterminer le matériel nécessaire pour l'automatisation d'un protocole complexe. Les caractéristiques de la plateforme robotique doivent être bien définies : paramètres physiques du robot, logiciels, effecteurs et capteurs. La pipette électronique a été conçue comme outil fixe pour le robot pour l'aspiration et la dispense de liquide dans les divers réceptacles. Le système d'outil automatique a été écarté dû à son côut. Donc, le robot a deux effecteurs en tout temps : une pince de préhension et une pipette électronique. Ensuite, une analyse détaillée des différents planificateurs de mouvements est requise afin de choisir le plus approprié selon certains critères. Un algorithme de détection de collision doit aussi être choisi. De plus, quelques contrôleurs sont explorés pour des trajets avec et sans interactions. Par la suite, les différents instruments de laboratoire doivent être caractérisés pour être en mesure de les manipuler avec un effecteur adéquat. La reconnaissance et la localisation sont choisies en fonction de la robustesse. Finalement, un protocole simple et un protocole complexe sont testés en simulation et exécutés sur le robot.

3.1 Système robotique

Le choix du robot repose sur des paramètres physiques, électriques, logiciels et de son application [45]. Pour le présent projet, la vitesse d'exécution du robot n'est pas d'une grande importance. Le robot doit être capable de lever une charge de 1 kg (effecteur non compris), ce qui est suffisant pour une très vaste majorité de protocoles en biologie

synthétique. La plateforme robotique doit être capable de manipuler les objets déposés sur une table de travail de 120 cm × 120 cm. Une répétabilité dans le domaine cartésien de plus ou moins 1 mm est requise, dictée par l'insertion entre le bout d'une pipette et de son embout. La simulation est aussi un grand facteur pour le choix du robot. Celle-ci permet de valider plusieurs actions avant de le tester dans le monde réel. Cela évite des bris (robot et instruments) et accélère le développement. Finalement, le système doit être robuste : certains échantillons à manipuler valent quelques milliers de dollars, et puisque certains protocoles s'étendent sur plusieurs jours, une erreur vers la fin du protocole peut engendrer une perte énorme pour l'utilisateur.

Même si la majorité des solutions actuelles utilisent des robots de types cartésiens, ils n'ont pas été envisagés dû à leur coût (100 000$ +) et leur orientation est limitée pour atteindre certraines poses requises par des protocoles complexes. En ajoutant deux autres joints rotatifs à ce sytème, l'orientation aurait pu être contrôlée. Par contre, cela aurait engendré des coûts supplémentaires. Aussi, chacune des solutions envisagées requiert d'acheter des objets (réceptacles, embout, module) compatibles avec la plateforme dont les prix sont souvent plus chers que les versions génériques. Avec le robot cartésien OT-2, la plupart des réceptacles et des embouts sont compatibles avec leurs plateformes, par contre il n'est pas possible d'ajouter de modules existants car ils requiert des modules compatibles créés par la compagnie. L'OT2 est aussi beaucoup moins dispendieux que les autres solutions, soit environ 10 000 $ avec quelques modules. Cela fonctionne bien pour des protocoles simples, mais c'est peu extensible pour des protocoles complexes.

Le robot articulé UR3 de Universal Robots, montré à la figure 3.1, a été sélectionné pour son coût, son orientation flexible, son aspect collaboratif et le fait qu'il soit bien établi. De plus, ce robot est compatible avec l'intergiciel *Robot Operating System* (ROS) [46]. Aussi, la plateforme robotique a son propre outil de simulation, URSim. Le robot Franka Emika, un prétendant au UR, n'a pas été retenu en raison de sa nouveauté et de la crainte de ne pas avoir le robot à temps pour ce projet. Les solutions offertes par des bras industriels ont été écartées en raison du besoin d'installer une cage de sécurité, de leurs coûts et de leur écosystème fermé.

Le robot doit interagir avec son environnement pour réaliser un protocole biologique complexe. Il n'y a présentement pas de solution adoptée universellement pour manipuler et interagir avec l'environnement. En 2015, Amazon initia la compétition *Amazon Picking Challenge* car elle juge que les solutions commerciales pour la cueillette d'objet dans les environnements non structurés restent un défi difficile [48]. Afin de respecter le cahier de charge pour l'architecture la plus robuste, il a été convenu de travailler dans un en-

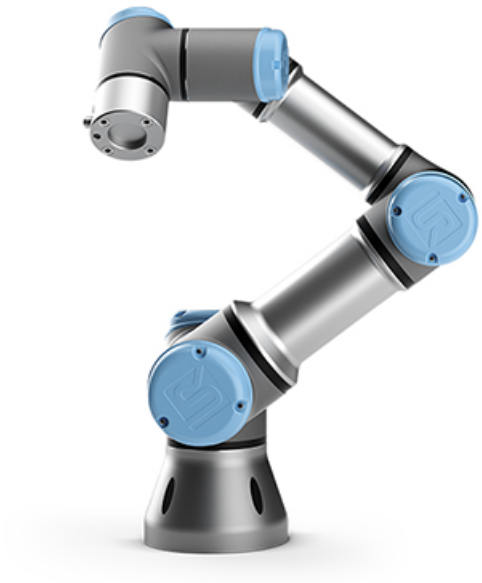


Figure 3.1 Universal Robots UR3 [47].

vironnement structuré, c'est-à-dire un environnement où tout est connu à l'avance avec une grande précision en termes de positions. L'environnement d'un laboratoire biologique permet ce choix. Il n'y a donc pas de système de vision pour les travaux présentés. Cela offre notamment des solutions plus fiables qui nécessitent moins de capteurs [49]. Comme la structure mécanique du robot UR3 n'est pas très compliante, certains outils doivent être conçus avec une compliance passive (*passive compliance*). Cette compliance est nécessaire pour ne pas endommager l'interaction avec certains objets fragiles. Sans celle-ci, il aurait fallu sacrifier la performance du robot en réduisant sa vitesse [12]. Ces choix ont un impact quant à la façon d'intégrer les objets dans l'environnement de travail structuré, la reconnaissance d'objet, la localisation d'objet ainsi qu'à la manipulation des objets.

3.1.1 Intégration des objets

L'objectif de ce projet est d'intégrer à la plateforme des appareils génériques déjà disponibles dans les laboratoires, car le coût des versions numériques destinées à l'utilisation robotique est très onéreux. Le but est d'éviter le plus possible de modifier ou de concevoir des objets ou outils, mais dans certains cas, cela est inévitable.

Le robot est disposé sur une table optique ayant des trous filletés à chaque pouce. Chaque objet/outil ajouté sur la table de travail doit être inséré sur un support (réalisé à l'aide d'une imprimante 3D). Ce support est fait pour être attaché de façon précise à la table et à l'objet en question. Aussi, la conception de certains outils peu complexes est envisagée pour

ajouter de la compliance mécanique à certains outils, comme un stylet pour l'interaction avec des écrans tactiles afin de diminuer les risques de brisures. Ces conceptions servent à diminuer la complexité générale du système et à pallier des limitations du robot.

La modification de la pipette manuelle est un exemple où les ressources étaient moins élevées versus la manipulation directe qui aurait requis un système de vision et/ou une pince complexe, comme celle montrée à la figure 1.2. Comme la pipette est l'instrument le plus utilisé dans les protocoles, nous avons choisi de la modifier pour que l'intégration avec la plateforme robotique soit simple et efficace. Finalement, un étaleur a été modifié pour laisser la gravité faire le contact avec des substances facilement déformables.

3.1.2 Reconnaissance d'objet

La plateforme robotique doit reconnaître les différents instruments déposés sur la table de travail. La majorité des techniques consiste à détecter des zones d'intérêts (*features*), c'est-à-dire trouver des caractéristiques dans une image qui sont propres à chaque objet comme des coins, des bordures et des taches (*blobs*). Certains objets n'ont pas beaucoup de caractéristiques tandis que d'autres ont une empreinte détaillée. Ces méthodes sont écartées car ils ne donnent qu'un taux de détection autour de 90 % [50, 51, 52, 53]. Des méthodes intrusives peuvent augmenter le nombre de caractéristiques pour faciliter la reconnaissance à un taux de 100 % [54]. Par contre, il faudrait ajouter un *DataMatrix code* sur chaque objet, ce qui n'est pas pratique surtout pour le grand nombre et de la petitesse des embouts de pipettes.

Afin d'avoir une solution robuste en tout temps, la table de travail utilisée est divisée en un système de coordonnées. L'opérateur place l'instrument à l'endroit voulu et entre manuellement le type d'instrument à l'endroit déposé. Si l'objet possède des éléments dynamiques (p.ex. bouton poussoir), l'utilisateur doit aussi entrer l'état initial de l'objet. Cette technique est simple et le niveau confiance pour la reconnaissance est de 100 % si l'opérateur ne se trompe pas. Par contre, il faut entrer les coordonnés manuellement, pour chaque objet. Une interface graphique, présenté à l'annexe A, est conçue pour faciliter le tout.

3.1.3 Localisation d'objet

Localiser un objet implique de déterminer sa pose, c'est-à-dire la combinaison de la position (x, y, z) et de l'orientation (r, t, l) d'un objet, sur la table de travail par rapport à un référentiel connu. Comme pour la reconnaissance d'objets, les objets sont précisément ancrés sur la table de travail avec un système de coordonnées. Cette technique est simple et robuste. Par contre, il faut créer des supports pour que l'appareil soit installé précisément

sur la zone de travail. De plus, la flexibilité est réduite quant au positionnement et à l'orientation des appareils. Aussi, cela ne fonctionne que pour des objets statiques qui ne changent pas.

3.1.4 Manipulation

Les manipulations et interactions nécessaires avec l'environnement sont la préhension d'objets, l'insertion d'objets, appuyer sur des boutons poussoir et tactiles, étaler des fluides sur la surface de l'agar contenu dans une boîte de pétris, aspirer et dispenser des liquides.

Préhension. La pince de préhension est utilisée pour prendre les divers objets dans l'espace de travail. Le mode *parallel grasping* est utilisé et la pince est contrôlée en force afin de bien tenir l'objet sans toutefois la déformer dû à une force excessive. Ce choix vient du fait que la totalité des objets qui doivent être préhendés ont une surface rigide, et sont bien ancrés dans l'environnement et sont connus à l'avance. De plus, le contrôle est simple et efficace. Par contre, cette méthode ne peut agripper des objets déformables ou fragiles. Afin d'utiliser plusieurs outils, la pince de préhension doit pouvoir saisir les autres outils qui ne sont pas attachés au robot.

Insertion. Comme le UR3 n'est pas commandable en couple, quelques contrôleurs en force ne peuvent être utilisés. Un contrôleur en admittance, aussi appelé *position based impedance control*, a été conçu spécifiquement pour l'insertion. Ce contrôle requiert un capteur de force et de torque à l'effecteur du robot. Le robot UR3 a ce type de capteur, mais n'avait pas une assez bonne résolution. Nous nous sommes donc tournés vers le FTS-300 de Robotiq inc.

Appuyer. Le robot UR3 et les effecteurs ne sont pas compliants. Afin de ne pas endomager les équipements du laboratoire lors de l'interaction avec des boutons, surtout les écrans tactiles, un stylet est utilisé avec le contrôle en admittance. Ceci permet aussi, d'exécuter cette action plus rapidement, car avec la boucle du robot (125 Hz) et du FTS (90 Hz), il n'est pas possible d'avoir un contact compliant à moins de contrôler le robot très lentement.

Étaler. Un étaleur est utilisé pour étendre une goutte de liquide sur une surface de gélose. Un des axes est compliants, donc c'est le poids de l'étaleur avec la gravité qui étale la goutte d'eau. La force minimale du capteur (2 N) est trop grande et déforme la gélose. Un contrôleur de mouvement est utilisé car la compliance de l'étaleur modifié fait naturellement contact avec la gélose via la force de gravité.

Aspirer/Dispenser. En général, l'embout de la pipette est en contact seulement avec des liquides. Comme la viscosité de ces liquides est semblable à l'eau, il n'y aura pas

une grande force. Un contrôle de mouvement pourra aspirer et dispenser les liquides. Toutefois, pour aspirer tout le liquide dans un contenant, un contrôle en force est nécessaire pour se rendre au fond. Le contrôle en admittance développé ne peut être utilisé car un simple contact à basse vitesse de l'embout avec le réceptacle déforme l'embout, ce qui obstrue le conduit pour l'aspiration ou la dispense. Il serait mieux d'ajouter un axe de compliance [1] à la pipette pour ces cas.

Pour la manipulation et le déplacement des objets, la pince de préhension de Robotiq (2-Finger 140) a été choisie, car la majorité des instruments ne requièrent que deux doigts pour être manipulés adéquatement. Aussi, elle est compatible avec le UR3 et a un contrôle en force et en position. Un package ROS est disponible pour la commander. De plus, deux modes de préhension sont disponibles : parallèle et englobante.

Point de prise de l'objet

Avant d'appréhender un objet, il faut savoir comment bien positionner la pince pour une prise stable. Comme il n'y a pas de système de vision, il faut nécessairement avoir des informations de prise *a priori*. Pour une pince parallèle, la méthode choisie est d'utiliser le centre géométrique de deux surfaces planes ainsi que la distance qui les séparent [55]. Avec ces informations, il est possible de construire une multitude de points de prise pour un objet. Il est également possible de générer une heuristique de surface pour la prise ou bien de trouver la pose qui est près du centre de gravité. De plus, il est possible de privilégier les prises qui sont le plus près du vecteur de gravité. Si un des effecteurs du robot avait des ventouse, il serait possible de prendre l'information des différentes surfaces géométriques pour positionner l'effecteur. La distance du point de prise réel est comparée, avec une tolérance, avec celle désirée. Cela permettrait de valider que l'objet préhendé concorde avec l'information donnée.

3.1.5 Logiciels et librairies

ROS est un intergiciel, une collection de logiciels au code source ouvert (*open source*) pour le développement des robots. Il fournit certains services d'abstraction matérielle, contrôle des appareils de bas niveau (*low-level device control*), offre un système de messages entre processus et une gestion de modules (*package*). Différents styles de communication sont implémentés : communication synchrone *Remote Procedure Calls* (RPC), le *streaming* asynchrone de données via une coordination par évènement (*event-based*) appelée *topic* ; le *Parameter Server*, un système de mémoire partagé (*shared data space*) [56]. De plus, ROS offre des outils pour faciliter l'implémentation de nouveaux modules. La programmation

1. Partie d'un objet qui est flexible ou souple dans une ou de multiples directions (ex. suspension d'une automobile).

distribuée de ROS permet d'avoir un transfert direct et transparent entre la simulation et le code de contrôle du robot.

Même si ROS offre une variété de modules logiciels, il n'est pas encore adopté universellement par les industries en robotique. La plupart des *packages* ROS sont conçus par la communauté de façon bénévole sur une base volontaire. Cela fait en sorte que certains modules moins matures ont des problèmes (*bugs*) informatiques. Certaines associations tentent de changer cela, notamment ROS-Industrial qui ont pour mission de rendre des modules ROS accessibles pour les industries avec les mêmes normes de qualité [57]. En 2017, le projet ROSIN fut lancé, financé par le programme Horizon 2020 de recherche et d'innovation de l'Union Européenne [58]. À titre d'exemple, le module ROS *MoveIt* est utilisé par plus de 126 robots dans la communauté, dont certains sont utilisés en mers profondes et dans l'espace [59].

3.1.6 Commandabilité du robot

Les joints du robot UR peuvent être commandés en position ou en vitesse. Cependant, ils ne peuvent être contrôlés en torque. Ces commandes sont accessibles via : le *teach pendant*, c'est-à-dire une tablette avec une interface-usager pour le contrôle et l'ajout de tâches des robots UR ; le langage *Universal Robotic Script* (URScript), la programmation basée sur Python pour contrôler les robots UR ; ou à l'aide d'une interface de programmation applicative (API) en langage C [60]. L'API en C est la façon la plus rapide d'envoyer des commandes. Par contre, les commandes ne peuvent pas être envoyées plus vite que 125 Hz, l'API est peu documentée, elle doit être installée à même le contrôleur et elle rend le *teach pendant* inutilisable. Le contrôle du robot via le *teach pendant* est simple et facile à utiliser pour commander le robot, et ce même pour une personne non-initiée à la robotique. Celle-ci ne peut-être réalisée que pour de la programmation hors ligne.

Pour programmer un robot UR, la méthode choisie est d'utiliser le langage URScript du côté robot, car c'est la méthode utilisé pour communiquer avec le module ROS *Universal_Robots_ROS_Driver* et qui offre le plus d'options pour une planification modulaire. Ce module est développé en collaboration avec Universal Robots et le Centre de recherche sur l'information de la technologie FZI, soutenu par ROSIN. En plus de pouvoir contrôler le robot via ce module, cela permet d'utiliser l'écosystème ROS qui comporte beaucoup de modules pour des applications tels la planification de mouvement, la planification de tâches, la vision artificielle et le contrôle de robot. Le logiciel graphique, créé par Universal Robots, est laissé de côté afin de pouvoir exécuter des protocoles biologiques complexes de façon automatisée. Présentement, avec le logiciel de UR, il faudrait refaire une partie de la programmation à chaque fois qu'il faut changer de place un objet.

Commande bas-niveau

Pour commander le robot, il y a deux contrôleurs. Le contrôleur en position servoj et le contrôleur en vitesse speedj. Le premier est une commande en position (radian) pour chacun des six joints. Il fait une interpolation entre l'état actuel et l'état désiré avec une boucle de rétroaction de gain P. Le second envoie directement des commandes de vitesse aux joints du robot. Il n'y a pas de rétroaction ni d'interpolation pour ce mode. Comme ces contrôleurs affectent directement le robot, ils sont nommés contrôleurs de bas-niveau ou contrôleurs internes.

Commande haut-niveau

Le module ros_control [61] est utilisé pour les différents contrôleurs. Celui-ci expose facilement le bas-niveau matériel d'un robot à ROS. De plus, il promouvoit la réutilisation du code de contrôle. Aussi, plusieurs controleurs typiques sont déjà implémentés et prêts à être utilisés. Enfin, le code est utilisé pour des applications en temps réel. Les contrôleurs utilisant cette implémentation sont nommés contrôleurs de haut-niveau ou contrôleurs externes.

Le module ROS du robot UR3 propose trois contrôleurs implémentés avec ros_control : pos_joint_traj_controller, vel_joint_traj_controller puis joint_group_vel_controller. Les deux premiers vont commander le contrôleur interne servoj et le dernier speedj. Le contrôleur pos_joint_traj_controller est un contrôleur en boucle ouverte au niveau de ros_control. Le contrôleur vel_joint_traj_controller est un contrôleur par anticipation avec PID. Des paramètres par défaut sont disponibles mais ne sont pas optimisés. Enfin, le contrôleur joint_group_vel_controller un contrôleur en boucle ouverte pour la boucle interne et pour la boucle externe.

3.1.7 Cinématique

Le robot UR3 a 6 DDL, soit six joints en rotation. La cinématique directe, représenté par les paramètres Denavit–Hartenberg [2] (DH) de la figure 3.2, ainsi que la cinématique inverse analytique du UR3 sont données par [62, 63]. Comme les planificateurs de mouvement utilisent la cinématique inverse plusieurs fois [63], il s'avère important de regarder les deux types d'algorithmes qui sont disponibles : analytique et numérique. La version analytique (*closed form*) donne une solution très rapide en quelques microsecondes. Elle ne fait que trouver la solution à une équation bien définie. L'espace nul de l'ensemble de solutions peut être exploré car toutes les solutions sont calculées. Des modules ROS tels ur_kinematic [46] et IKFast [63] sont disponibles comme méthodes de cinématiques inverses analytiques. Pour un robot avec 6 DDL, il faut respecter certains critères pour trouver une solution

2. Convention pour décrire les liens d'une chaîne cinématique dans l'espace (ex. un robot manipulateur).

analytique [64, 63]. En général, il n'y a pas de solution analytique pour des robots ayant 7 DDL et plus. Beaucoup moins vite que l'algorithme analytique, dû à la recherche d'une solution de façon itérative, une solution numérique est généralement trouvée en quelques millisecondes et dernièrement en quelques centaines de microsecondes [65]. Le taux de réussite est moins bon, mais certains algorithmes réussissent à avoir un taux de réussite à plus de 99.5 % [65, 66]. Aussi, il est important qu'un algorithme itératif ne retourne pas de solution lorsqu'il n'y en a pas.

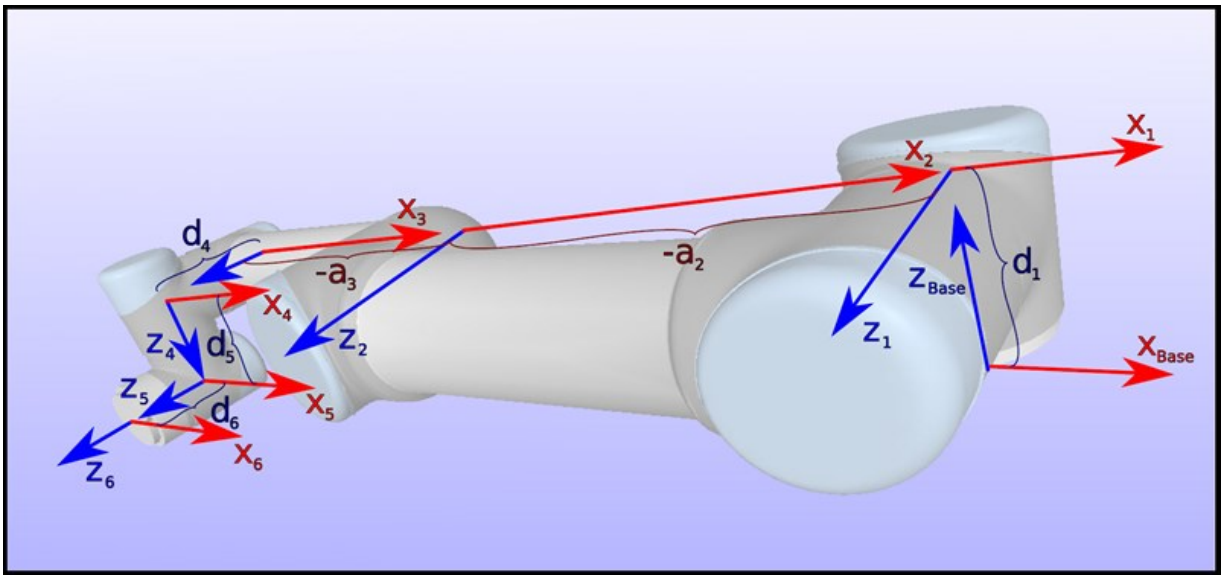


Figure 3.2 Paramètres DH pour un robot UR [67].

En théorie, le choix le plus judicieux serait d'utiliser un algorithme analytique. Cependant le UR3, comme la plupart des robots, a besoin d'une calibration afin de minimiser les incertitudes en position du robot lors de la fabrication. Comme nous sommes détachés du programme du UR, le fichier de calibration n'est pas pris en compte dans la description géométrique du robot. Sans cette calibration, il y jusqu'à 5 mm d'erreurs dans le domaine cartésien dépendamment de la pose et indépendamment du type d'algorithme de cinématique utilisé. Or, le cahier de charge cible une erreur absolue de ± 1 mm. Donc, un script fut réalisé afin d'ajouter la calibration à la modélisation d'un robot UR pour que l'erreur relative soit celle indiquée par le robot, soit de ± 0.1 mm dans le domaine cartésien. En prenant en compte la nouvelle géométrie du robot, cela fait en sorte que l'algorithme ur_kinematic ne peut plus être utilisé sans modification, car il faut ajuster la représentation cinématique avec les nouvelles valeurs de calibration. Aussi, IkFast n'arrive plus à trouver une solution analytique, car il est supposé que l'ajout de la calibration fait en sorte que la contrainte d'avoir les trois premiers ou les trois derniers joints doivent être sécant avec un point commun [64]. Les solutions numériques fonctionnent car elles

cherchent la solution en itérant. Présentement, la solution est de prendre Track-Ik car il retourne une solution qui inclut la calibration, est la solution numérique la plus rapide et celle qui donne le plus haut taux de réussite en incluant la calibration du robot. Même si cet algorithme est cent fois plus lent que la solution analytique, ce n'est pas la partie qui va contraindre le temps de calcul dans l'architecture de planification.

3.2 Conception de la pipette électronique

La pipette est l'outil le plus utilisé dans les manipulations biologiques et sert de transport entre différents appareils. Pour ces raisons, une modification de la pipette a été conçue comme effecteur au UR3.

La figure 3.3 montre la pipette électronique conçue, basée sur une modification de la pipette manuelle de Gilson (montrée à la figure 3.4) qui a une excellente précision et répétabilité [68]. De plus, Gilson a des spécifications beaucoup plus sévères par rapport à la norme ISO-8655 qui définit les requis pour effectuer des calibrations précises et fiables pour des pipettes à piston [69, 70]. Pour s'assurer d'avoir des caractéristiques équivalentes, la partie qui sert à pipetter a été gardée. La partie qui sert à sélectionner le volume désiré a été enlevée pour être remplacée par un moteur pas-à-pas ainsi qu'une vis sans fin (*leadscrew*). Lorsque le moteur tourne, un mouvement linéaire est généré, ce qui cause un déplacement de volume d'air pour l'aspiration ou l'éjection de liquides. Aussi, l'embout peut être éjecté en amenant le piston à sa position la plus avancée. Ce système est manipulé électroniquement via un protocole série (RS-485). Le tableau 3.1 compare la tolérance d'une pipette manuelle conçue par Gilson versus une pipette électronique conçue par Opentrons [68, 71]. La pipette manuelle de Gilson est plus précise et plus répétable.

Des tests préliminaires basés sur la procédure de calibration de Gilson [72] suggèrent que la pipette électronique conçue serait plus précise et répétable que la pipette manuelle de Gilson. La figure 3.5 présentent les observations. La plage de volume pour une pipette manuelle (P200) se situe autour de 20 µL à 200 µL tandis que la pipette électronique peut

Tableau 3.1 Tolérance d'une pipette P10 (Gilson vs Opentrons).

Volume (µL)	**Erreur systématique** (µL) †		**Erreur aléatoire** (µL) ‡	
	Gilson	Opentrons	Gilson	Opentrons
1	0.025	0.1	0.012	0.2
5	0.075	0.25	0.03	0.25
10	0.1	0.3	0.04	0.05

†Précision. ‡Répétabilité

pipetter des volumes aussi petits que 1 µL jusqu'à 275 µL. Finalement, elle accepte différent type de pipettes : P10 (1 µL à 10 µL), P20 (2 µL à 20 µL), P200 (20 µL à 200 µL) et P1000 (100 µL à 1000 µL). Accepter plusieurs gammes de volumes était physiquement impossible avec une seule pipette manuelle.

3.3 Planification de mouvement

La planification de mouvement considère les mouvements qui n'interagissent pas avec l'environnement. Comme mentionné à la section 3.1, l'environnement de la plateforme robotique est structuré et il n'y a pas de perturbation externe. La planification hors ligne a été choisie car la réactivité du robot n'est pas requise et elle permet une planification optimale globale théorique. La planification différentielle/dynamique est laissée de côté afin de simplifier la planification, et l'exécution du robot à sa pleine capacité n'est pas requise. Aussi, le contrôle interne du UR3 ne permet pas un contrôle en torque, donc la planification avec des accélérations ne pourrait être respectée. Pour ces raisons, seulement les planificateurs cinématiques sont pris en considération. Finalement, les mouvements générés sont enregistrés afin de pouvoir exécuter un protocole sans avoir à replanifier.

Pour la planification de mouvement, MoveIt, un logiciel pour la manipulation mobile utilisé en robotique, a été choisie car il est disponible en distribution libre de droit et pour sa compatibilité avec ROS. L'architecture de ce système est montrée à la figure 3.6. Il y a trois façons d'interagir avec ce système : en C++, en Python ou via une interface graphique. *A priori*, la représentation du robot est donnée par le *Unified Robot Description Format* (URDF) qui contient la cinématique directe, la dynamique et les capteurs du robot.

Plusiseurs algorithmes de planification de mouvement sont déjà intégrés avec MoveIt dont *The Open Motion Library* (OMPL) [75], une librairie contenant des algorithmes de planification de mouvement randomisé – PRM, RRT, RRT*, RRTConnect. Quelques algorithmes d'optimisation de trajectoire sont aussi inclus tels CHOMP, STOMP et TrajOpt. Ces dernier sont encore en développement actif, donc certaines implémentations sont limitées. La planification avec la librairie Tesseract aurait aussi été un bon choix, mais elle est encore en développement active.

3.3.1 Analyse d'algorithmes de planification de mouvement

Il n'y a pas encore de caractérisation à savoir si un algorithme de planification est mieux adapté à un type de problème [76]. Notre problème, pour les mouvements sans interactions, est de se déplacer dans un environnement de travail structuré avec un robot UR3 ayant 6 DDL. L'environnement comprend divers réceptacles et modules déposés sur une table de travail. L'environnement physique de la plateforme robotique est montré à la figure 3.7.

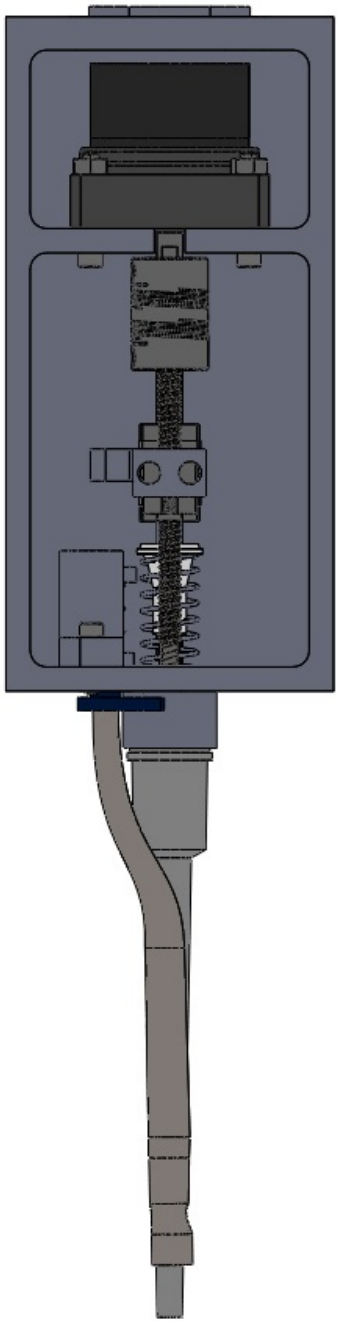


Figure 3.3 Pipette électronique pour bras robotique (P200).

Figure 3.4 Pipette manuelle Gilson (P200) [73].

Pour réussir le problème, le planificateur doit trouver un chemin entre une configuration de départ et de fin dans un certain lapse de temps avec des contraintes cinématiques.

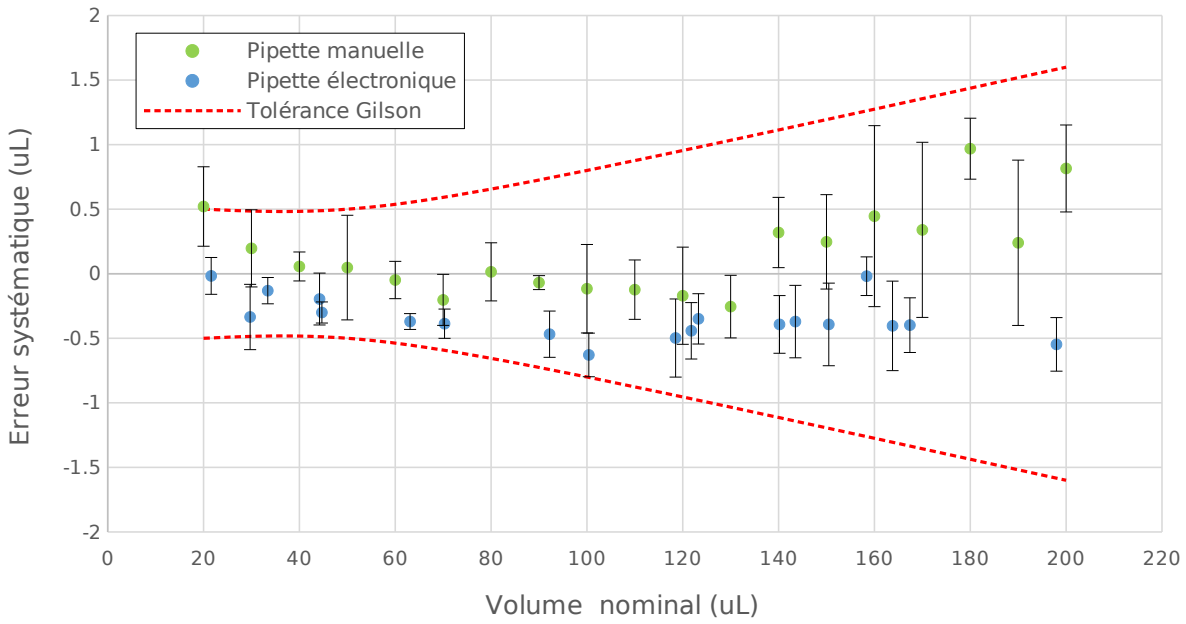


Figure 3.5 Erreur systématique et aléatoire (barre d'erreur) d'une pipette manuelle versus la pipette électronique (P200).

Quatre tests ont été réalisés afin de dégager une vue d'ensemble sur la faisabilité d'exécuter des protocoles biologiques. Ces tests ont une contrainte de départ et de fin dans le domaine des joints. Il n'y a pas d'autres contraintes cinématiques durant le trajet. Les configurations de départ et d'arrivée ont été sélectionnées pour représenter plusieurs cas typiques : faciles (grande visibilité) et difficiles (encombrement avec plusieurs obstacles) ; avec de courts et longs déplacements. L'idéal serait d'avoir un algorithme qui prend le moins de temps possible, qui donne un chemin de grande qualité, qui est le plus court possible et qui ne retourne pas une solution en collision.

L'infrastructure de Moll *et coll.* [76] est utilisée pour analyser les différents planificateurs entre eux. Il propose aussi plusieurs métriques pour l'analyse comme le temps de planification, le taux de réussite où la planification a trouvé une solution, le taux d'exactitude qui représente les solutions qui respectent certains critères (évitement d'obstacles, contraintes), la distance du chemin qui est la somme du parcours des angles de chaque joint et la qualité du chemin. L'annexe B présente une description plus détaillée. Certains algorithmes peuvent optimiser des critères, et le critère d'optimisation choisi est celui de minimiser la distance parcourue des joints afin de réduire le temps d'exécution et d'avoir des chemins faisables pour le robot.

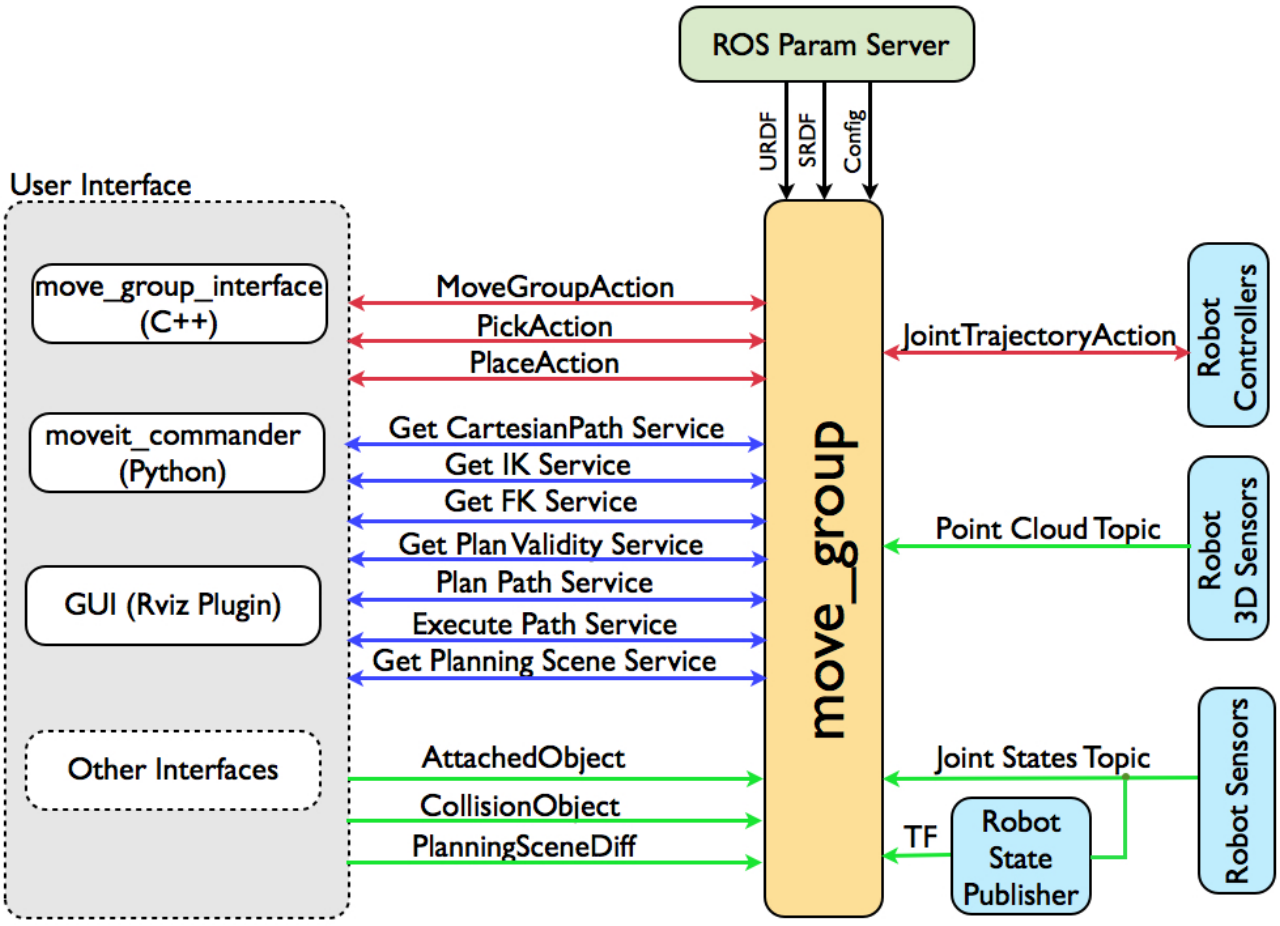


Figure 3.6 Architecture du système MoveIt [74].

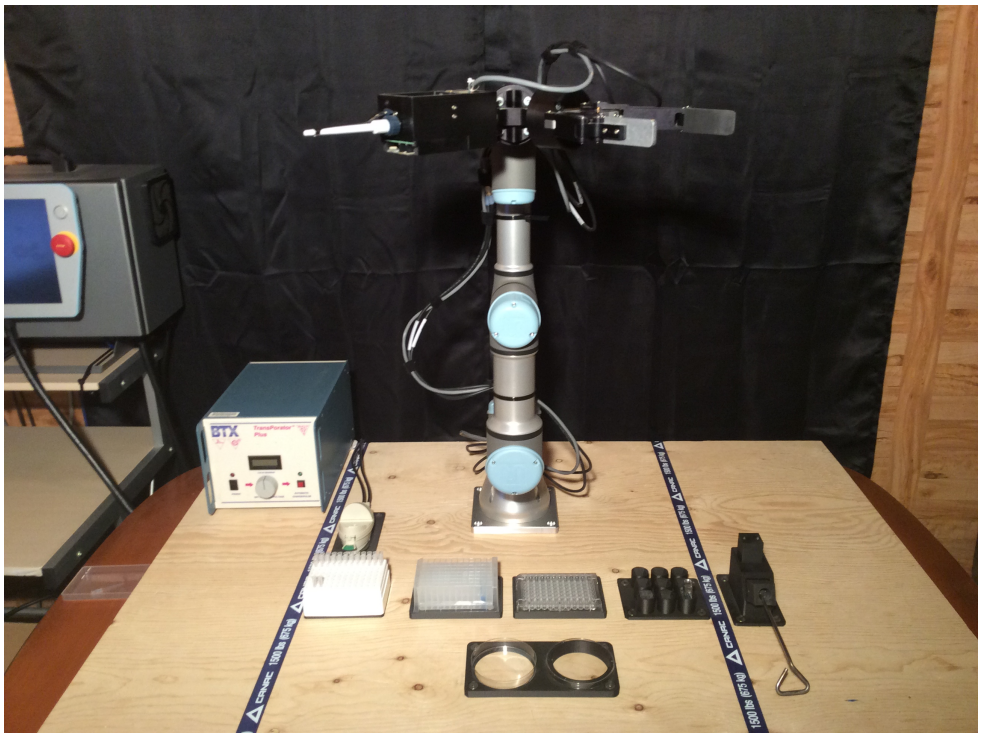


Figure 3.7 Environnement physique de la plateforme robotique.

Huit planificateurs sont analysés : six sont basés sur l'échantillonnage provenant de la librairie OMPL, soit RRT, RRTConnect, RRT*, PRM, PRM* et AnytimePathShortening ; les deux autres sont CHOMP et STOMP, tous deux des algorithmes d'optimisation de trajectoire. La comparaison entre ces deux types de planificateurs peut parfois être difficile pour certaines métriques, car l'implémentation diffère beaucoup. Malheureusement, l'algorithme TrajOpt n'a pu être analysé car il n'est pas encore inclus dans MoveIt. De plus, la technique d'utiliser un algorithme d'OMPL pour ensuite l'optimiser avec CHOMP n'a pas été sélectionné due à l'infrastructure qui ne permet pas ce type d'analyse [77]. La configuration de ces algorithmes est présentée à l'annexe C. Les algorithmes qui discrétisent leur recherche doivent avoir un paramètre qui indique la distance minimale que doivent avoir les points discrets. Ce paramètre important doit être choisi adéquatement (comme expliqué à l'annexe D).

Il est utile de représenter certaines métriques avec des diagrammes en boîtes (*box plot*). Les paramètres utilisés sont donnés à l'annexe E. Chaque type de planification a été exécuté 50 fois et le temps maximal alloué était de 20 secondes avant de retourner une solution. Les tests ont été effectués sur un processeur Intel i7-7700HQ à une fréquence de 2.8 GHz avec 16 GB de mémoire RAM.

Les métriques d'analyse les plus importantes sont le taux de réussite et le taux d'exactitude : un algorithme qui échoue souvent peut faire en sorte qu'une planification de tâche va également échouer. Ensuite, la métrique de longueur du chemin est importante, car celle-ci est fortement lié au temps d'exécution requis pour suivre ce chemin. L'idéal est d'avoir de courts chemins pour minimiser le temps d'exécution du robot. Enfin, le temps de planification est utile mais puisqu'il est possible de faire la planification hors-ligne, cette métrique sert à évaluer la possibilité de générer des chemins plus courts sans toutefois prendre trop de temps pour la planification. La métrique de la fluidité du chemin donne une information supplémentaire sur la qualité du chemin retourné. En théorie, la trajectoire du robot serait plus fluide plus cette métrique est basse (voir l'annexe B).

Test I

La figure 3.8a montre l'environnement du Test I. Comme configuration de départ, l'embout de la pipette est attaché à la pipette électronique qui est l'effecteur utilisé. Cet embout est entré à l'intérieur du puit de 2 mm. Comme configuration de fin, l'embout de la pipette est au-dessus d'un puit afin de tester un déplacement court et la capacité à éviter le support pour se rendre à la destination finale avec une coupure abrupte entre le support et la plaque de 96 puits. Finalement, cette configuration a été sélectionnée pour montrer les effets de la discrétisation sur la performance et la fiabilité des solutions basées sur la librairie OMPL en présence d'objets étroits (voir l'annexe D).

Taux de réussite. Plusieurs algorithmes ont échoué quelques tests. Ce test est difficile même avec un bon paramètre de discrétisation. Fait intéressant, RRT* a échoué la moitié des tests, et ce même s'il prend la totalité du temps qui lui est alloué pour optimiser la distance la plus courte. RRTConnect, AnytimePathShortening et STOMP ont un taux de réussite de 100 %. CHOMP n'a pas été en mesure de retourner une solution, car il arrête après 200 itérations, soit environ 4 sec. L'hypothèse est qu'il est resté pris dans un minimum local dû à la trajectoire naive de départ qui est une ligne droite et qui est en collision avec la boîte d'embouts à pipette.

Taux d'exactitude. Toutes les solutions réussies retournées par les planificateurs ont un taux d'exactitude de 100 %. Seule exception, AnytimePathShortening n'a pas réussi deux planifications sur les 50 effectués. Il faut comprendre qu'il y a plusieurs étapes de post-traitement à la suite de la planification de mouvement. Le chemin qui était au départ sans collision peut devenir en collision suite à une modification du chemin résultant d'un post-traitement.

Temps total. Les planificateurs RRT*, PRM* et AnytimePathShortening utilisent tout le temps disponible (20 sec) afin d'optimiser le chemin le plus court. RRTConnect

est généralement le plus rapide (médiane 6 sec), suivi de STOMP (médiane 7 sec). CHOMP retourne une erreur parce qu'il n'a pas trouvé de solution (sûrement coincé dans un minimum local).

Longueur du chemin. L'algorithme AnytimePathShortening retourne la longueur de chemin le plus court, suivi de très près de STOMP. Comme AnytimePathShortening, RRT* optimise le chemin le plus court, mais pour cette configuration il a de la difficulté à être aussi bon.

Qualité du chemin. Comme prévu, les algorithmes qui minimisent la longueur du chemin (RRT* et AnytimePathShortening) donnent les chemins de meilleure qualité. Dans cette configuration, STOMP est le pire pour la qualité du chemin retourné, mais reste près des autres planificateurs. Cependant, puisque ces mesures sont obtenues en utilisant les paramètres par défaut, il se peut qu'avec un peu de réglage les performances pourraient être meilleures.

Dans cette configuration spécifique, STOMP serait le premier choix car il a réussi à trouver un chemin sans collision 100 % du temps, des chemins de courte distance et avec un des temps de planification les plus court. AnytimePathShortening et de RRTConnect seraient le deuxième et troisième choix, respectivement.

Test II

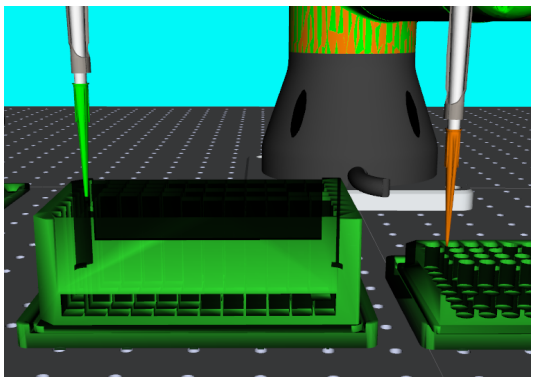
La figure 3.9a montre l'environnement du Test II. Comme configuration de départ, l'embout de la pipette est attaché à la pipette électronique qui est au-dessus d'une cuvette, et la configuration de fin place l'embout au-dessus d'une poubelle prêt à être éjecté. Cette configuration a été choisie afin de voir le comportement des différents planificateurs lors d'un long déplacement (1 m) cartésien avec un bon espacement au niveau des collisions.

Taux de réussite. Tous les algorithmes retournent un chemin dans le temps alloué, excepté pour RRT et RRT* ayant respectivement 88 % et 70 % de succès.

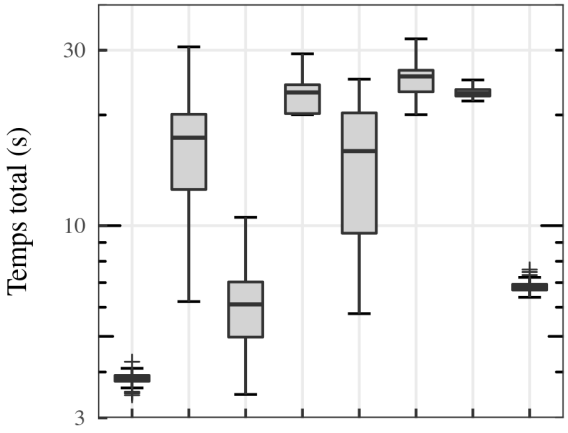
Taux d'exactitude. Les algorithmes ont tous réussi avec succès cette métrique.

Temps total. CHOMP est l'algorithme le plus rapide avec 0.5 sec, suivi de près de STOMP et de RRTConnect. Il est difficile de comparer CHOMP au planificateur de OMPL, car les points de discrétisation semblent être beaucoup plus distants.

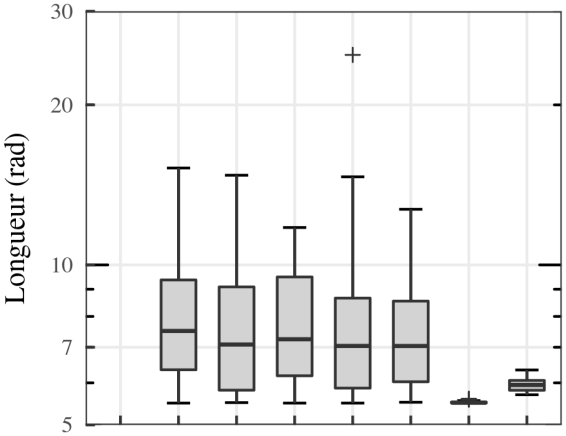
Longueur du chemin. La longueur minimale du chemin planifié est donnée par CHOMP et AnytimePathShortening. Cependant, AnytimePathShortening a quelques échantillons aberrants, mais reste tout de même près de la médiane. RRTConnect et PRM donnent les chemins les plus longs.



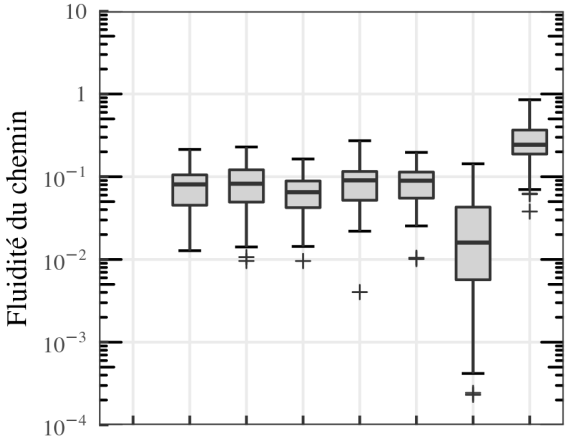
(a) Départ (vert) et fin (orange)



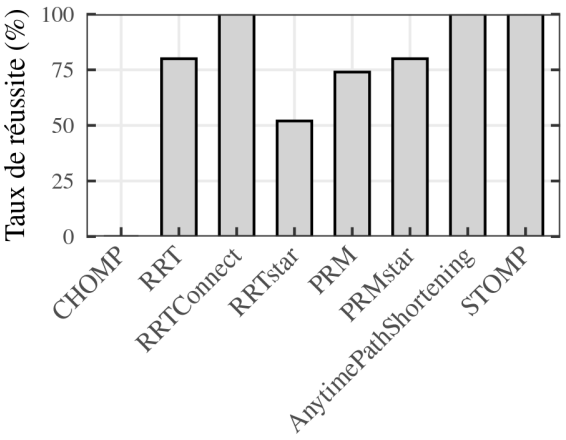
(b) Temps de planification



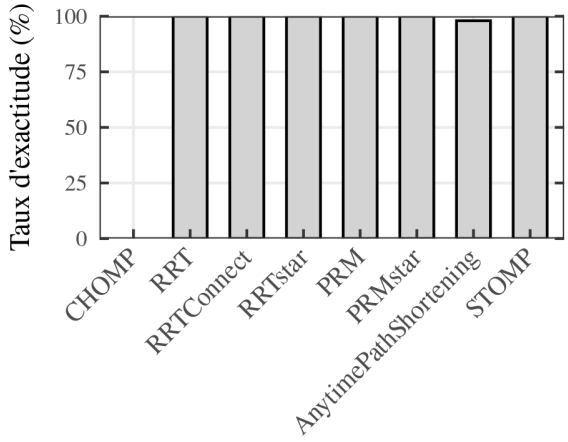
(c) Longueur du chemin



(d) Qualité du chemin



(e) Taux de réussite



(f) Taux d'exactitude

Figure 3.8 Test I : (a) configuration ; (b) l'analyse de planificateur de mouvement sur le temps de planification ; (c) la somme des angles parcourus pour chaque joint ; (d) la fluidité du chemin ; (e) le taux de réussite ; (f) le taux d'exactitude. Les planificateurs sont seulement indiqués pour les figures (e) et (f) sur l'axe des X, mais ont la même représentation pour les figures (b), (c) et (d).

Qualité du chemin. CHOMP donne les chemins les plus fluides suivis de AnytimePathShortening. STOMP donne les chemins les moins fluides, mais reste tout de même près des autres planificateurs.

Au final, CHOMP est choisi pour le Test II car il retourne une bonne solution sans problèmes ainsi que le chemin le plus court, et en plus c'est l'algorithme le plus rapide. STOMP et AnytimePathShortening sont les deux prétendants.

Test III

La figure 3.10a montre l'environnement du Test III. Comme configuration de départ, la pince de préhension tient une cuvette au-dessus de son réceptacle. Comme configuration de fin, la cuvette est au-dessus d'un autre réceptacle pour l'électropolation. Ce test est intéressant car il est possible d'observer le comportement des planificateurs lorsqu'ils doivent planifier près de la base du robot avec plusieurs obstacles aux alentours.

Taux de réussite. L'algorithme RRTConnect, AnytimePathShortening et STOMP ont tous un taux de réussite de 100 %.

Taux d'exactitude. Tous réussisent à atteindre 100 % excepté AnytimePathShortening qui échoue un seul test sur les 50.

Temps total. CHOMP n'est pas considéré car il a échoué toutes ses planifications puisque la scène contient beaucoup de collisions. RRTConnect est donc le plus rapide avec 14 sec. Les autres planificateurs utilisent tous le temps alloué.

Longueur du chemin. AnytimePathShortening donne les résultats les plus courts avec une médiane à 8 rad, suivis de STOMP avec 9.5 rad.

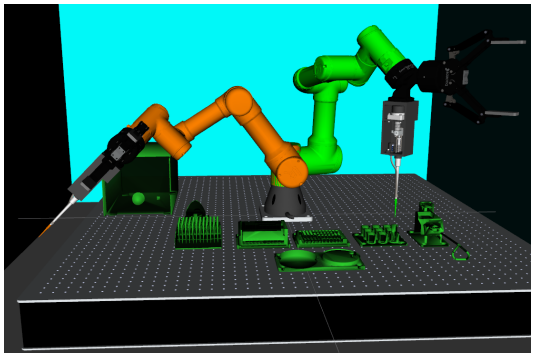
Qualité du chemin. AnytimePathShortening est aussi l'algorithme qui donne le chemin de plus grande qualité. Le pire est l'algorithme STOMP.

Le planificateur sélectionné ici serait AnytimePathShortening car il a un taux de réussite presque parfait et trouve les solutions les plus courtes. Par contre, le temps pris est l'un des plus élevé. STOMP et RRTConnect sont les autres choix plausibles.

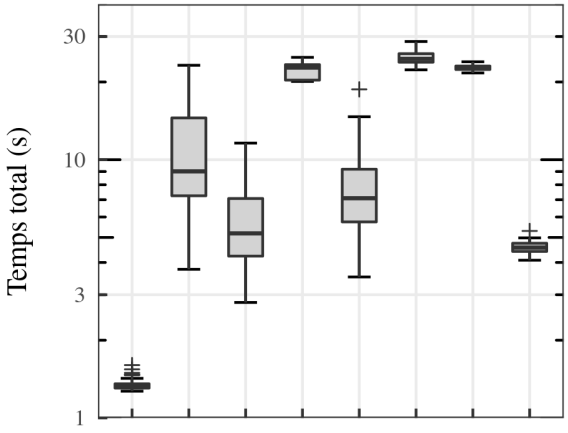
Test IV

La figure 3.11a montre l'environnement du Test IV. Au départ, la pince de préhension tient un stylet au-dessus de son réceptacle, et à la fin le stylet est positionné en face du bouton poussoir du module d'électraporateur. La configuration a été choisie de façon à ce que le robot passe d'un bout à l'autre de la table de travail.

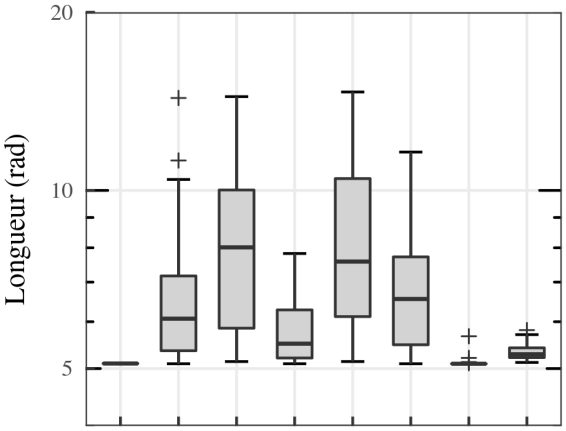
Taux de réussite. Seulement RRTConnect, AnytimePathShortening et STOMP ont un taux de succès parfait.



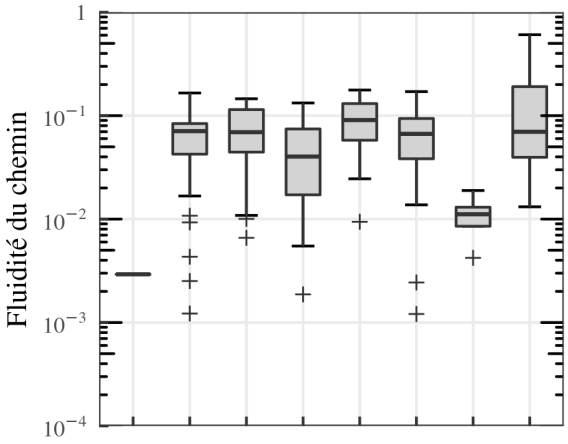
(a) Départ (vert) et fin (orange)



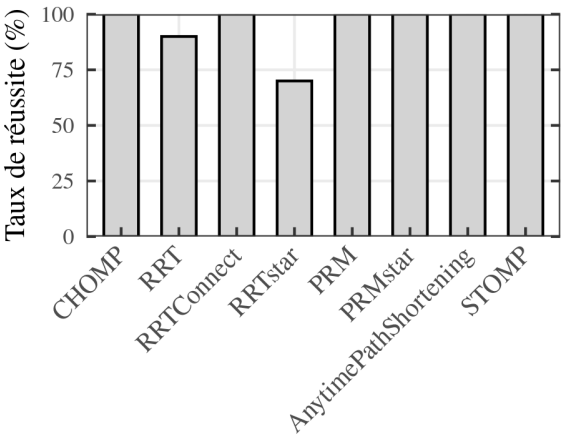
(b) Temps de planification



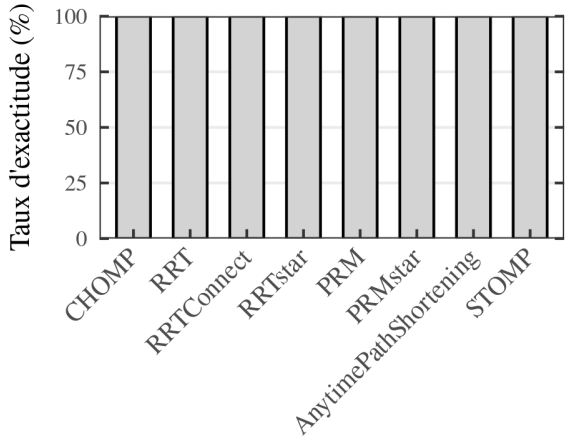
(c) Longueur du chemin



(d) Qualité du chemin

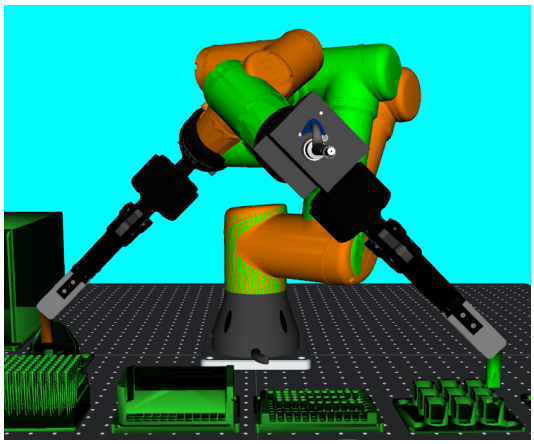


(e) Taux de réussite

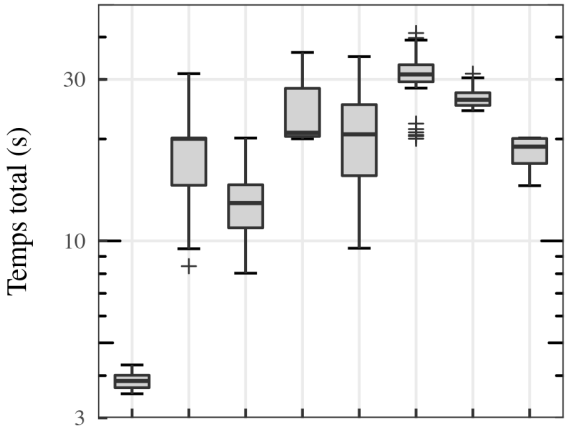


(f) Taux d'exactitude

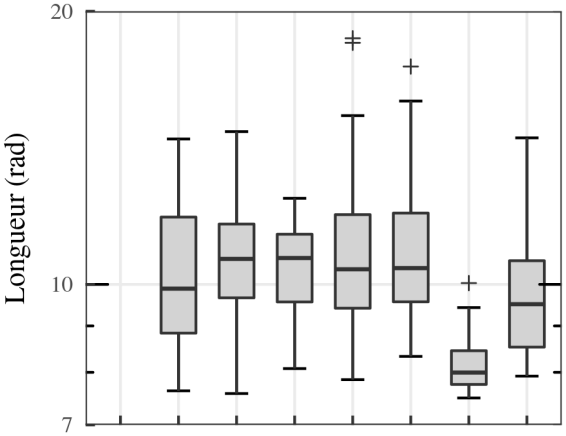
Figure 3.9 Test II : (a) configuration ; (b) l'analyse de planificateur de mouvement sur le temps de planification ; (c) la somme des angles parcourus pour chaque joint ; (d) la fluidité du chemin ; (e) le taux de réussite ; (f) le taux d'exactitude. Les planificateurs sont seulement indiqués pour les figures (e) et (f) sur l'axe des X, mais ont la même représentation pour les figures (b), (c) et (d).



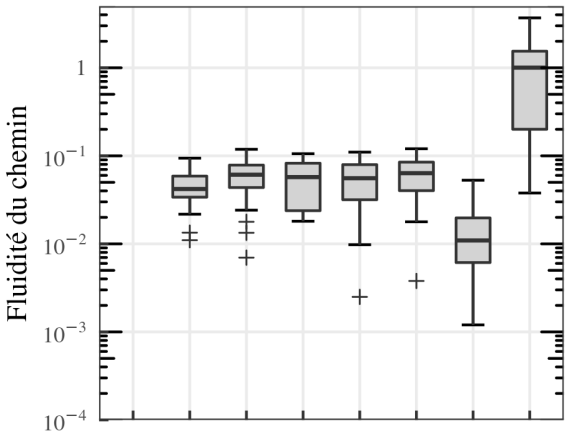
(a) Départ (vert) et fin (orange)



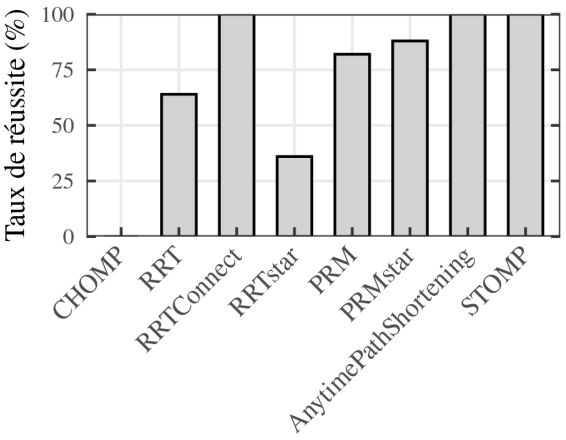
(b) Temps de planification



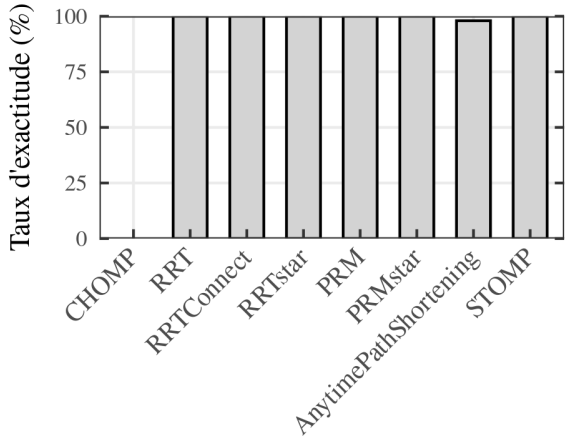
(c) Longueur du chemin



(d) Qualité du chemin



(e) Taux de réussite



(f) Taux d'exactitude

Figure 3.10 Test III : (a) configuration ; (b) l'analyse de planificateur de mouvement sur le temps de planification ; (c) la somme des angles parcourus pour chaque joint ; (d) la fluidité du chemin ; (e) le taux de réussite ; (f) le taux d'exactitude. Les planificateurs sont seulement indiqués pour les figures (e) et (f) sur l'axe des X, mais ont la même représentation pour les figures (b), (c) et (d).

Taux d'exactitude. Réussite à 100 % pour tous les algorithmes.

Temps total. CHOMP est disqualifié car il ne réussit pas à retourner de solution. RRT-Connect donne les solutions le plus rapidement, suivi de STOMP.

Longueur du chemin. Les chemins les plus courts sont retournés respectivement par AnytimePathShortening, STOMP et RRT*.

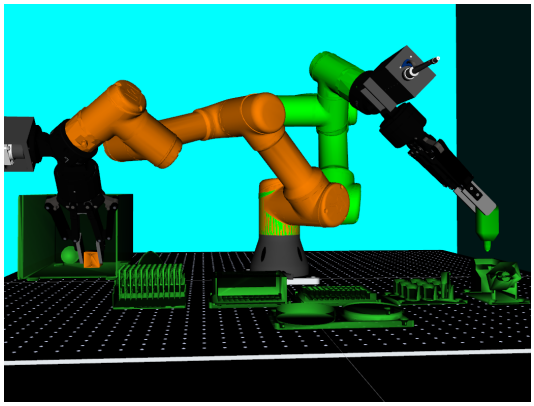
Qualité du chemin. AnytimePathShortening donne les chemins les plus fluides tandis que STOMP donne les chemins les moins fluides.

STOMP serait le premier choix, suvi de très près par AnytimePathShortening avec un taux de réussite et d'exactitude de 100 %, il donne des chemins très courts et est le deuxième plus rapide de tous les planificateurs dans ces conditions.

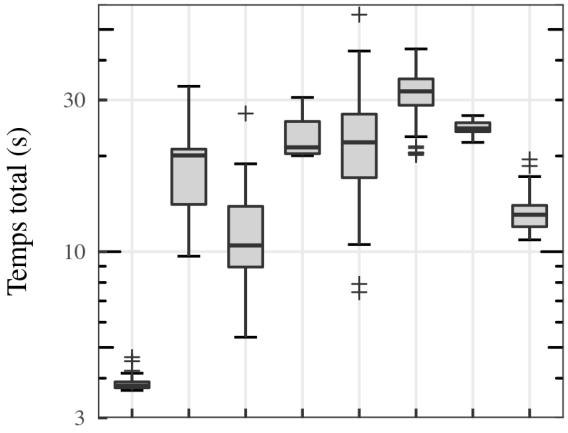
Choix de l'algorithme de planification

Les quatre tests ont été sélectionnés pour représenter le plus possible les configurations typiques rencontrées dans les protocoles biologiques. Ultimement, seulement un planificateur serait utilisé dans toutes les configurations. Les résultats suggèrent que le meilleur planificateur diffère selon les tests, et les choix sont STOMP, CHOMP et AnytimePathShortening. CHOMP démontre qu'il est très rapide et donne des chemins très courts. Cependant, il n'a réussi que pour le Test III et semble rester coincé dans un minimum local pour les autres tests, ce qui concorde avec la littérature [32]. De plus, cet algorithme a beaucoup de paramètres ajustables, ce qui n'est pas viable pour une structure générique de planification de tâche. AnytimePathShortening est toujours dans le top deux ou trois pour tous les tests. Il a un taux de réussite parfait et un taux d'exactitude presque parfait. Sa plus grande force est la petitesse des chemins retournés. Il y parvient car c'est un algorithme qui optimise le chemin le plus court en prenant tout le temps qui lui est permis, ce qui est en contre-partie son défaut. Il excelle aussi dans la fluidité des chemins retournés. STOMP est dans le top deux de tous les tests, dont deux fois premier. Son taux de réussite et d'exactitude est parfait. La longueur des chemins est très près de ceux retournés par AnytimePathShortening. De plus, c'est l'algorithme le plus rapide en produisant des chemins très courts. Par contre, c'est le pire au niveau de la métrique de fluidité de chemin.

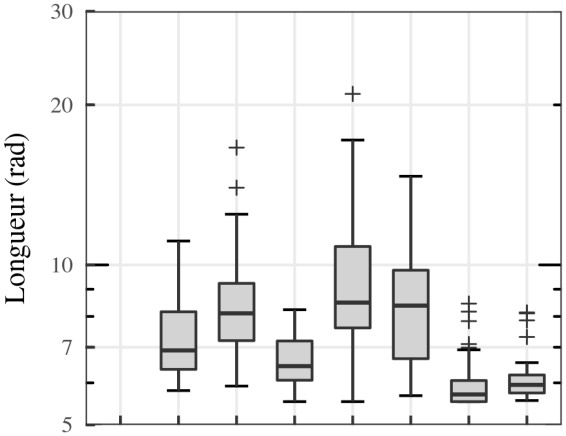
Donc, STOMP semble le choix idéal avec le meilleur compromis avec la longueur des chemins et le temps de planification. Ce choix n'est cependant pas définitif. En effet, la planification de protocole doit planifier des centaines, voir des milliers de trajectoires. Il y a beaucoup plus de type de configuration de robot que les quatre tests analysés ici. Néanmoins, cette étape donne une bonne idée des avantages et désavantages de chacun des types de planificateurs.



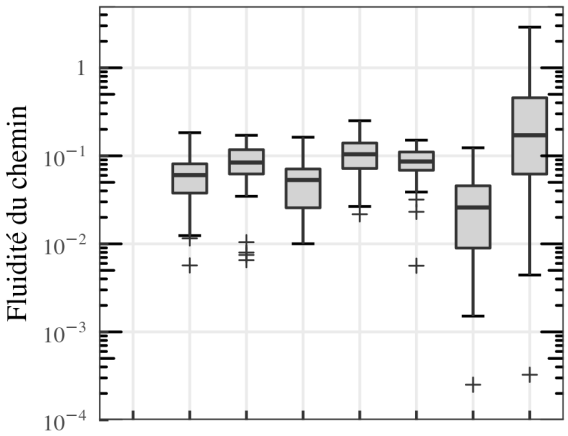
(a) Départ (vert) et fin (orange)



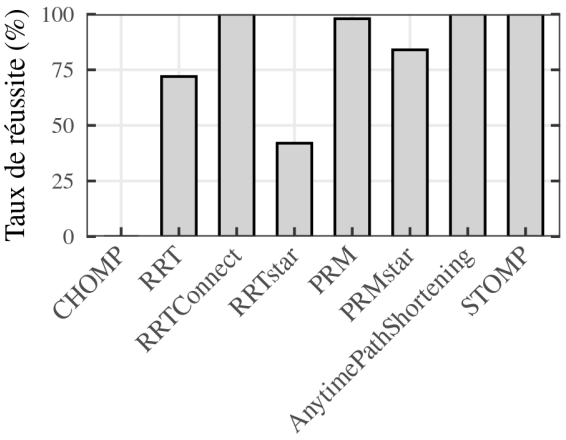
(b) Temps de planification



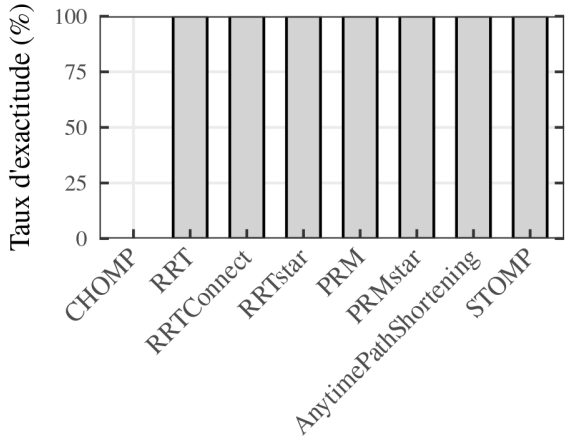
(c) Longueur du chemin



(d) Qualité du chemin



(e) Taux de réussite



(f) Taux d'exactitude

Figure 3.11 Test IV : (a) configuration ; (b) l'analyse de planificateur de mouvement sur le temps de planification ; (c) la somme des angles parcourus pour chaque joint ; (d) la fluidité du chemin ; (e) le taux de réussite ; (f) le taux d'exactitude. Les planificateurs sont seulement indiqués pour les figures (e) et (f) sur l'axe des X, mais ont la même représentation pour les figures (b), (c) et (d).

3.3.2 Planificateur cartésien

MoveIt regroupe plusieurs planificateurs cartésiens [78] : MoveIt Servo, OMPL Constraint Planning, Pilz Industrial Motion, BioIK, Bolt, Descartes et MoveIt Cartesian Interpolator. Le tableau 3.2 présente un sommaire. Les contraintes cartésiennes ne sont pas encore soutenues dans MoveIt pour les algorithmes d'optimisation de trajectoire.

Tableau 3.2 Sommaire des planificateurs cartésiens dans MoveIt [78].

	MoveIt Servo	OMPL Constraint Planning	Pilz Industrial Motion	BioIK (Gradient Descent solver)	Bolt	Descartes	MoveIt Cartesian Interpolator
Complete?	No	Probabilistically	No	Probabilistically	Resolution	Resolution	No
Realtime?	Yes	No	Fairly, Deterministic	Yes	No	No	Somewhat
Plans Ahead?	No* *Can be used offline	Yes	Yes	No	Yes	Yes	Yes
Supports underconstrained?	Yes* *No tolerance, on or off	Yes	No	Yes	Yes	Yes	No
Singularity / JLimit Avoidance?	Singularities Slows down for joint limits	No* You could add it	No	Yes* * via penalty cost	Yes	Yes	No
Minimizes Joint Motion / Jumps	Yes	Not by default	Yes* *joint	Yes* * via minimal displacement weight	Yes	Yes	No
Supports Multiple Intermediate Points	No?	No	Yes	No* * But can solve multiple waypoints while minimizing pairwise distance	Yes	Yes	Yes

Présentement, le planificateur cartésien par défaut MoveIt Cartesian Interpolator est utilisé car c'est le plus simple à utiliser avec MoveIt Task Constructor et certains planificateurs n'étaient pas disponibles lors de l'implémentation du projet. Comme il échoue souvent pour des mouvements avec des contraintes complexes, il serait judicieux d'essayer d'autres algorithmes qui sont *resolute complete* ou *probabillistically complete* pour réduire le taux d'échecs.

3.3.3 Détection de collision

Le robot, la table de travail, les outils et tous les objets sont représentés sous forme de maillage (*Mesh*). C'est une collection de sommets, d'arêtes et de faces organisés en polygones pour représenter un polyhèdre [79]. Les objets sont représentés par un mesh triangulaire de format *Standard Triangle Language* (STL). Pour certains algorithmes de planification, le mesh est utilisé tel quel pour représenter un obstacle. Pour d'autre, celui-ci est converti en une représentation différente qui est dictée par l'algorithme de planification utilisé (voir le tableau 2.3).

MoveIt intègre deux type de détection de collision : *Flexible Collision Library* (FCL) et *Bullet*. FCL est actuellement implémenté pour détecter une collision discrétisée. Même s'il n'y a pas de collisions dans un chemin discret, il se peut qu'une collision ait lieu entre deux points discrets. Pour pallier à ce problème, surtout présent avec des objets de laboratoire, il faut augmenter la résolution de points, mais ceci augmente le temps de calcul pour la planification. Bullet détecte les collisions discrètes et continues (CCD). La détection continue tente de trouver s'il y a collision entre deux états discrétisés [80]. Bullet serait jusqu'à 10 fois plus performant en terme de vitesse [81]. En pratique, FCL est beaucoup plus rapide que Bullet car la collision dans MoveIt n'est pas optimisée pour le multi-processus avec Bullet. Donc, FCL est choisi car il est le plus rapide présentement avec l'intégration dans MoveIt. La librairie FCL permet la détection continue, mais ceci n'est présentement pas intégrée à MoveIt. Il faut donc prendre une résolution adéquate du nombre de points discrétisés selon l'environnement du robot.

3.3.4 Planification de trajectoire

Le chemin retourné par la planification cinématique doit être transformé en une trajectoire. L'algorithme TOPP-RA aurait été choisi pour sa robustesse, mais elle n'est pas incluse dans MoveIt. Au final, TOTG a été choisi même si ce dernier dévie un peu du chemin original. Dans les tests effectués tout au long de ce projet, l'algorithme donnait de bons résultats au niveau de la fluidité de la trajectoire, et ce sans recourir à de la détection de collision supplémentaire.

3.4 Exécution du protocole

Les trajectoires retournées par la planification de mouvement sont acheminées à un contrôleur de mouvement. Pour l'interaction avec l'environnement, un contrôleur en force est utilisé. Donc, deux contrôleurs sont utilisés en alternance selon les situations.

Comme contrôleur de mouvement, le contrôleur pos_joint_traj_controller est sélectionné pour suivre les trajectoires qui sont libres de collision. Les paramètres de vitesse et d'accélération des trajets ont été réduits pour que le robot ne diverge pas de la trajectoire désirée dû aux effets non-linéaires qui ne peuvent être compensés avec les contrôleurs accessibles. Ces effets sont accentués par la configuration et la masse des deux effecteurs : ils sont espacés de 90 degrés avec une longueur de plus de 40 cm, une pesanteur autour de 2 kg sur une capacité maximale (*payload*) de 3 kg.

Comme contrôleur en force, le contrôleur en admittance est choisi afin d'interagir avec les différents éléments dans l'environnement. Ce contrôleur peut être configuré pour être compliant ou rigide dans un ou plusieurs axes. Les différents gains pour le ressort et

l'amortisseur virtuel sont automatiquement choisis en fonction de la vitesse cartésienne maximale dans l'axes où il y a l'interaction. Pour les autres axes, ces gains ont été choisis de façon empirique afin de préserver une certaine compliance. Avant le contact, cette vitesse est automatiquement réduite car la fréquence de la boucle de contrôle du robot et du capteur de force ne permet pas une stabilité : sans cette réduction, le robot oscille dans l'axe d'interaction sans jamais être en mesure d'appliquer la force désirée.

3.5 Planification de tâche

Une tâche est définie comme une séquence d'actions ou de sous-tâches pour atteindre un objectif. Chaque action est ensuite relayée au planificateur de mouvement afin de produire un chemin ou une trajectoire. Voici un exemple d'une tâche de *pick and place* décomposée en une suite d'actions :

- Approcher la pince au-dessus de l'objet.
- Descendre la pince.
- Appréhender l'objet.
- Remonter la pince.
- Déplacer l'objet au-dessus de l'objectif.
- Descendre la pince.
- Relâcher l'objet.

Il n'y a pas encore de solution adoptée universellement pour planifier une telle séquence qui doit interagir avec l'environnement [8, 13]. Les problèmes potentiels dus à la planification de tâches sont les suivants : l'action est impossible à réaliser à cause d'une action précédente ; l'action est impossible à réaliser dûe à un choix de configuration (p.ex. la main gauche échoue alors que la main droite aurait réussi) ; mémoire insuffisante et un temps de planification très long.

Afin de résoudre ces problèmes, Moveit Task Constructor (MTC) est utilisée comme librairie pour planifier une tâche [82]. Elle s'intègre à la plateforme MoveIt et possède une interface graphique pour visualiser chaque action. Ensuite, la faisabilité à planifier une tâche au complet versus une tâche décomposée en plusieurs actions est examinée. Enfin, la disposition initiale des objets a un impact sur la planification de tâche.

3.5.1 *Moveit Task Constructor*

Les tâches sont décomposées de façon hiérarchique avec des couches qui décrivent un problème de planification de mouvement avec une configuration de départ et de fin spécifiées dans l'espace des joints ou cartésiens. Chaque couche tente de se connecter à la couche

suivante en validant si la configuration de fin de la première couche fonctionne avec le départ de la couche suivante.

Il y a trois types de couches qui affectent le flux de résultat : générateur, propagation et connecteur [83]. Le générateur calcule ses résultats indépendamment des couches voisines et achemine ses résultats à la couche supérieure (*backward*) et inférieure (*forward*). Par exemple, la cinématique inverse d'une pose pourrait alimenter le mouvement d'approche et de départ des couches voisines. La couche de propagation reçoit le résultat d'un des voisins, résout son problème et propage son résultat à l'autre voisin. Cette propagation peut être avant, arrière ou dans les deux sens. Les connecteurs tentent de lier deux couches voisines, comme par exemple le calcul d'une planification de mouvement d'un état vers un autre. Ces couches sont dites primitives car ils sont spécifiques à un type de planificateur, comme l'illustre la figure 3.12.

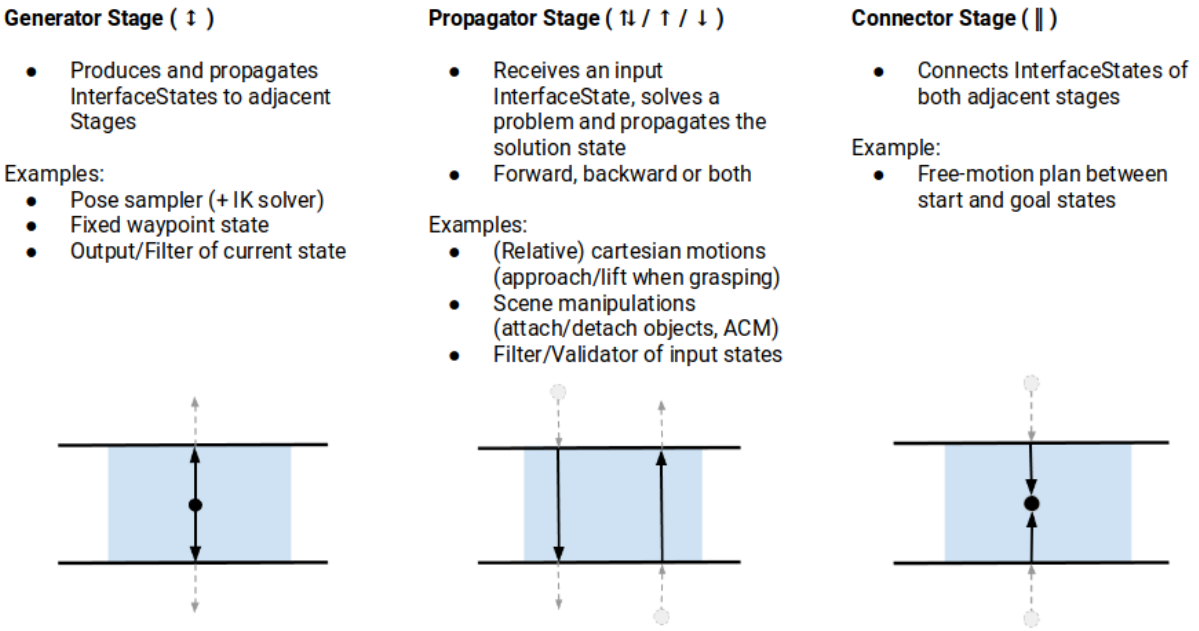


Figure 3.12 Couches primitives de la librairie MTC [83].

Il y a aussi la notion de couches d'encapsulation (*container stages*) qui sont de haut niveau. Une encapsulation sérielle est utilisée pour une séquence d'actions cohérentes afin de considérer si la séquence est valide de bout en bout. L'encapsulation parallèle est utilisée pour passer la meilleure réponse de couches alternatives, pour avoir une planification de secours ou pour fusionner différentes solutions indépendantes. Voici quelques exemples : exécuter plusieurs planificateurs de mouvement différents et de prendre la meilleure solution ; prendre un objet avec la main droite et comme solution de secours la main gauche ; ou bien de bouger le bras robotique et la pince de préhension en même temps [83].

3.5.2 Planifications globale et locale

La librairie MTC énumère exhaustivement toutes les solutions possibles pour réussir une tâche. Plus une tâche contient de sous-tâches ou d'actions, plus le nombre de solutions augmente exponentiellement (*combinatorial explosion*) [82]. Afin de contourner le problème, MTC utilise des heuristiques de planification pour prioriser la recherche d'une solution à travers les meilleurs candidats. Quelques exemples d'heuristiques sont la distance parcourue des joints et la durée de la trajectoire.

Même si une heuristique est utilisée, une tâche avec beaucoup de sous-tâches va finir par prendre une infinité de temps. Par conséquent, la mémoire sera insuffisante pour garder tous les trajets et solutions des différents couches. Au lieu planifier globalement toutes les tâches, il est possible de planifier localement, c'est-à-dire de planifier pour quelques sous-tâches seulement et de prendre la meilleure solution pour alimenter la prochaine séquence. Cela réduit de beaucoup le temps de calcul car des solutions potentielles sont enlevées. Par contre, il faut choisir de façon intelligente les sous-tâches à planifier, sinon un système de *backtracking* doit être utilisé. En effet, il serait possible de se retrouver dans une situation où il n'y aurait pas de solution viable due au point de départ qui résulte de la planification précédente.

3.6 Planification de protocole

Un protocole est une combinaison d'actions réalisées par le robot, en utilisant des outils dans le but d'interagir avec les objets dans la cellule robotique. C'est un niveau d'abstraction supplémentaire qui alimente le planificateur de tâche. Le but est de générer une séquence de sous-tâches en fonction du protocole à exécuter.

Les prochaines sous-sections expliquent le processus. Tout d'abord, il faut caractériser les objets de la table de travail, caractériser les outils (p.ex. pince de préhension) qui permettent certaines actions. Une représentation de l'environnement de la plateforme, aussi appelée scène robotique, est nécessaire afin de suivre l'évolution de la plateforme dans le temps. Ensuite, la structure du protocole est décrite. Après, la séquence de planification est expliquée pour un protocole simple. Puis, quelques pistes sont dévoilées pour optimiser la présente solution. Finalement, une énumération de la détection d'erreur est produite.

3.6.1 Caractériser les objets

A priori, il faut caractériser les objets pour pouvoir les manipuler ou les faire fonctionner. Le mesh est requis pour ne pas entrer en collision avec celui-ci lors de la planification de mouvement. Ensuite, il faut ajouter, directement dans le code, les éléments de l'objet. Un élément est en fait un point d'intérêt comme un point d'insertion, un récipient pour

ajouter des liquides ou d'autres objets, un bouton poussoir, un écran, une surface, etc. C'est un endroit pouvant avoir une certaine interaction. Plusieurs données géométriques et de statiques sont requises telles la longueur, la surface, la pose, le volume, le centre de masse et la force.

Ce processus est exhaustif si le *mesh* n'est pas disponible pour l'objet et si l'objet est complexe. Par exemple, s'il y a dix boutons poussoirs à caractériser, il faut enregistrer ces informations : la pose du bouton par rapport à un référentiel connu, la forme du bouton, la force minimale requise pour l'enfoncer ainsi que la distance d'enfoncement. Par contre, il n'a besoin que d'être fait une seule fois et est réutilisable pour des objets identiques. Comme les éléments sont des caractéristiques physiques, ils peuvent être facilement réutilisés dans un autre cadre logiciel. Le but de cette caractérisation est de permettre au planificateur de choisir le meilleur robot/outil disponible pour interagir avec cet élément.

3.6.2 Caractériser les outils

L'outil est un objet auquel certaines actions sont ajoutées. Il peut être attaché sur un effecteur du robot ou être disponible sur la table de travail et va interagir avec un élément d'un autre objet. Les actions disponibles sont : l'aspiration et la dispense de liquide, la préhension parallèle et la relâche d'un objet, l'application d'une force sur un objet, l'insertion d'un objet dans un autre, activer un bouton poussoir (capacitif ou résistif) ainsi que d'étaler sur une surface.

3.6.3 Scène robotique et protocole

La scène robotique est une représentation de toute l'information nécessaire pour représenter l'état de la plateforme robotique à un instant donné : le robot, les effecteurs, les contrôleurs, les objets, les outils et les liquides.

La scène comprend donc le robot UR3 avec ses deux effecteurs : 1) la pince de préhension, et 2) la pipette. Le premier effecteur peut être bougé par deux contrôleurs propres au UR3 : en position et en vitesse. Même chose pour la pipette électronique. L'ajout d'un senseur de force et de torque donne aux deux effecteurs la possibilité d'être contrôlé en admittance.

Les objets ont tous un lien parent et enfant (structure arborescente) qui les unis. Par exemple, la table de travail est le parent d'un support pour rack de tip, celui-ci est le parent du rack de tip et les tips sont les enfants du rack.

Une trace est préservée des différents liquides (volume + type) qui peuvent être dans un élément de type conteneur. Un conteneur n'est pas seulement limité à des liquides, il peut

aussi inclure des objets. Par exemple, un embout peut être déposé dans un conteneur. Comme il n'y a pas d'insertion fixe entre les deux objets, la pose exacte de l'objet est alors perdue. Il faudrait alors un système de vision pour aller le chercher.

Le tableau 3.3 montre la liste des actions possibles pour générer un protocole. Ces actions dépendent des outils disponibles sur la scène robotique de départ. L'usager doit choisir une action et l'outil qui permet cette action. Ensuite, il faut ajouter avec quels objets et éléments cette action interagit. Les actions sont ajoutées de façon hiérarchique afin de construire le protocole désiré. Des paramètres globaux et par type d'action peuvent également être ajoutés pour remplacer les options par défaut.

3.6.4 Planification de protocole

La planification de protocole repose sur la caractérisation des objets et des outils, de la scène robotique ainsi que la description du protocole. De plus, chaque action est transformée en différentes couches pour alimenter le planificateur de tâches. Voici une brève description du flux de planification pour un protocole biologique simple, qui consiste à aspirer une quantité d'eau avec la pipette électronique dans un certain puit. Ceci implique qu'il faut, au préalable, insérer un embout à la pipette. Ensuite, il faut dispenser ce liquide dans un autre puit. Enfin, il faut disposer de l'embout dans une poubelle.

Avant toute chose, il faut ajouter les objets, les outils, le robot, les effecteurs et les contrôleurs à la scène de départ. De plus, les liquides doivent être ajoutés à la scène : leur volume et dans quel conteneur. Une fois que la scène est populée, le protocole est construit à l'aide des actions disponibles. Le tableau 3.4 montre ce protocole avec les paramètres requis. La planification peut ensuite débuter. Une validation logiciel est effectuée pour voir si les objets, les éléments et les outils sont disponibles dans la scène de départ et dans les scènes futures. Par exemple, il est vérifié s'il y a assez de liquide pour exécuter l'action d'aspiration ou si le réceptacle peut recevoir un tel volume. La validation est très rapide et se fait pour toutes les actions du protocole. Si la validation réussie, les actions du protocole

Tableau 3.3 Sommaire des actions d'un protocole.

Action	Outil	Source		**Élément**	Destination		**Élément**
aspirer	pipette	tip0	⇒	**Ctn0**	puit1	⇒	**Ctn1**
dispenser	pipette	tip1	⇒	**Ctn1**	puit1	⇒	**Ctn2**
saisir	pince	pince	⇒	**PP0**	rack	⇒	**PP3**
détacher	pince	puit1	⇒	**PP1**	support	⇒	**In0**
insérer	pipette	pipette	⇒	**In0**	tip2	⇒	**In1**
peser	stylet	stylet	⇒	**Ac0**	centrifugeuse	⇒	**Btn2**

sont transformées en couches hiérarchiques afin d'alimenter le planificateur de tâche. La librairie MTC effectue la planification de ces différentes couches et retourne de multiples solutions avec une fonction de coût associée à chacune d'entre elle. Cette planification peut être visualisée en simulation afin de voir les différentes solutions proposées. La solution est ensuite enregistrée afin d'être exécuté sur le vrai robot.

Présentement, l'exécution d'un protocole enregistré se fait via un terminal. Lors de l'exécution d'un protocole pré-enregistré, il faut suivre les étapes suivantes. Premièrement, il faut sélectionner le protocole souhaité via l'interface graphique. Ensuite, une représentation visuelle est donnée pour ajouter les différents objets et outils à l'environnement. Après, le robot doit être déplacé à la configuration de départ afin de pouvoir lancer l'exécution du protocole.

3.6.5 Optimisation

Présentement, le regroupement des actions pour créer une tâche est fait manuellement. Pour avoir de bons résultats, il faut planifier certaines actions ensembles. Par exemple, pour prendre un objet (*grasp*) et l'insérer (*insert*) dans un nouvel endroit, il faut planifier ensemble ces deux actions, sinon il serait possible de trouver une solution localement pour la préhension de l'objet mais l'insertion pourrait échouer selon l'endroit utilisé pour la préhension de l'objet.

Il n'y présentement pas de système-expert au niveau du protocole, c'est-à-dire des règles logiques (heuristique) se basant sur des connaissances *a priori* pour déduire de nouvelle information (inférence). Un tel mécanisme pourrait réduire le nombre de paramètres à ajouter lors de la création de protocoles et même enlever certaines actions. Par exemple, pour aspirer un liquide avec la pipette électronique, il faut nécessairement un embout. Cela simplifierait grandement la construction d'un protocole. Cependant, certaines heuristiques peuvent être difficiles à concevoir. En effet, comment déterminer le meilleur outil à utiliser pour la préhension d'un objet entre une pince de préhension et un outil à suction ?

Tableau 3.4 Protocole simple représenté sous forme d'actions hiérarchisées.

Action	Outil	Source		**Élément**	Destination		**Élément**
insérer	pipette	pipette	⇒	**In0**	tip2	⇒	**In1**
aspirer	pipette	tip0	⇒	**Ctn0**	puit1	⇒	**Ctn1**
dispenser	pipette	tip1	⇒	**Ctn1**	puit1	⇒	**Ctn2**
éjecter	pipette	tip1	⇒	**Ctn1**	poubelle	⇒	**Ctn0**

3.6.6 Détection d'erreurs

Il n'y a présentement pas de système de vision sur la plateforme. Ceci dit, il y a beaucoup d'informations disponibles pour valider ou non si le protocole biologique a bien été complété. Lors de l'exécution, plusieurs choses peuvent se passer (oubli d'un objet, collision, etc).

Il est souhaitable de réduire le plus possible les faux positifs et les faux négatifs. Pour ce faire, lorsque le robot exécute une trajectoire, cette trajectoire est suivie avec un paramètre permissif en position, sans quoi le robot arrête et le protocole est interrompu. Si il y a une collision non désirée, le robot arrête avec une certaine marge de force, mais il pourrait avoir endommagé de l'équipement fragile ou lui-même. Le contrôleur en admittance a beaucoup de paramètres pour assurer une exécution sécuritaire : filtration, limite en force cartésienne, détection de singularité, limite en vitesse (domaine des joints et cartésien), limite en position cartésienne avec une certaine marge de sécurité. Par exemple, le contrôleur arrête si la pose pour l'insertion vient à dépasser une certaine distance due à un objet manquant. Donc, plusieurs mesures de sécurité dans le contrôleur du robot UR3 et dans les contrôleurs de mouvements et de force sont utilisées dans notre montage.

CHAPITRE 4
RÉSULTATS

Les sections suivantes portent sur les résultats obtenus lors de la planification et de l'exécution réelle par le robot de deux protocoles biologiques. La scène robotique du protocole I et II est donné respectivement à la figure 4.1 et 4.2. Deux vidéos sont disponibles : la première [1] montre concrètement la planification du protocole biologique II ; la deuxième [2] montre l'exécution du protocole II sur la plateforme robotique.

4.1 Protocole biologique I : prélèvement d'échantillons pour des études de cinétique de croissance bactérienne

De nombreuses expériences en biologie visent à déterminer la vitesse de réactions enzymatiques ou le temps de doublement de cultures cellulaires. Ce type d'information nécessite de diluer de manière précise les enzymes ou les cellules afin de quantifier plus précisément les valeurs recherchées. Une manière simple d'y arriver utilise les dilutions en série. Une fraction d'un échantillon de départ est ainsi mélangé avec un solvant dans des proportions prédéterminées. Cette opération est répétée autant de fois que nécessaire, généralement à l'aide d'une pipette et d'une plaque de 96 puits.

Ce protocole simple est utilisé comme référence pour comparer la solution proposée (robot articulé) dans ce projet aux solutions existantes (robot cartésien) comme le robot OT2 et Hamilton Nimbus. C'est une tâche répétitive qui peut s'échelonner sur de longues périodes. La comparaison débute avec la planification de mouvement, suivie de l'intégration des modules et de la vitesse d'exécution pour ce protocole. Enfin, un verdict est donné pour déterminer quelle solution serait à privilégier.

Planification. Il est beaucoup plus simple de planifier avec un robot cartésien (trois axes de translation) car il n'y qu'une seule configuration possible dans le domaine des joints pour une pose désirée dans le domaine cartésien, tandis que pour un robot articulé il y a plusieurs poses possibles pour une même destination. Du même coup, la contrainte pour que les fluides ne se déversent pas sur la table de travail

1. `https://drive.google.com/file/d/1pXSL083qCg6t2EC-Vt3UR3aiKZIdqwja/view?usp=sharing`
2. `https://drive.google.com/file/d/1caUC2tY3fhG8hdMYIDHmY4ZkWapOmGlK/view?usp=sharing`

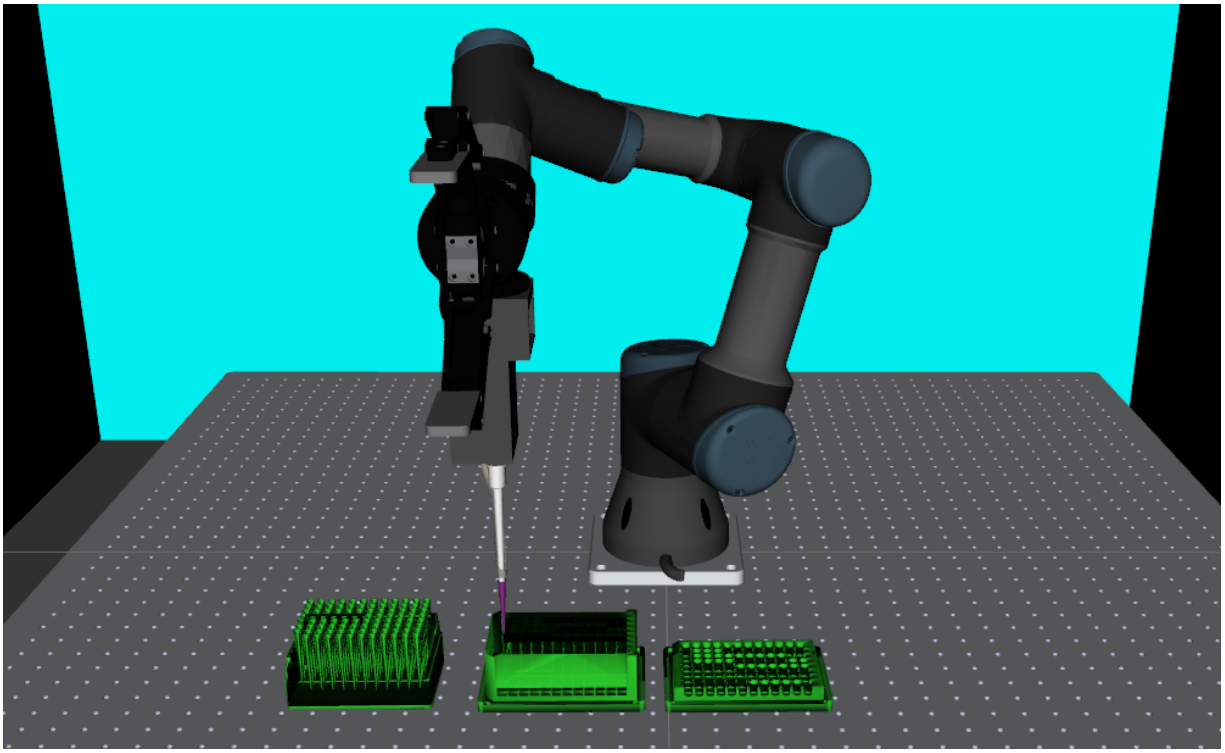


Figure 4.1 Scène robotique du protocole I.

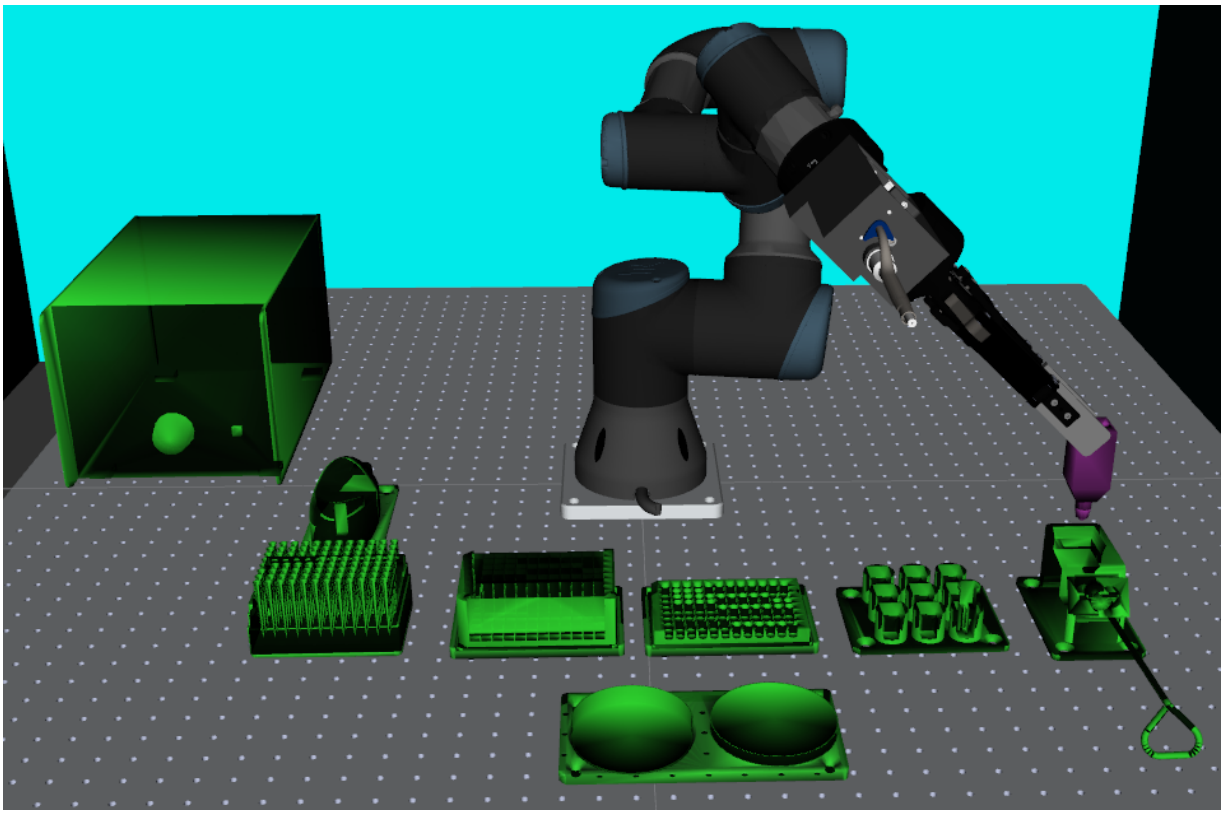


Figure 4.2 Scène robotique du protocole II.

doit être prise en considération pour la planification de robots articulés. Les robots cartésiens ont par défaut cette contrainte lors de la planification. La détection de collisions peut être relaxée pour les solutions existantes puisqu'il suffit de respecter une certaine hauteur et le tour est joué. Ceci est par contre beaucoup plus difficile pour un robot articulé avec 6 DDL. Avec le UR3, cela prend environ 1000 secondes pour planifier la séquence complète de mouvements pour exécuter le protocole 16 fois avec l'algorithme RRTConnect. En termes de temps de planification, il est supposé que les solutions existantes sont beaucoup plus rapides avec les robots cartésiens que pour la solution proposée utilisant un bras articulé.

Modules. Au niveau des modules, Opentrons permet de prendre des boîtes d'embouts génériques. Par contre, pour les modules actifs, les utilisateurs doivent s'adapter à ceux offerts par cette compagnie. Avec le Hamilton Nimbus, des modules dédiés doivent être utilisés sur la plateforme robotique. Notre système robotique permet de caractériser n'importe quel module et de l'intégrer. L'avantage est d'avoir une plus grande modularité, donc un plus grand nombre de choix à la disposition pour les différents protocoles. Il est aussi possible de prendre des équipements déjà existant et qui sont moins coûteux. Refaire des modules aurait été trop long et n'aurait peut-être pas permis d'avoir les mêmes spécifications qu'un module déjà accessible dans le laboratoire. Par contre, au niveau de l'exécution, il est beaucoup plus rapide d'avoir des modules faits sur mesure car il est possible de les contrôler électroniquement plutôt que de les utiliser comme un humain le ferait.

Vitesse d'éxecution. L'analyse se fait avec le protocole de la section 3.6.4 au lieu de celui mentionné ci-haut afin de pouvoir se comparer avec le robot OT2. Les deux protocoles sont très similaires et ne devraient pas donner une analyse biaisée. Le tableau 4.1 montre les temps d'exécution pour les deux robots. Deux méthodes de remplissage sont comparées. La première méthode dispense 200 µL dans 96 puits avec le même embout. Notre solution robotique avec l'UR3 s'avère plus rapide d'environ 30 secondes avec l'algorithme RRTConnect. La deuxième méthode consiste à dispenser 200 µL dans chacun des puits, et ce avec un nouvel embout à chaque fois. Notre solution robotique UR3 est plus rapide de 6 minutes versus le robot OT2. La disposition des modules (puits, rack et poubelle) n'a pas été optimisée pour le test. La planification a été faite par RRTConnect qui est très rapide, mais pour minimiser la longueur des trajectoires, d'autres algorithmes sont mieux adaptés. Ceci aurait pu réduire davantage le temps d'exécution. Les robots plus dispendieux comme le Microlab NIMBUS ou le Cavro Omni Flex seraient potentiellement plus rapides que notre solution robotique.

Pour ce type de protocole, les solutions existantes comme OT2 seraient à privilégier car elles sont plus abordables d'un facteur allant de deux à cinq fois et le temps de planification est beaucoup plus rapide. Les robots cartésiens ont une structure ayant une contrainte naturelle pour ne pas déverser de liquide accidentellement. L'avantage de la solution poroposée est la vitesse d'exécution comparativement au robot OT2. Par contre, les autres solutions plus coûteuses comme celles proposées par Hamilton et par Tecan devraient être plus rapides.

Tableau 4.1 Temps d'exécution pour remplir 96 puits avec OT2 et UR3.

Robot	**Embout unique**	**Embouts multiples**	**Pipette**
OT2	4:01	41:31	P300 simple
UR3	3:33	35:26	P200 simple

4.2 Protocole biologique II : transformation bactérienne par électroporation

La biologie synthétique utilise les informations contenues dans le matériel génétique pour programmer des cellules afin qu'elles excécutent des actions spécifiques. Il est donc courant de devoir insérer des molécules d'ADN dans des cellules d'intérêt. Une méthode très répandue pour y arriver est l'électroporation. Ce protocole consiste à mélanger un échantillon d'ADN avec des cellules avant de les transférer dans une cuvette spécialisée. Un choc électrique de haut voltage (variant généralement entre 1 et 3 kV) est ensuite utilisé pour produire de manière transitoire des pores dans la membrane, ce qui permet le passage des molécules d'ADN de l'extérieur vers l'intérieur des cellules. L'échantillon contenant les cellules est alors dilué dans du milieu nutritif, incubé pendant environ une heure, puis étalé sur une gélose afin de permettre la croissance de colonies isolées les unes des autres.

Ce protocole complexe nécessite plus de 15 objets différents, requiert des actions d'aspiration et de dispense de liquide, de multiples insertions entre différents objets, de peser sur des boutons poussoirs, d'ouvrir des couvercles et d'étaler une goutte sur un gélose. Le tableau 4.2 résume les étapes pour réussir le protocole complexe.

Planification. Le temps de planification pour la solution proposée peut être très long. Pour donner une estimation de temps, cela prend environ 113 secondes pour planifier ce protocole d'un bout à l'autre avec l'algorithme RRTConnect. Avec d'autres algorithmes qui donnent des trajectoires plus courtes, cela peut prendre des heures dépendamment du nombre de points minimaux pour la discrétisation. Si ce protocole doit être répété une centaine de fois (dans des puits différents), la planification

Tableau 4.2 Protocole biologique II : sommaire des actions.

Action	Outil	Source		**Élément**	Destination		**Élément**
insérer	pipette	pipette	⇒	**In0**	tip0	⇒	**In0**
aspirer	pipette	tip0	⇒	**Ctn0**	puit_2ml	⇒	**CtnG11**
dispenser	pipette	tip0	⇒	**Ctn0**	cuvette	⇒	**Ctn0**
éjecter	pipette	tip0	⇒	**Ctn0**	poubelle	⇒	**Ctn0**
saisir	pince	pince	⇒	**PP0**	cuvette	⇒	**PP0**
insérer	pince	cuvette	⇒	**In0**	shockpod	⇒	**In0**
saisir	pince	pince	⇒	**PP0**	stylet	⇒	**PP0**
peser	pince	stylet	⇒	**Ca0**	transporateur	⇒	**Btn0**
insérer	pince	stylet	⇒	**In1**	support_outil	⇒	**In0**
saisir	pince	pince	⇒	**PP0**	cuvette	⇒	**PP0**
insérer	pince	cuvette	⇒	**In1**	support_cuvette	⇒	**InC1**
insérer	pipette	pipette	⇒	**In0**	tip1	⇒	**In0**
aspirer	pipette	tip1	⇒	**Ctn0**	cuvette	⇒	**Ctn0**
dispenser	pipette	tip1	⇒	**Ctn0**	puit_200ul	⇒	**CtnH02**
éjecter	pipette	tip1	⇒	**Ctn0**	poubelle	⇒	**Ctn0**
insérer	pipette	pipette	⇒	**In0**	tip2	⇒	**In0**
aspirer	pipette	tip2	⇒	**Ctn0**	puit_200ul	⇒	**CtnH02**
dispenser	pipette	tip2	⇒	**Ctn0**	petri	⇒	**Ctn0**
éjecter	pipette	tip2	⇒	**Ctn0**	poubelle	⇒	**Ctn0**
saisir	pince	pince	⇒	**PP0**	étaleur	⇒	**PP0**
étaler	étaleur	étaleur	⇒	**Ar**	petri	⇒	**Ctn0**
insérer	pince	étaleur	⇒	**In0**	support_outil	⇒	**In1**

sera non seulement longue, mais elle va échouer due à une explosion combinatoire, c'est-à-dire une explosion du nombre de possibilités, ce qui va utiliser toute la mémoire disponible. Il a été proposé de réduire le nombre de possibilités en planifiant chaque protocole de façon successive. De plus, la planification se base sur une heuristique pour privilégier l'ordre de planification. Un des points les plus difficiles est de donner le plus de chances à la planification de protocole (p.ex. en donnant des poses multiples pour la préhension) sans pour autant la submerger d'informations, ce qui rallongerait encore plus le temps de planification. Il y a d'autres algorithmes de planification prometteurs comme TrajOpt qui pourraient accélérer de beaucoup le temps de planification. De plus, cet algorithme permet d'intégrer des fonctions de coûts pour avoir des chemins qui ont un dégagement minimal. Présentement, certaines trajectoires passent près des obstacles. Cela est dû à certains algorithmes utilisés qui optimisent le chemin le plus court.

Pour faciliter l'expérience, il a été convenu de faire la planification hors ligne pour avoir des trajets les plus courts. L'utilisateur peut donc choisir un protocole préenregistré dans la base de données et l'exécuter instantanément afin de reproduire l'expérience des solutions disponibles sur le marché. Cela apporte son lot de désavantages. Par exemple, l'utilisateur doit replacer exactement tous les objets sur la table de travail ainsi que la position de départ du robot. Une interface graphique est en développement pour faciliter ces ajouts.

Modules. Il n'y a pas de module d'électroporation performant d'intégré aux solutions commerciales. La solution UR3 permet d'ajouter le module d'électroporation souhaité en utilisant un instrument conçu pour une application manuelle. Ceci à l'avantage de sauver des coûts et de s'assurer que le module donne de bons résultats. S'il n'est pas utilisé pour une séquence, le module peut simplement être réutilisé normalement de façon manuelle et réintégré par la suite. Cette flexibilité est possible grâce à la caractérisation faite *a priori* du module. Celle-ci peut être ardue pour des modules complexes, mais n'est requise qu'une seule fois.

Vitesse d'exécution. La solution proposée doit être contrainte à 80 % de sa vitesse maximale et de 50 % en accélération afin de respecter les trajets désirés. Le contrôleur de mouvement utilisé ne compense pas les non-linéarités du robot et des deux effecteurs attachés au robot. La plateforme exécute le protocole en 2 minutes et 30 secondes. Un chercheur expérimenté serait potentiellement plus rapide de 20%. Même si la plateforme est moins rapide, le robot peut répéter un protocole indéfiniment, sans commettre d'erreur, peu importe le moment de la journée. La plateforme

réussit tout de même à être suffisamment rapide pour la plupart des protocoles en biologie synthétique.

Pour être pleinement fonctionnel, le protocole nécessite plusieurs échantillons. Les actions décrites dans la section 4.2 ne sont valides que pour un échantillon. De plus, une action supplémentaire serait requise pour désinfecter l'étaleur. Ceci pourrait être fait en trempant l'étaleur dans de l'éthanol. Aussi, il manque l'action d'enlever et de remettre le couvercle sur la boîte de pétri. Il suffirait de caractériser le couvercle et d'ajouter ces actions dans le protocole. Enfin, l'action d'ajouter et d'enlever un couvercle sur la cuvette est nécessaire pour éviter l'évaporation et la contamination. Cette action est plus difficile à réaliser, car le robot doit tourner le couvercle tout en s'assurant que la cuvette reste dans le support en plastique. Il aurait été intéressant de réaliser le protocole avec de vrais échantillons biologiques, mais il restait encore un peu de travail pour y arriver.

CHAPITRE 5

CONCLUSION

La majorité des plateformes robotiques disponibles sur le marché sont capables d'effectuer seulement quelques actions de pipettage formant des protocoles simples. La plupart de celles-ci nécessitent ensuite des modules spécialisés très dispendieux pour toutes autres tâches de complexité plus élevée. De plus, certains instruments ne sont tout simplement pas disponibles pour des plateformes robotiques. Par exemple, il n'y a présentement aucun module d'électropration ou de centrifugeuse performants sur les plateformes robotiques commerciales. Si un protocole demande un module qui n'est pas intégré à la plateforme, une intervention humaine est nécessaire. Cette intervention est pénible pour le chercheur, surtout lorsque qu'une manipulation doit être exécutée à chaque deux ou trois heures pendant quelques jours.

La présente recherche tente d'éliminer cette intervention humaine en permettant l'ajout de pratiquement n'importe quel appareil ou module à une plateforme robotique. Dans le domaine du possible, le robot utilise directement, ou avec le moins de modifications possibles, les appareils qui sont normalement employés par un humain et non une machine. Pour ce faire, nous devons laisser de côté les robots de type cartésien pour ajouter des degrés de liberté supplémentaires permettant une multitude d'orientations. Ceci est un des facteurs limitant des robots cartésiens pour des modules qui requièrent une interaction avec un certain angle et non par le dessus. Le robot articulé UR3 a été sélectionné pour offrir cette flexibilité. L'exécution du robot est divisée en deux parties : un contrôle de mouvement pour tous les trajets qui sont libres de collision, et un contrôle en admittance pour ceux où il y a un contact désiré avec un objet. Cette dernière option est exécutée en temps réel car il est difficile de prévoir avec précision où le contact se fera dû aux incertitudes en position. Cette flexibilité a cependant un coût : la planification est beaucoup plus compliquée et nécessite plus de temps afin de trouver une solution. De plus, des algorithmes de détection de collisions sont requis. Comme pour les solutions existantes, un système de vision pour la reconnaissance et la localisation a été laissé de côté afin d'avoir une solution plus robuste puisque notre environnement est structuré. Chaque objets ou modules doivent alors avoir son propre support pour être placés sur la table de travail. Aussi, chaque objet doit être caractérisé afin d'éviter les collisions et de permettre à une personne n'ayant aucune connaissance en robotique d'ajouter des actions pour générer un protocole. Pour alléger

ce processus, l'aspect collaboratif du robot pourrait servir davantage. Il serait intéressant d'avoir une interface graphique pour automatiser cette caractérisation à l'aide d'effecteurs spécialement conçus pour caractériser les différents éléments et même générer un maillage grossier de l'appareil. Cette caractérisation serait nécessaire qu'une seule fois. Une fois complétée, elle pourrait être partagée à d'autres chercheurs afin de construire une base de données contenant une multitude d'objets et d'outils qui pourraient être utilisés directement pour la planification d'un protocole. Une interface graphique est en processus de conception afin de faciliter l'exécution de protocoles biologiques.

Deux protocoles biologiques ont été effectués en simulation et sur le robot réel. Le premier est une tâche répétitive qui s'échelonne sur plus de 30 minutes. La solution proposée est en mesure d'exécuter le protocole en entier, et ce sans échec. Le deuxième est un protocole complexe qui requiert plusieurs interactions avec des modules d'électroporation et des outils tels que la pipette, la pince de préhension, un stylet ainsi qu'un étaleur. Ce protocole réussit aussi en simulation et sur le robot. La problématique majeure survient lors de la planification. Elle ne prend pas en compte les fils qui sont nécessaires au fonctionnement du préhenseur, du senseur de force et de la pipette électronique. En pratique, certaines trajectoires sont impraticables pour plusieurs raisons : le fil s'entortille sur une partie du robot, ou le fil touche à des objets qui sont stériles. Cette limitation fait en sorte que la planification doit se faire manuellement, car ce serait difficile d'avoir une heuristique pour minimiser l'entortillement ou la proximité des fils versus les objets. Il faut parfois choisir des segments de chemin moins intéressants pour s'assurer que le fil ne sera pas en contact avec un objet. Pour l'entortillement, il est possible de limiter les joints dans un certain éventail de valeurs. Les robots de deuxième génération, comme le robot UR3e, n'ont pas ce problème, car ils donnent accès à des fils qui passent à l'intérieur du robot. Une autre problématique se présente avec le choix de la boucle de contrôle pour les mouvements libre de contact, soit le compensateur proportionnel. Il faut réduire la vitesse et l'accélération du robot pour qu'il puisse suivre la trajectoire le plus fidèlement possible. La réduction est nécessaire pour minimiser les effets inertiels et centrifuges du robot. Le contrôleur par anticipation avec PID pourrait assurément augmenter la vitesse et l'accélération en optimisant les gains appropriés. D'autres types de contrôleur pourraient permettre d'utiliser le robot près de sa capacité maximal, en ayant recours à un modèle virtuelle de dynamique, faute d'avoir un contrôleur interne en torque pour le UR3.

Comme autres limitations à nos travaux, la planification des déplacements du robot peut être très longue pour un protocole comprenant plusieurs actions. Certaines configurations de la plateforme peuvent également être complexes et nécessiter un plus grand temps de

planification, surtout avec des contraintes cartésiennes. Il est parfois difficile de déterminer le temps à allouer à un algorithme pour trouver une trajectoire. Il faut trouver un compromis entre fournir assez de temps pour converger vers une solution asymptotiquement optimale, et pas trop pour ne pas attendre de longues périodes inutilement. La planification avec des contraintes cartésiennes devra être analysée rigoureusement pour privilégier la solution qui retourne le plus haut taux de réussite et d'exactitude à l'intérieur d'un délai minimal.

De plus, il n'y a présentement pas de système de vision intégré à la plateforme robotique, ce qui limite l'ajout de modules qui ne sont pas déterministes. Par exemple, la position exacte des microtubes ne peut être prévue lorsque la centrifugeuse est au repos. L'ajout de vision est prévue dans les travaux futurs, mais seulement pour les objets qui ont des éléments non prévisibles. Comme l'environnement est structuré, si l'opérateur se trompe lors du placement des objets, alors il n'y a pas possibilité de replanification en ligne, ce qui nécéssiterait plusieurs capteurs supplémentaires dont un système de vision.

La caractérisation des objets est simple mais il n'y en a encore relativement peu dans la base de données (environ 25). Il reste à voir si la représentation choisie pour la caractérisation va rester flexible et simple pour une centaine d'objets, surtout pour l'implémentation logicielle qui utilise cette caractérisation. Il serait intéressant d'avoir une solution assez générique pour qu'une communauté puisse partager leurs caractérisations d'objets. Ceci pourrait servir non seulement pour ce projet, mais pour n'importe quelle architecture qui veut interagir avec des objets.

Il serait envisageable d'avoir recours à un système automatique de changement d'outil qui utiliserait un seul effecteur plutôt que de transporter en permanence la pince et la pipette à l'extrémité du bras articulé. Cette option a été écartée pour le projet en raison de son coût plus élevé et à au temps supplémentaire requis pour changer les effecteurs, ce qui forcément implique un temps d'exécution plus long. Cet aspect pourrait être reconsidéré dans le futur afin d'augmenter la diversité d'outils pouvant être utilisés facilement par le bras robotique.

Plusieurs protocoles ont des temps morts entre certaines opérations (plus d'une heure), il serait donc intéressant de combiner plusieurs protocoles à la fois. Avec la structure proposée dans ce projet, il serait difficile d'y parvenir, car tout doit être connu *a priori*.

Finalement, une attention particulière doit être portée quant à la disposition initiale des objets sur la table de travail. Un mauvais placement des objets peut faire échouer la planification : certains endroits sont inatteignables dus à la cinématique du robot, des collisions

ou des singularités. De plus, l'efficacité d'une tâche est fortement liée à cette disposition. Dans le futur, cette disposition manuelle pourrait être automatisée pour optimiser certains critères comme le temps d'exécution et le travail mécanique du robot.

ANNEXE A

Interface graphique

Une interface graphique est en cours de développement dans RViz, un logiciel de visualisation 3D pour l'écosystème de ROS, afin de manipuler la scène robotique. À ce jour, l'interface permet d'ajouter des objets, préalablement ajoutés dans la base de données, pour créer la scène désirée. La scène peut aussi être exportée et importée.

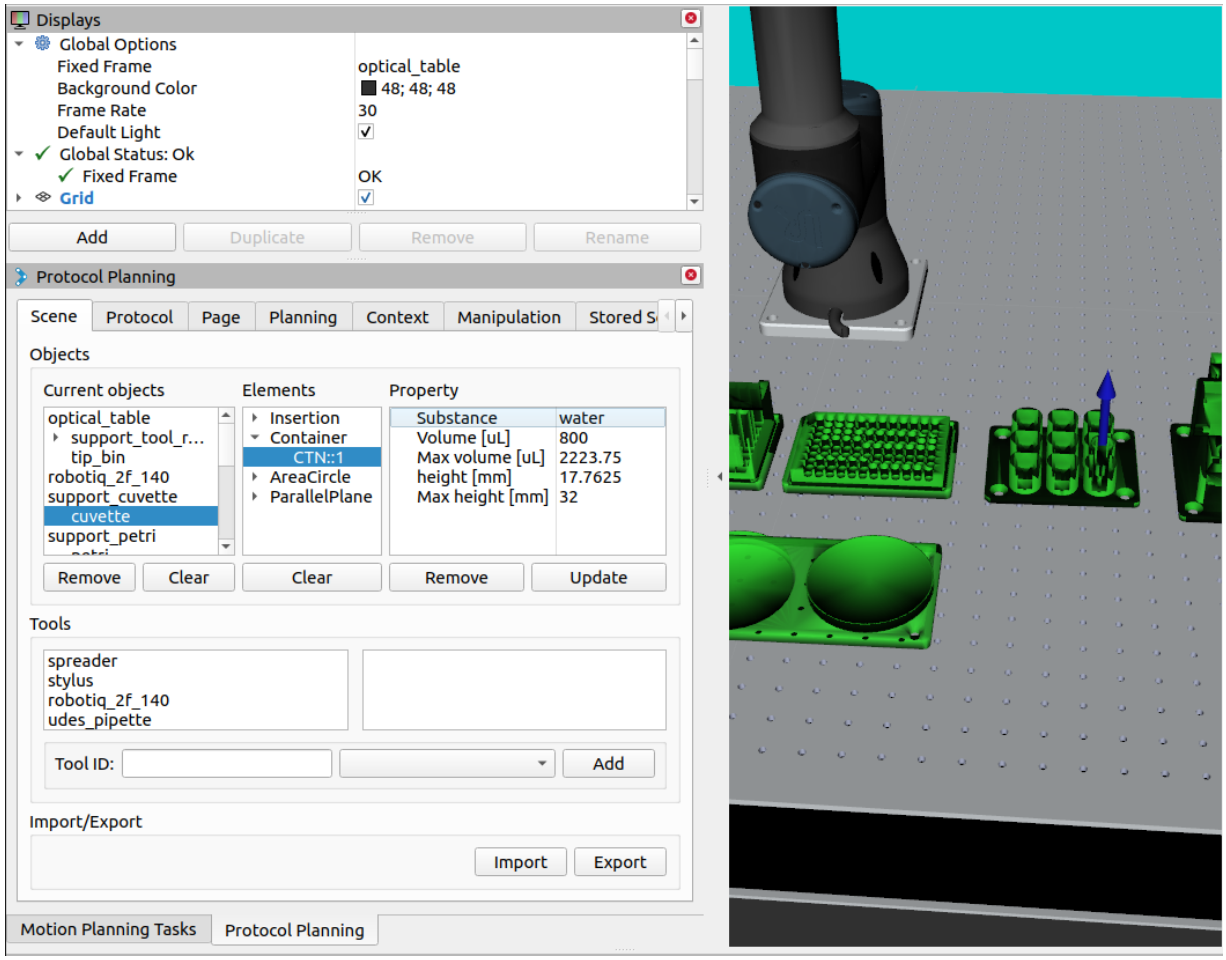


Figure A.1 Interface graphique pour générer la scène robotique.

ANNEXE B

Métriques pour l'analyse de planificateur

La figure B.1 donne une définition qualitative et mathématique des métriques utilisées lors de l'analyse des planificateurs de mouvement.

Total time (s):
Time taken by the whole process. It is calculated with the following formula:
T = plan_time + interpolation_time + simplify_time + process_time
The lower the value, the better performance of the planner.
Solved (%):
Percentage of queries that the planner was able to find a solution for.
The higher the value, the better performance of the planner.
Correct (%):
Percentage of solved plans generated by the planner that are correct. Correctness means that path trajectory avoids collisions and all waypoints satisfy given bounds.
The higher the value, the better performace of the planner.
Length (rad):
Calculated by a sum of angles traveled by each of the joints. Formula:
$L = sum_{i=0}^{n-1}\{abs(x_i - x_{i0})\}$, where:
n - number of robot's joints,
x - joint's goal position,
x_0 - joint's initial position.
The lower the value, the better the plan.
Smoothness:
Looks at three consecutive waypoints and the angle formed between them. Value is calculated as a square of sum of all the angles calculated that way. Formula:
$S = sum_{i=2}^{n-1}\{(2 * (pi - arccos((d_{i-2,i-1}^2 + d_{i-1,i}^2 - d_{i-2,i}^2)/(2 * d_{i-2,i-1} * d_{i-1,i}))))^2\}$, where:
n - number of waypoints on the path,
$d_{x,y}$ - distance between waypoints with index x and y.
Aligned points result in S = 0.
The lower the value, the smoother plan.

Figure B.1 Définition des métriques pour l'analyse de planificateur [84].

ANNEXE C

Configuration pour l'analyse des planificateurs

Le module ROS *moveit_ros_benchmarks* a été employé pour générer les différentes données utilisées dans les graphiques pour l'analyse des planificateurs. La version 1.5.2 a été utilisée pour la librairie OMPL. La version 1.0.7 a été utilisée pour la librairie MoveIt. Les figures C.1, C.2 et C.3 montrent respectivement les fichiers de configuration utilisés pour l'analyse des planificateurs sous OMPL, CHOMP et STOMP.

ompl_planning.yaml

```yaml
RRT:
  type: geometric::RRT
  range: 0.0        # Max motion added to tree.
  goal_bias: 0.05   # When close to goal select goal, with this probability.

RRTConnect:
  type: geometric::RRTConnect
  range: 0.0  # Max motion added to tree.

RRTstar:
  type: geometric::RRTstar
  range: 0.0  # Max motion added to tree.
  goal_bias: 0.05  # When close to goal select goal, with this probability.
  delay_collision_checking: 1  # Stop collision checking as soon as C-free parent found.

PRM:
  type: geometric::PRM
  max_nearest_neighbors: 10  # use k nearest neighbors.

PRMstar:
  type: geometric::PRMstar

AnytimePathShortening:
  type: geometric::AnytimePathShortening
  shortcut: true        # Attempt to shortcut all new solution paths.
  hybridize: true       # Compute hybrid solution trajectories.
  max_hybrid_paths: 48  # Number of hybrid paths generated per iteration.
  num_planners: 32      # The number of default planners to use for planning.
```

Figure C.1 Fichier de configuration pour les planificateurs dans OMPL.

chomp_planning.yaml

```yaml
planning_time_limit: 20.0
max_iterations: 200
max_iterations_after_collision_free: 5
smoothness_cost_weight: 0.1
obstacle_cost_weight: 1.0
learning_rate: 0.01
animate_path: true
add_randomness: false
smoothness_cost_velocity: 0.0
smoothness_cost_acceleration: 1.0
smoothness_cost_jerk: 0.0
hmc_discretization: 0.01
hmc_stochasticity: 0.01
hmc_annealing_factor: 0.99
use_hamiltonian_monte_carlo: false
ridge_factor: 0.0
use_pseudo_inverse: false
pseudo_inverse_ridge_factor: 1e-4
animate_endeffector: false
animate_endeffector_segment: "panda_rightfinger"
joint_update_limit: 0.1
collision_clearence: 0.2
collision_threshold: 0.07
random_jump_amount: 1.0
use_stochastic_descent: true
enable_failure_recovery: false
max_recovery_attempts: 5
trajectory_initialization_method: "quintic-spline"
```

Figure C.2 Fichier de configuration pour l'algorithme CHOMP.

stomp_planning.yaml

```yaml
stomp/right_eef:
  group_name: right_eef
  optimization:
    num_timesteps: 51
    num_iterations: 50
    num_iterations_after_valid: 15
    num_rollouts: 10
    max_rollouts: 100
    initialization_method: 1
    control_cost_weight: 0.0
  task:
    noise_generator:
      - class: stomp_moveit/NormalDistributionSampling
        stddev: [0.2, 0.2, 0.2, 0.2, 0.2, 0.2]
    cost_functions:
      - class: stomp_moveit/CollisionCheck
        check_intermediate_collisions: false
        kernel_window_percentage: 0.07
        collision_penalty: 1.0
        cost_weight: 1.0
        longest_valid_joint_move: 0.005
    noisy_filters:
      - class: stomp_moveit/JointLimits
        lock_start: True
        lock_goal: True
```

Figure C.3 Fichier de configuration pour l'algorithme STOMP.

ANNEXE D
Discrétisation et planification de mouvement

Les planificateurs qui discrétisent l'espace de configuration ont un paramètre pour ajuster la longueur minimale requise entre deux points discrets. Par défaut, le paramètre *longest_valid_segment_fraction* (LVSF) est de 0.005. Ceci est bon pour la majorité des problèmes, sauf pour des modèles géométriques qui sont très fins comme des puits ou des racks à embout de pipette. En effet, comme il n'y a pas assez de points discrétisés, l'algorithme ne détectera pas nécessairement une collision. Une collision pourrait être détectée à la suite d'étapes d'optimisation internes ou non. Un détecteur de collision de type CCD pourrait détecter ces problèmes, mais il n'est pas encore intégré à MoveIt. Évidemment, le temps de planification va augmenter considérablement [85].

En faisant quelques analyses, un LVSF à 0.0001 semble être un bon compromis pour bien détecter les collisions pour des objets étroits ou de petites tailles. Les figures D.2a, D.2b, D.2c, D.2d, D.2e et D.2f démontrent l'importance de ce paramètre pour avoir une bonne analyse, sans quoi les trajets retournés ont de fortes chances d'être en collision. Pour la même configuration, deux tests sont comparés avec deux paramètres de discrétisation différents : 0.005 (paramètre par défaut) et 0.0001 (paramètre qui donne plus de points discrétissé). À noter que ce paramètre n'affecte pas CHOMP et STOMP car ils utilisent un autre type de paramètre.

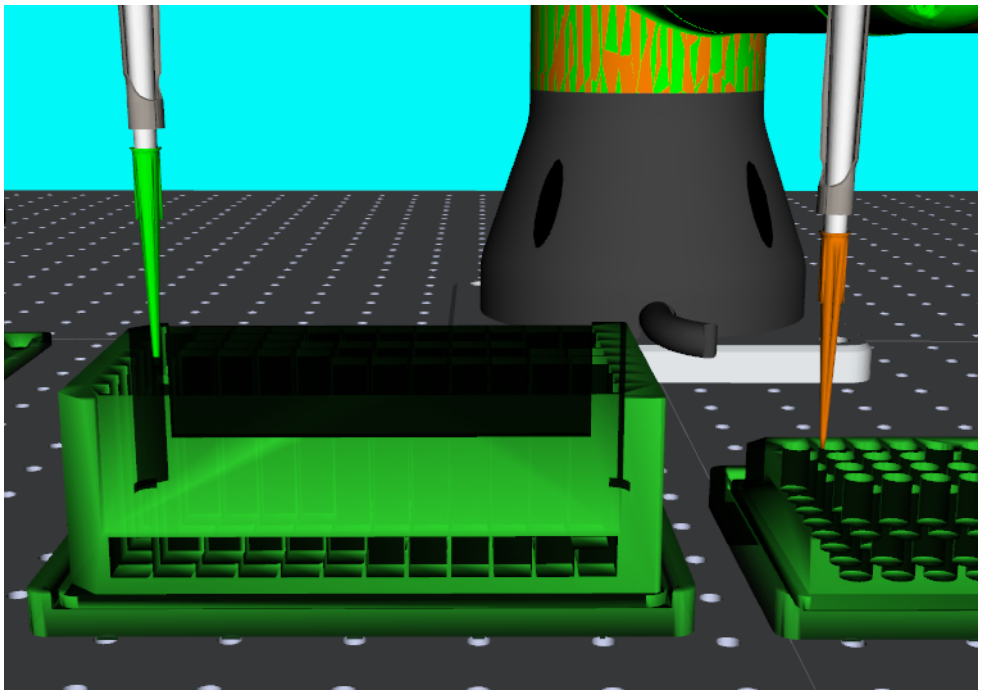
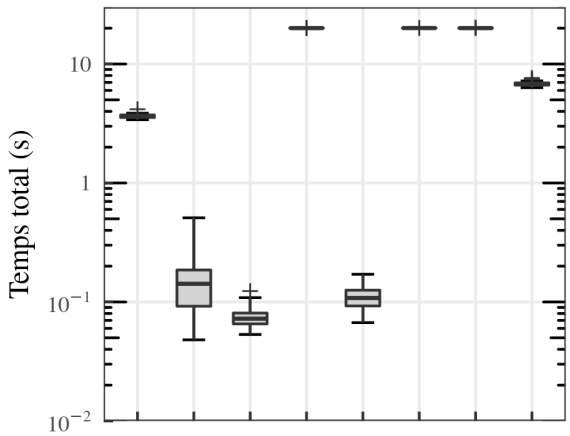
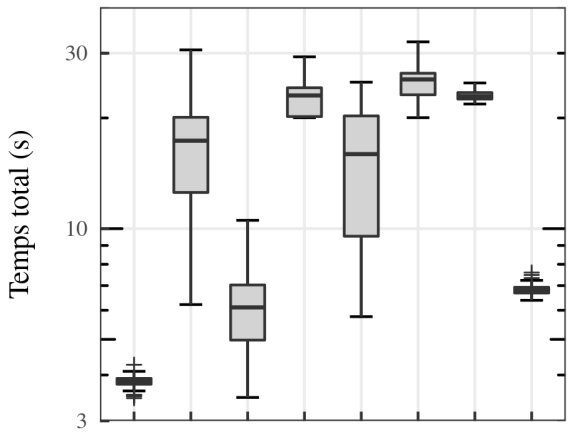


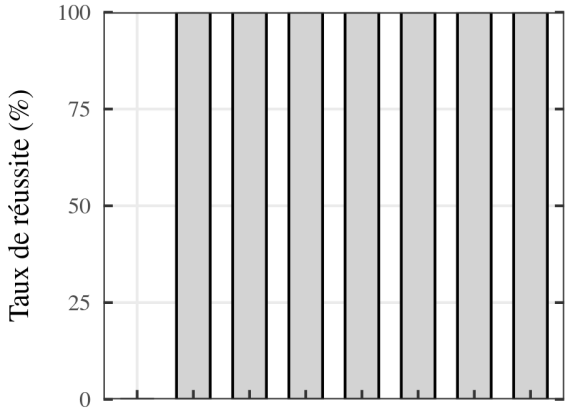
Figure D.1 Test I : Configuration de départ (vert) et de fin (orange).



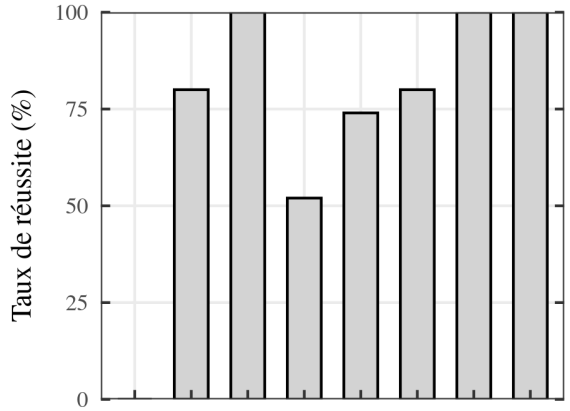
(a) Temps de planification (LVSF = 0.005)



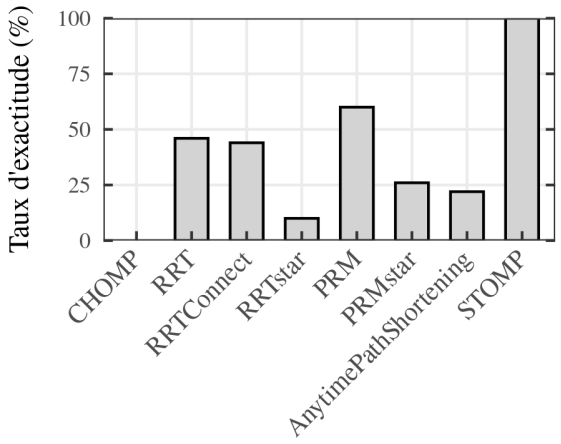
(b) Temps de planification (LVSF = 0.0001)



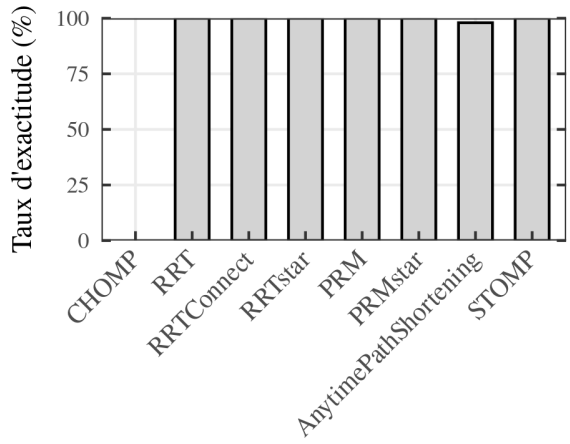
(c) Taux de réussite (LVSF = 0.005)



(d) Taux de réussite (LVSF = 0.0001)



(e) Taux d'exactitude (LVSF = 0.005)



(f) Taux d'exactitude (LVSF = 0.0001)

Figure D.2 Analyse du paramètre LVSF (0.005 et 0.0001) et de ses effets sur le temps de planification (a, b), le taux de réussite (c, d) et le taux d'exactitude (e, f) pour la configuration du Test I. Les planificateurs sont seulement indiqués pour les figures (e) et (f) sur l'axe des X, mais ont la même représentation pour les figures (a), (b), (c) et (d).

ANNEXE E
Statistiques descriptives

Les tests présentés à la section 3.3.1 utilisent des diagrammes en boîte (*box plot*). La figure E.1 décrit spécifiquement les statistiques de la distribution des échantillons et ce qu'elles représentent sur le graphique. Le diagramme en boîte donne de l'information sur l'échantillon minimal et maximal (excluant les valeurs aberrantes), la médiane, le quartile supérieur, le quartile inférieur et les données aberrantes (*whiskers*).

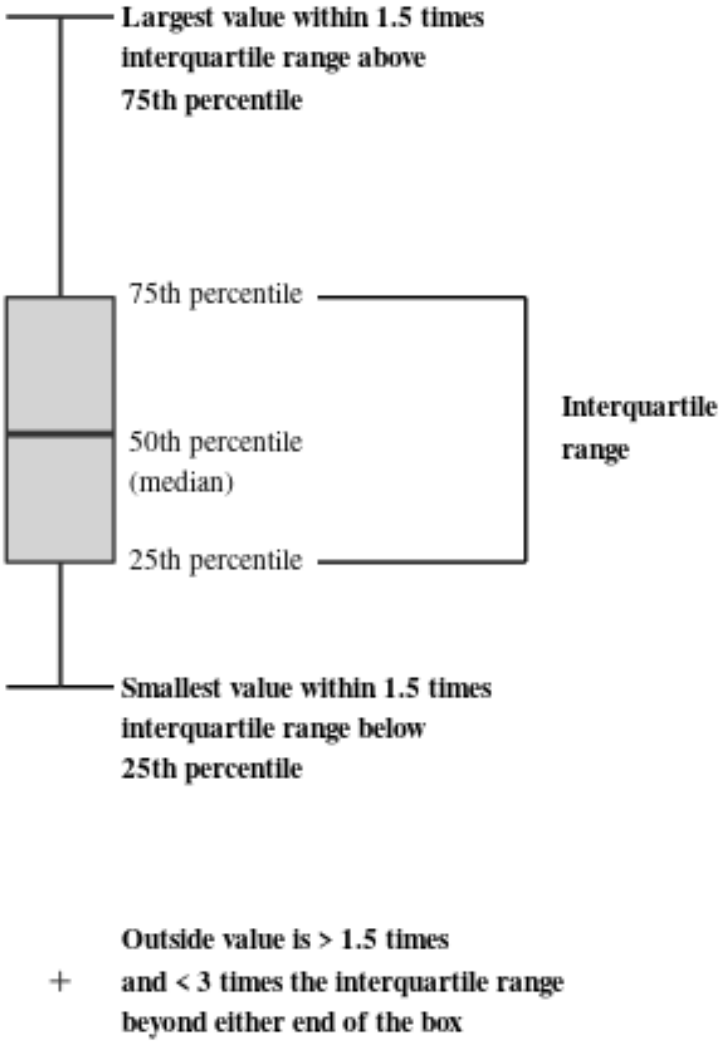


Figure E.1 Standard utilisé pour les diagrammes en boîte.

LISTE DES RÉFÉRENCES

[1] Korus, S. et Analyst, A. R. K. (août 2017). *Are Industrial Robot Costs Hitting an Inflection Point ?* `https://ark-invest.com/research/industrial-robot-costs` (page consultée le 2018-07-12).

[2] Sandberg, T. E., Salazar, M. J., Weng, L. L., Palsson, B. O. et Feist, A. M. (2019). The emergence of adaptive laboratory evolution as an efficient tool for biological discovery and industrial biotechnology. *Metabolic Engineering*, volume 56, p. 1–16.

[3] Wang, H. H., Isaacs, F. J., Carr, P. A., Sun, Z. Z., Xu, G., Forest, C. R. et Church, G. M. (2009). Programming cells by multiplex genome engineering and accelerated evolution. *Nature*, volume 460, numéro 7257, p. 894–898.

[4] Eppendorf (2018). Skk_5_2014_p. *Training epMotion 5070/5075.* `http://web2.mendelu.cz/af_239_nanotech/nanolabsys/skoleni/SKK_5_2014_p.pdf` (page consultée le 2018-05-12).

[5] Andrew Alliance (2018). Andrew 1000g. *Choose the Right Liquid Handing Robot for Your Application.* `https://www.andrewalliance.com/wp-content/uploads/2018/04/1000g.png` (page consultée le 2018-04-27).

[6] Opentrons (2018). ot2_precise. *Lab Automation Has Never Been Easier.* `http://opentrons.com/ot-2` (page consultée le 2018-05-12).

[7] Opentrons (2018). 01_temp_deck_no_block. *Modules.* `https://opentrons.com/modules` (page consultée le 2018-05-12).

[8] Siciliano, B., Sciavicco, L., Villani, L. et Oriolo, G. (2009). *Robotics : Modelling, Planning and Control.* Advanced Textbooks in Control and Signal Processing, Springer-Verlag, London, 632 p.

[9] Greenfield, D. (2017). *Choosing Between Cobots and Industrial Robots.* `https://www.automationworld.com/choosing-between-cobots-and-industrial-robots` (page consultée le 2018-07-27).

[10] Smith, N. (2018). *Cobots and Industrial Robots : Choose the Right Robot for the Job.* `https://www.techbriefs.com/component/content/article/tb/supplements/motion-design/features/28809` (page consultée le 2018-07-25).

[11] Tobe, F. (décembre 2015). *Why Co-Bots Will Be a Huge Innovation and Growth Driver for Robotics Industry.* `https://spectrum.ieee.org/automaton/robotics/industrial-robots/collaborative-robots-innovation-growth-driver` (page consultée le 2018-05-15).

[12] Lynch, K. et Park, F. (2017). *Modern Robotics : Mechanics, Planning, and Control.* Cambridge University Press.

[13] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K.

[14] Lozano-Perez (1983). Spatial planning : A configuration space approach. *IEEE Transactions on Computers*, volume C-32, numéro 2, p. 108–120.

[15] Teller, S. (2010). RSS lecture 10 : Configuration space for motion planning. Cambridge MA.

[16] Hayes, M., L. Husty, M. et J. Zsombor-Murray, P. (2002). Singular configurations of wrist-partitioned 6R serial robots : A geometric perspective for users. *Transactions of the Canadian Society for Mechanical Engineering*, volume 26, p. 15.

[17] Quillen, A. (2018). Phy411 lecture notes part 1. *PHY 411 : Lecture Notes and Problems*. `http://astro.pas.rochester.edu/~aquillen/phy411/lecture1.pdf` (page consultée le 2021-02-29).

[18] Klingensmith, M. (2013). *Overview of Motion Planning*. `https://www.gamasutra.com/blogs/MattKlingensmith/20130907/199787/Overview_of_Motion_Planning.php` (page consultée le 2021-01-15).

[19] Ratliff, N., Zucker, M., Bagnell, J. A. D. et Srinivasa, S. (2009). Chomp : Gradient optimization techniques for efficient motion planning. Dans *Proceedings of IEEE International Conference on Robotics and Automation*. p. 489 – 494.

[20] Karaman, S. (2010). Soundness and completeness of state space search. *Lecture Notes*. `https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec04.pdf` (page consultée le 2021-03-12).

[21] Shiller, Z. (2015). *Off-Line and On-Line Trajectory Planning, volume 29*. p. 29–62.

[22] Bakdi, A., Hentout, A., Boutamai, H., Maoudj, A., Hachour, O. et Bouzouia, B. (2017). Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control. *Robotics and Autonomous Systems*, volume 89, p. 95–109.

[23] Pan, J. et Manocha, D. (2015). Efficient configuration space construction and optimization for motion planning. *Engineering*, volume 1, numéro 1, p. 046 – 057.

[24] Teller, S. (2010). RSS lecture 11 : Rapidly-exploring random trees (RRTs) for motion planning. Cambridge MA.

[25] Hsu, D., Latombe, J.-C. et Kurniawati, H. (2006). On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, volume 25, numéro 7, p. 627–643.

[26] Kavraki Lab Rice University (2021). Open motion planning library : A primer. *The Open Motion Planning Library*. `https://ompl.kavrakilab.org/OMPL_Primer.pdf` (page consultée le 2021-02-16).

[27] Kuffner, J. et LaValle, S. (2000). RRT-Connect : An efficient approach to single-query path planning. Dans *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2. p. 995–1001.

[28] Meijer, J., Lei, Q. et Wisse, M. (2017). Performance study of single-query motion planning for grasp execution using various manipulators. Dans *Conference : 2017 18th International Conference on Advanced Robotics (ICAR)*. p. 450–457.

[29] Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K. et Abbeel, P. (2014). Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, volume 33, numéro 9, p. 1251–1270.

[30] Luna, R., Sucan, I., Moll, M. et Kavraki, L. (2013). Anytime solution optimization for sampling-based motion planning. Dans *Proceedings - IEEE International Conference on Robotics and Automation*. p. 5068–5074.

[31] Petrovic, L., Persic, J., Seder, M. et Markovic, I. (2019). Stochastic optimization for trajectory planning with heteroscedastic gaussian processes. *CoRR*, volume abs/1907.07521.

[32] MoveIt (2021). *CHOMP Planner*. `https://ros-planning.github.io/moveit_tutorials/doc/chomp_planner/chomp_planner_tutorial.html` (page consultée le 2021-01-15).

[33] Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P. et Schaal, S. (2011). Stomp : Stochastic trajectory optimization for motion planning. Dans *Proceedings - IEEE International Conference on Robotics and Automation*. p. 4569–4574.

[34] MoveIt (2021). *STOMP Planner*. `https://ros-planning.github.io/moveit_tutorials/doc/stomp_planner/stomp_planner_tutorial.html` (page consultée le 2021-01-15).

[35] MoveIt (2021). *TrajOpt Planner*. `https://ros-planning.github.io/moveit_tutorials/doc/trajopt_planner/trajopt_planner_tutorial.html` (page consultée le 2021-01-15).

[36] Dai, S. (2018). *Probabilistic Motion Planning and Optimization Incorporating Chance Constraints*. Thèse de doctorat, Department of Mechanical Engineering, Massachusetts Institute of Technology.

[37] Kingston, Z., Moll, M. et Kavraki, L. (2018). Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, volume 1.

[38] Pham, H. et Pham, Q. C. (2018). A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, volume 34, p. 645 – 659.

[39] Kunz, T. et Stilman, M. (2012). Time-optimal trajectory generation for path following with bounded acceleration and velocity. Dans *Robotics : Science and Systems*. p. 09–13.

[40] MoveIt (2021). *Time Parameterization*. `https://ros-planning.github.io/moveit_tutorials/doc/time_parameterization/time_parameterization_tutorial.html` (page consultée le 2021-01-15).

[41] Martins, F. N., Sarcinelli-Filho, M. et Carelli, R. (2017). A velocity-based dynamic model and its properties for differential drive mobile robots. *Journal of Intelligent & Robotic Systems*, volume 85, numéro 2, p. 277–292.

[42] Nakamura, Y. (1991). *Advanced Robotics : Redundancy and Optimization*. Addison-Wesley.

[43] Owen-Hill, A. (2016). *Robotics Research 101 : Getting Started with Force Control*. `https://blog.robotiq.com/robotics-research-101-getting-started-with-force-control` (page consultée le 2021-02-19).

[44] Siciliano, B. (1996). Parallel Force/Position Control of Robot Manipulators. Dans Giralt, G. et Hirzinger, G., *Robotics Research*. Springer, London, p. 78–89.

[45] Robotiq (2017). Cobots Ebook. *Collaborative Robots Buyer's Guide*. `https://blog.robotiq.com/collaborative-robot-ebook` (page consultée le 2018-07-11).

[46] Messmer, F., Hawkins, K., Edwards, S., Glaser, S. et Meeussen, W. (2017). *Universal Robots*. `http://wiki.ros.org/universal_robots` (page consultée le 2018-07-12).

[47] Universal Robots (2018). Ur3. *Universal Robots UR3*. `https://www.universal-robots.com/media/1802342/ur3.png` (page consultée le 2021-02-19).

[48] Amazon Robotics (2016). 2016 amazon picking challenge official rules. *Challenge Details*. `http://pwurman.org/amazonpickingchallenge/APC_2016_Official_Rules.pdf` (page consultée le 2021-02-16).

[49] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M. et Wurman, P. R. (2016). Analysis and Observations from the First Amazon Picking Challenge. *arXiv :1601.05484 [cs]*.

[50] Zhang, H. et Cao, Q. (2017). Texture-less object detection and 6D pose estimation in RGB-D images. *Robotics and Autonomous Systems*, volume 95, p. 64–79.

[51] Tobías, L., Ducournau, A., Rousseau, F., Mercier, G. et Fablet, R. (2016). Convolutional neural networks for object recognition on mobile devices : A case study. Dans *IEEE 23rd International Conference on Pattern Recognition*. p. 3530–3535.

[52] Bui, H. M., Lech, M., Cheng, E., Neville, K. et Burnett, I. S. (2016). Object recognition using deep convolutional features transformed by a recursive network structure. *IEEE Access*, volume 4, p. 10059–10066.

[53] Andreopoulos, A. et Tsotsos, J. K. (2013). 50 years of object recognition : Directions forward. *Computer Vision and Image Understanding*, volume 117, numéro 8, p. 827–891.

[54] Belussi, L. et Hirata, N. (2011). Fast QR code detection in arbitrarily acquired images. Dans *24th IEEE SIBGRAPI Conference on Graphics, Patterns and Images*. p. 281–288.

[55] Lévesque, F. (2016). *Prise autonome d'objets divers avec un robot sériel industriel, un préhenseur sous-actionné et une caméra 3D*. Mémoire de maîtrise, Université Laval, Québec, Canada.

[56] ROS (2018). *Introduction*. `http://wiki.ros.org/ROS/Introduction` (page consultée le 2021-02-15).

[57] ROS Industrial (2017). *The Challenge : Transitioning Robotics R&D to the Factory Floor*. `https://rosindustrial.org/the-challenge` (page consultée le 2021-02-15).

[58] ROSIN (2017). The ROSIN Project : Towards an eu digital industrial platform for robotics through open-source software. *ROSIN Press Release [03-2017]*. `https://www.rosin-project.eu/wp-content/uploads/2017/03/ROSIN-press-release.pdf` (page consultée le 2021-02-14).

[59] MoveIt (2018). *Robots*. `https://moveit.ros.org/robots/` (page consultée le 2021-01-23).

[60] Andersen, T. T. (2015). *Optimizing the Universal Robots ROS Driver.* (Rapport technique). Technical University of Denmark, Department of Electrical Engineering, 35 p.

[61] Chitta, S., Marder-Eppstein, E., Meeussen, W., Pradeep, V., Rodríguez Tsouroukdissian, A., Bohren, J., Coleman, D., Magyar, B., Raiola, G., Lüdtke, M. et Fernández Perdomo, E. (2017). ros_control : A generic and simple control framework for ros. *The Journal of Open Source Software*.

[62] Hawkins, K. P. (2013). *Analytic Inverse Kinematics for the Universal Robots UR-5/UR-10 Arms* (Rapport technique). Georgia Institute of Technology, 5 p.

[63] Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. Thèse de doctorat, Carnegie Mellon University, Pittsburgh, PA.

[64] Spong, M., Hutchinson, S. et Vidyasagar, M. (2005). *Robot Modeling and Control*. Wiley select coursepack, Wiley.

[65] Beeson, P. et Ames, B. (2015). TRAC-IK : An open-source library for improved solving of generic inverse kinematics. Dans *Conference : IEEE RAS Humanoids Conference*.

[66] Beeson, P. et Ames, B. (2015). *Public Repository for the Current Stable Version of the TRACLabs' IK Solver*. https://bitbucket.org/traclabs/trac_ik/src/master/ (page consultée le 2021-01-07).

[67] Universal Robots (2021). *DH Parameters for Calculations of Kinematics and Dynamics*. https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/ (page consultée le 2021-02-23).

[68] Gilson (2018). LT380069-05 Pipette Specs Chart. *Gilson Pipette Specifications*. http://www.gilson.com/Resources/LT380069-05%20Pipette%20Specs%20Chart_No%20Spreads.pdf (page consultée le 2018-07-13).

[69] ISO (2002). *Piston-Operated Volumetric Apparatus*. International Organization for Standardization, Geneva, CH, 99999 p. (Norme ISO 8655 :2002).

[70] Gilson (2018). Norm ISO 8655 and Gilson Specifications. *The Maximum Errors Tolerated*. http://bilder.pretech.nu/files/norm_iso_8655_and_gilson_specificationspdf.pdf (page consultée le 2018-09-05).

[71] Opentrons (2018). OT-2 Pipette White Paper. *Intro to Opentrons Electronic Pipettes*. https://s3.amazonaws.com/opentrons-landing-img/pipettes/OT-2-Pipette-White-Paper.pdf (page consultée le 2018-07-13).

[72] Gilson (2018). GGP Pipette Service and Maintenance. *Pipette Calibration*. http://www.gilson.com/Resources/GGP_Pipette_Service_and_Maintenance.pdf (page consultée le 2018-07-19).

[73] Gilson (2018). Piston Assembly (Disassembled). *User's Guide*. https://www.gilson.com/pub/static/frontend/Gilson/customtheme/en_US/images/docs/PIPETMANCLASSIC_UG_LT801120-F.pd (page consultée le 2021-03-19).

[74] MoveIt (2018). System Architecture. *Concept*. https://moveit.ros.org/assets/images/diagrams/move_group.png (page consultée le 2021-02-19).

[75] Şucan, I. A., Moll, M. et Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, volume 19, numéro 4, p. 72–82.

[76] Moll, M., Şucan, I. A. et Kavraki, L. E. (2015). Benchmarking motion planning algorithms : An extensible infrastructure for analysis and visualization. *IEEE Robotics & Automation Magazine*, volume 22, numéro 3, p. 96–102.

[77] MoveIt (2021). *Benchmarking*. https://ros-planning.github.io/moveit_tutorials/doc/benchmarking/benchmarking_tutorial.html (page consultée le 2021-02-20).

[78] Coleman, D., Moll, M. et Zelenak, A. (2021). *Guide to Cartesian Planners in MoveIt*. https://picknik.ai/cartesian%20planners/moveit/motion%20planning/2021/01/07/guide-to-cartesian-planners-in-moveit.html (page consultée le 2021-02-19).

[79] Wikipedia (2021). *Polygon Mesh*. https://en.wikipedia.org/wiki/Polygon_mesh (page consultée le 2021-02-17).

[80] Souto, N. (2015). *Video Game Physics Tutorial - Part II : Collision Detection for Solid Objects*. https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects (page consultée le 2021-02-17).

[81] Petit, J. (2020). *Integrating Bullet for Collision Detection*. https://moveit.ros.org/bullet/collision%20detection/moveit/2020/11/18/bullet-collision.html (page consultée le 2021-01-23).

[82] Görner, M., Haschke, R., Ritter, H. et Zhang, J. (2019). MoveIt ! Task constructor for task-level motion planning. Dans *Proceedings IEEE International Conference on Robotics and Automation*.

[83] MoveIt (2021). *MoveIt Task Constructor*. https://ros-planning.github.io/moveit_tutorials/doc/moveit_task_constructor/moveit_task_constructor_tutorial.html (page consultée le 2021-01-15).

[84] Shadow Robot Company (2018). *Benchmark Description*. https://planners-benchmarking.readthedocs.io/en/latest/user_guide/3_benchmark_description.html#benchmark-metrics (page consultée le 2021-02-20).

[85] MoveIt (2021). *OMPL Planner*. https://ros-planning.github.io/moveit_tutorials/doc/ompl_interface/ompl_interface_tutorial.html (page consultée le 2021-01-15).