POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Système autonome d'assistance au repas pour les personnes avec un handicap du haut du corps

GABRIEL DESCÔTEAUX

Département de génie mécanique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie mécanique

Août 2020

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Système autonome d'assistance au repas pour les personnes avec un handicap du haut du corps

présenté par **Gabriel DESCÔTEAUX**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Marek BALAZINSKI, président
Sofiane ACHICHE, membre et directeur de recherche
Maxime RAISON, membre et codirecteur de recherche
Abolfazl MOHEBBI, membre

REMERCIEMENTS

J’aimerais d’abord remercier mes directeurs de recherche pour leurs conseils et l’aide qu’ils m’ont apportée dans le cadre de mon projet de maîtrise.

D’abord, professeur Sofiane Achiche qui m’a accueilli dans son laboratoire alors que je commençais tout juste ma 2e année de baccalauréat et qui m’a fait confiance. Grâce à lui, j’ai pu entamer une carrière en R&D et j’ai pu découvrir une branche du génie qui me passionne au plus haut point.

Ensuite, le professeur Maxime Raison, qui m’a fait profiter de son expertise en génie de la réadaptation lorsque prof. Achiche était en année sabbatique à l’étranger.

Je tiens aussi à remercier tous les autres étudiants au laboratoire de Conception de Systèmes intelligents et Mécatroniques (CoSIM) du prof. Sofiane Achiche, qui ont tous contribué de manière significative à l’agréable atmosphère de travail dans laquelle j’ai pu évoluer dans les dernières années. Cette atmosphère a sans aucun doute contribué à ma productivité et à mes apprentissages.

Enfin, je tiens à remercier mes parents, qui m’ont toujours encouragé à atteindre mes objectifs, autant au niveau académique que personnel. Sans leur soutien inconditionnel, je n’en serais probablement pas là aujourd’hui. Je remercie aussi ma sœur qui a toujours été un exemple pour moi et qui a toujours été présente lorsque j’avais besoin de soutien et d’encouragements dans mes études.

Je veux aussi remercier le professeur Marek Balazinski pour avoir accepté de présider le jury pour ma présentation de mémoire, ainsi que le professeur Abolfazl Mohebbi pour avoir accepté d’être membres de ce jury.

RÉSUMÉ

Pour les personnes souffrant de troubles de mobilité du haut du corps, des tâches quotidiennes qui semblent anodines peuvent devenir extrêmement complexes. La tâche d'alimentation en est un excellent exemple. Dans les 20 dernières années, une multitude de solutions à cette tâche ont été proposées par des laboratoires de recherche ou des entreprises afin d'assister ces personnes. Toutefois, la plupart de ces solutions ont d'importantes limitations : complexité d'installation et de transport, difficulté à contrôler, peu intuitif, etc. L'objectif de ce mémoire est de présenter une preuve de concept de solution abordable permettant à des personnes à mobilité réduite au niveau du haut du corps de s'alimenter plus aisément.

La solution proposée inclut d'abord une caméra de stéréovision combinée à un système de reconnaissance d'objets basé sur l'apprentissage profond afin de repérer et localiser la nourriture dans une assiette située devant l'utilisateur. Cette partie permet à l'utilisateur d'éviter de se soucier de spécifier la position de la nourriture qu'il veut manger; le système le fait pour lui, ce qui rend le tout beaucoup plus intuitif et simple à contrôler. La caméra permet également d'établir un modèle de la scène et des obstacles présents, ce qui permet au système d'éviter ces obstacles.

Les coordonnées de la nourriture à aller chercher sont ensuite envoyées au bras robotique MICO de Kinova, simulé avec ROS et Gazebo. Il s'agit d'un bras de réadaptation conçu pour être monté sur une chaise roulante, ce qui réduit la complexité d'installation et de transport. Un calcul de trajectoire est alors effectué et la cinématique inverse du bras robotique est calculée afin que le bras puisse approcher la nourriture avec l'effecteur choisi, une fourchette, selon un angle d'approche prédéfini. La nourriture est ensuite ramenée à une position fixe près de la bouche de l'utilisateur, le tout de manière entièrement autonome.

En utilisant le système développé, l'utilisateur peut donc se nourrir de manière intuitive et autonome. Selon les tests effectués, le système constitue une preuve de concept valide, n'ayant échoué aucun des cinq essais effectués, et permet à l'utilisateur de manger deux morceaux de nourriture en un temps moyen de 90,22 secondes (dont 18,3% sont dédiés aux calculs). Le reste est donc entièrement utilisé pour le déplacement du bras, qui est une limitation physique du système robotique.

ABSTRACT

For people who suffer from upper-body disabilities, some casual tasks that seem innocuous can actually become extremely complex. Eating is an excellent example of such a task. In the last 20 years, multiple solutions have been suggested by academia and industries in order to help those people. However, most of these solutions have important limitations: complexity of installation or transport, control difficulty, lack of intuitiveness, etc. The goal of this thesis is to present a proof of concept of a low-cost solution that addresses this difficulty in carrying out the feeding task for people with disabilities.

The suggested solution includes a stereovision camera combined with an object recognition system based on deep learning in order to detect and locate food on a plate in front of the user. This part of the system removes the need for the user to specify the exact position of the food he wants to eat; the robotic system does it for him or her, reducing significatively the complexity of the task and making it a lot more intuitive. The camera also allows the system to build a model of the scene, including obstacles, allowing the system to build a collision-free trajectory.

The coordinates for the food to reach is then sent to the robot arm MICO by Kinova, which is simulated using ROS and Gazebo. This arm is specifically made to fit on wheelchairs, thus reducing the complexity of required installation and transport. The path planning is then completed and the inverse kinematics of the robot arm is computed in order to reach the food with the tool selected for the project, a fork, at a predefined angle of approach. The food is then brought back to a constant position near the user's mouth. Everything is done autonomously.

By using the developed system, the user can feed himself in an autonomous and intuitive manner. According to the tests, the system is a good proof of concept, having failed none of the 5 attempts, and allows the user to eat two pieces of food in an average time of 90.22 seconds (of which only 18.3% is dedicated to computations). The remaining 81.7% is thus used entirely for the movement of the robot arm, which is due to a physical limitation of the system.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ADA	Assistive Dextrous Arm
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
DDL	Degrés De Liberté
EQSP	Enquête québécoise sur la Santé de la Population
HOG	History of Oriented Gradient
mAP	Mean Average Precision
OMPL	Open Motion Planning Library
PoGARA	Point of Gaze Assistive Robotic Arm
RCNN	Region Convolutional Neural Network
ResNet	Residual Network
RNC	Réseau de neurones convolutif
ROS	Robot Operating System
RTL	Angle de Roulis/Tangage/Lacet
SBPL	Search-Based Planning Library
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
SOX	Sous-Objectif #X
SSD	Single Shot Detector
STOMP	Stochastic Trajectory Optimization for Motion Planning
TMS	Troubles Musculo-Squelettiques
YOLO	You Only Look Once

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

1.1 Aperçu

Il existe une multitude de troubles physiques qui limitent fortement l'autonomie des personnes. Les troubles musculo-squelettiques (TMS), par exemple, sont parmi les maladies chroniques les plus répandues et peuvent engendrer des douleurs intenses. À Montréal, l'Enquête Québécoise sur la Santé de la Population (EQSP) estime qu'environ 25% des Montréalais souffrent de TMS au courant d'une année [1]**.** La dystrophie musculaire est un autre exemple encore plus limitant. Selon Dystrophie Musculaire Canada, plus de 50 000 Canadiens sont atteints de maladies neuromusculaires comme la dystrophie musculaire [2]. Ces personnes ont alors des limitations fonctionnelles importantes qui génèrent des coûts sociétaux et médicaux importants. Heureusement, depuis plusieurs années, des avenues de solution sont explorées pour améliorer la qualité de vie de ces personnes. La robotique est, entre autres, l'une des avenues les plus explorées. Des systèmes robotiques visant à les accompagner dans leur vie quotidienne ont ainsi été développés. Par exemple, des bras robotiques contrôlés par joystick permettent aux personnes souffrant de ces limitations d'accomplir des tâches de la vie quotidienne qui semblent anodines, comme se nourrir ou tourner une poignée de porte sans assistance.

Toutefois, ces systèmes sont généralement limités par leur architecture trop simple, n'étant ainsi pas assez autonomes, ou par leur architecture trop complexe ; ils sont souvent difficiles à opérer pour les utilisateurs parce qu'ils ont souvent 6 axes de liberté, alors que le joystick ne permet que 3 axes à la fois. Leur manipulation requiert alors un niveau de concentration insupportable à long terme, car les mouvements du préhenseur sont longs et complexes. Il est donc très intéressant de se pencher sur deux problèmes : le premier est l'automatisation des tâches pour simplifier l'utilisation du bras. Le deuxième est l'architecture même de ces bras qui pourrait être moins imposante et plus naturelle.

L'objectif principal de ce projet est donc de développer un système ayant le potentiel d'améliorer la qualité de vie des gens atteints de problèmes de mobilité du haut du corps en automatisant une tâche quotidienne effectuée par un bras robotique, c'est-à-dire la tâche d'alimentation. Le présent mémoire présente donc une solution autonome abordable permettant de les assister dans la tâche d'alimentation.

On présente ainsi un système combinant la stéréovision, la reconnaissance d'objets, la planification de trajectoire et le bras MICO de Kinova afin de repérer et saisir la nourriture dans une assiette située devant l'utilisateur.

Ci-dessous, on retrouve donc une revue de littérature présentant les systèmes d'aide à l'alimentation existants, les caméras stéréo disponibles, les algorithmes de vision par ordinateur, ainsi que les méthodes de planification de trajectoire et résolution de la cinématique inverse pour les bras robotiques. Le Chapitre 2 présente ensuite les objectifs de recherche, puis le Chapitre 3 présente la méthodologie adoptée ainsi que les tests effectués, et le Chapitre 4 présente les résultats de tests, suivis d'une discussion et d'une conclusion.

1.2 Revue de littérature

1.2.1 Robots d'assistance

Le but du présent travail de maîtrise étant de développer un système d'assistance à l'alimentation pour les personnes à mobilité réduite utilisant un robot sériel, il faut d'abord étudier les systèmes existants développés pour répondre à ce besoin. Le système présenté par SECOM [3] est un système d'assistance effectuant uniquement la tâche d'alimentation. L'assiette est fixée sur un plateau et les déplacements de l'effecteur sont préprogrammés. Le système est contrôlé par un pointeur laser que l'utilisateur oriente vers différents boutons photosensibles qui commandent les déplacements droite, gauche, haut, avant, etc. Handy-1 [4, 5], lui, permet d'effectuer plusieurs tâches comme l'alimentation, le brossage de dents, le rasage, etc. Ce dispositif possède aussi une assiette fixe; tous les déplacements sont fixes. Ainsi, le processus n'est pas entièrement autonome et le système nécessite une installation complexe. L'assiette est raclée dans son entièreté pour chaque bouchée.

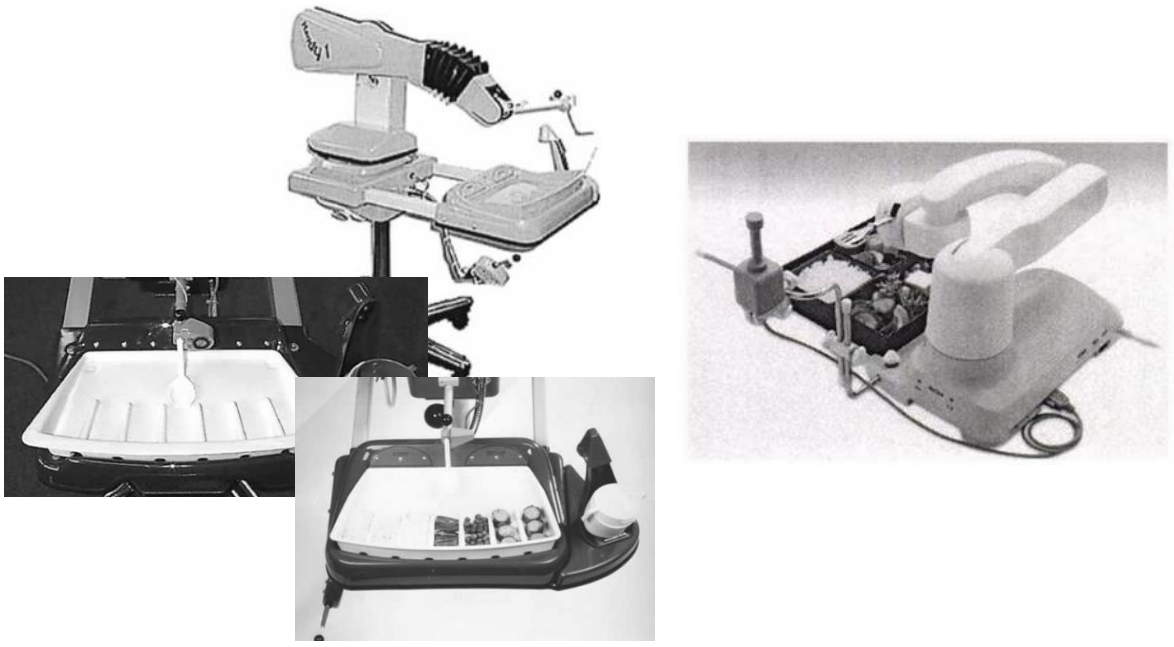


Figure 1.1 Handy-1 (gauche)(Tiré de [4, 5]), MySpoon (droite)(Tiré de [7])

Le dispositif MySpoon [6, 7], aussi développé par SECOM Co., LTD. est plus portatif puisque ses dimensions sont réduites de beaucoup. Cependant, ce dispositif est limité par le placement spécifique de la nourriture dans l'assiette qui est requis. Les robots Mealtime Partner [8], et celui développé par Song et Kim [9] utilisent sensiblement le même fonctionnement. Ces robots ne sont pas non plus autonomes, et requièrent également un arrangement de la scène particulier. Ensuite, *Tokai University*, au Japon, a développé un robot capable d'utiliser des baguettes afin de nourrir l'utilisateur [10]**.** Ce dispositif est ainsi contraint à certains types de nourriture. Bestic [11] et Neater Eater Robotic [12], eux, nécessitent d'être mis en place par une tierce personne. Ces robots d'assistance possèdent une cuillère qui balaie une assiette fixe avant d'aller vers une position fixe où l'utilisateur mange. Il faut donc qu'une personne place correctement la nourriture dans l'assiette, et l'utilisateur n'a pas la possibilité de choisir ce qu'il veut manger.

Figure 1.2 Bestic (Tiré de [11])

Le robot « pousseur » [13] offre une alternative à la manière usuelle de se nourrir. Plutôt que d'aller saisir la nourriture et la ramener à la bouche de l'utilisateur, ce robot permet à l'utilisateur de sélectionner un aliment à manger avec le clignement des yeux et une série de voyants lumineux, puis pousse la nourriture directement vers une cuillère. En réalité, ce système n'est pas très pratique puisqu'il requiert de séparer manuellement les aliments dans différentes rangées.

Le bras d'Ohshima [14] est un pas de plus dans la direction d'un système autonome d'aide au repas, mais n'est pas un système d'assistance au repas en tant que tel. Il s'agit en fait d'un bras robotique utilisé dans le cadre d'une étude visant à étudier l'utilisation d'une cuillère afin de déplacer des liquides d'un bol vers une autre position. Ainsi, le bras doit être capable de maintenir un certain angle et de réduire les vibrations, afin d'éviter de renverser le contenu.

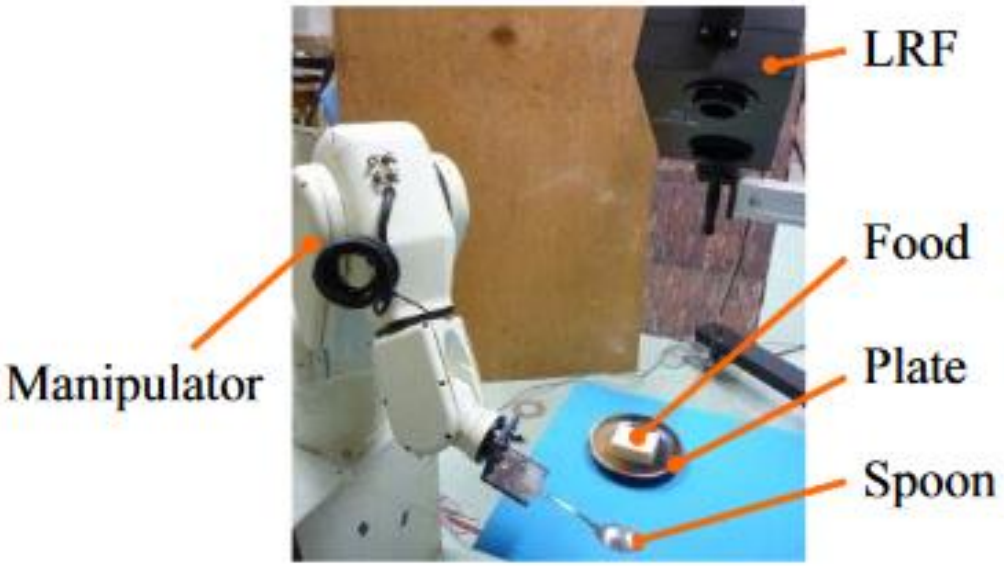


Figure 1.3 Bras d'Ohshima (Tiré de [14])

Le système d'assistance utilisant MANUS [15] utilise un écran tactile permettant à l'utilisateur d'indiquer les points où placer la pince du bras afin de saisir l'objet. MANUS est d'ailleurs un bras d'assistance général à 6 DDL monté sur une chaise roulante. Ce bras permet donc de saisir des objets sélectionnés, mais n'est pas adapté spécifiquement pour saisir de la nourriture puisqu'il n'est pas en mesure de tenir un outil comme une fourchette, et n'est pas autonome puisqu'il requiert que l'utilisateur spécifie manuellement où l'effecteur du bras doit se déplacer.

Figure 1.4 Système d'assistance utilisant MANUS (Tiré de [15])

Dans la même veine, le système PoGARA développé par Law-Kam-Cio et al. [16, 17] à Polytechnique Montréal, utilise un bras de réadaptation afin d'aller saisir automatiquement des objets identifiés par un dispositif de suivi du regard. Ce système, semblable à ceux présentés dans [18], [19] et [20] se rapproche beaucoup d'un système autonome d'aide à l'alimentation. Il n'est toutefois pas tout à fait adapté à la tâche, car il n'est pas en mesure de tenir un outil d'alimentation comme une fourchette.

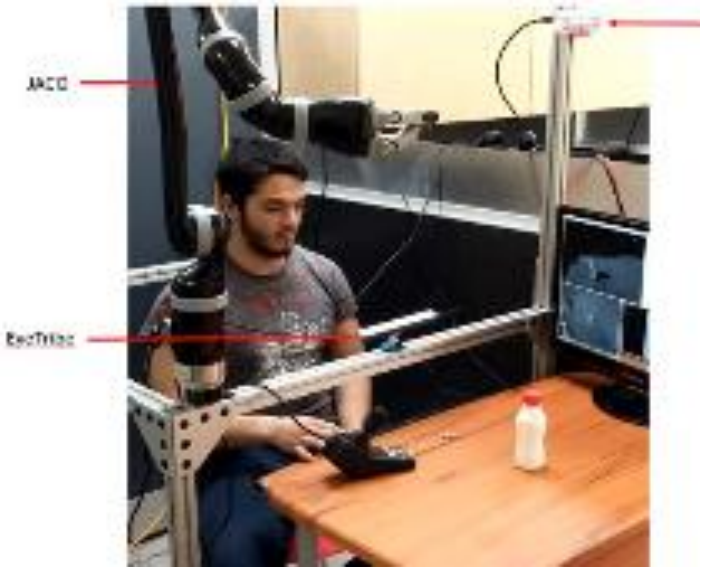


Figure 1.5 Utilisation du système PoGARA (Tiré de [16])

Enfin, Assistive Dextrous Arm (ADA) [21, 22], développé par un laboratoire de l'Université de Washington, offre une solution au problème d'aide au repas. Ce robot, qui utilise le bras MICO de Kinova [23], utilise une caméra stéréo montée sur le poignet du bras pour prendre des images de la scène et l'algorithme RetinaNet [24] afin d'identifier la nourriture se trouvant dans l'assiette, puis utilise la caméra pour localiser la bouche de l'utilisateur et y amener directement la nourriture. Également, le système utilise une fourchette conçue spécifiquement pour ce problème.

Figure 1.6 Démonstration du système ADA (Tiré de [19])

Tableau 1.1 Résumé des systèmes d'assistance existant

Système	Adapté pour des tâches autre que l'alimentation	Adapté pour l'alimentation	Nécessite une installation précise	Autonome
1er robot de SECOM	Non	Oui	Oui	Non
Handy-1	Oui	Oui	Oui	Non
MySpoon, Mealtime Partner, et Song & Kim	Non	Oui	Oui	Non
Robot de Tokai University	Non	Oui	Oui	Non
Bestic et Neater Eater	Non	Oui	Oui	Non
Robot pousseur	Non	Oui	Oui	Non
Ohshima	Non	Oui	Non	Non
MANUS	Oui	Non	Non	Non
PoGARA	Oui	Non	Non	Oui
ADA	Oui	Oui	Non	Oui

On remarque donc que le système ADA est le seul système qui semble permettre de nourrir quelqu'un de manière entièrement autonome, tout en étant adaptable à différentes scènes.

1.2.2 Structure de système robotique

La première méthode possible pour relier les différents modules du projet et assurer la communication entre eux est de les implémenter individuellement, puis d'avoir un système central qui permet d'assurer la liaison entre eux. Toutefois, cette méthode nécessite d'implémenter soi-même tous les modules, ce qui peut être une perte de temps considérant qu'une multitude d'outils

ont déjà été implémentés et optimisés. Par exemple, il serait possible d'implémenter des algorithmes d'apprentissage machine ou d'optimisation soi-même, mais les librairies d'apprentissage machine en Python *Scikit-Learn* offrent des algorithmes optimisés pour la performance. Ainsi, cette méthode pourrait être équivalente à réinventer la roue. Il vaut ainsi mieux d'utiliser un cadre robotique déjà créé.

Robot Operating System (ROS) [25, 26] développé par Willow Garage et le Stanford Artificial Intelligence Laboratory, permet de de simplifier la communication entre les différents modules d'un système robotique, ainsi que d'offrir des librairies faciles à utiliser pour de tels systèmes. Ainsi, ROS permet de compartimenter les parties d'un projet en gérant le protocole de communication entre eux. Les messages échangés sont donc normalisés selon des formats définis par ROS, ce qui permet de rendre chaque module invisible aux yeux des autres modules, car tout ce qui est observé sont les messages d'entrée et de sortie. Il est donc possible d'avoir certains modules écrits en C++ et d'autres écrits en Python, ce qui est beaucoup plus complexe à faire sans ce framework. Également, ROS offre une multitude de librairies utiles en robotique, comme des librairies implémentant SLAM (Simultaneous Localization And Mapping), le contrôle et la cinématique inverse et directe de robots, la vision par ordinateur, l'évitement d'obstacles (*MoveIt!*) [27], etc. ROS offre aussi les librairies de contrôle pour des caméras stéréo et pour une multitude de robots utilisés en recherche et en industrie. Enfin, ROS s'interface facilement avec le simulateur Gazebo, permettant ainsi de simuler un robot avec la physique du monde réel. Ce simulateur permet de rendre le développement d'un système robotique beaucoup plus simple, car on ne court pas le risque d'endommager le système réel.

En date d'octobre 2018, le site web de ROS comptait près de 22 500 membres et 330 000 téléchargements de ROS. En 2018 seulement, environ 5000 articles scientifiques citaient le papier initial de ROS.

1.2.3 Vision par ordinateur

1.2.3.1 Caméras

La présente section vise à comparer les différentes caméras disponibles sur le marché et qui pourraient être utilisées dans le cadre de ce projet de maîtrise. En observant les fiches techniques

de ces capteurs et en définissant les besoins du projet, il a été possible de dresser une liste de critères.

Cette liste comprend :

1. Prix : La caméra choisie doit être abordable.
2. Dimensions : La caméra doit pouvoir être facilement montée sur une chaise roulante de réadaptation.
3. Résolution : Une image ayant une plus grande résolution permettra aux algorithmes de détection d'objet d'obtenir de meilleurs résultats de reconnaissance.
4. Portée : La caméra doit être en mesure de déterminer avec précision la position en profondeur d'objets de 0.5m à 2m de distance, ce qui représente la distance de la nourriture dans une scène d'alimentation.
5. Calibration : La caméra doit être simple à calibrer et ne pas requérir des calibrations trop fréquentes.
6. Types d'images : La caméra doit permettre d'obtenir à la fois des informations sur la profondeur et des images en couleur afin d'identifier les objets d'intérêt.
7. Compatibilité : Le capteur sélectionné doit offrir la possibilité d'être interfacé avec divers cadres robotiques, tels que ROS.

1.2.3.1.1 Kinect

La Kinect [28] est un capteurs souvent utilisé en recherche et développement. Il s'agit d'un capteur simple initialement développé pour l'industrie du jeu vidéo. Le capteur utilise une caméra couleur, ainsi qu'un projecteur et un capteur de rayons infrarouges afin de créer un nuage de points représentant la profondeur des objets. Puisqu'elle utilise des infrarouges, cette caméra et difficilement fonctionnelle à l'extérieur, car les rayons du soleil ont tendance à saturer les capteurs infrarouges. Les images et les informations sont tirées de [28].

Tableau 1.2 Caractéristiques de la caméra Kinect

	Caractéristique	
	Prix($CAD)	135
	Dimensions(mm)	249 x 66 x 67
	Résolution(pixels)	1920 x 1080 à 30 fps
	Portée(m)	0.5 – 4.5
	Calibration	Non nécessaire
	Types d'images	Images couleur & nuage de points (infrarouge)
	Compatibilité	Simple à utiliser en développement ou avec ROS

1.2.3.1.2 Caméra stéréo Tara

La caméra stéréo Tara [29, 30] est une alternative aux systèmes utilisant des rayons infrarouges pour déterminer la profondeur. En utilisant les données de deux images d'une même scène, il devient possible de déterminer la distance d'un point du centre de la caméra. Les images et les informations sont tirées de [29, 30].

Tableau 1.3 Caractéristiques de la caméra Tara

Caractéristique	
Prix($CAD)	336
Dimensions(mm)	100 x 30 x 35
Résolution(pixels)	1504 x 480 à 60 fps
Portée(m)	0.5 – 3.0
Calibration	Une seule fois
Types d'images	Images couleur & nuage de points (stéréo)
Compatibilité	Simple à utiliser en développement ou avec ROS

1.2.3.1.3 Caméra stéréo Intel D415

La caméra stéréo D415 d'Intel [31, 32] est spécialement conçue pour la recherche et le développement. Il s'agit d'une caméra qui offre la possibilité d'évaluer la profondeur d'un point à la fois en stéréo et en infrarouge, ce qui la rend plus robuste aux rayons du soleil. Les images et les informations sont tirées de [31, 32].

Tableau 1.4 Caractéristiques de la caméra D415

Caractéristique	
Prix($CAD)	182
Dimensions(mm)	99 x 20 x 23
Résolution(pixels)	1920 x 1080 à 30 fps
Portée(m)	0.16 – 10.0
Calibration	Une seule fois
Types d'images	Images couleur & nuage de points (stéréo et infrarouge)
Compatibilité	Simple à utiliser en développement ou avec ROS

En observant les caractéristiques des caméras ci-dessus, il semble évident que la caméra à prioriser est la caméra stéréo Intel D415. Effectivement, celle-ci est à peine plus cher que la Kinect, est de loin la plus petite en termes de dimensions, offre la meilleure résolution d'images et la meilleure portée. De plus, elle combine les technologies des deux autres caméras, c'est-à-dire l'utilisation d'infrarouge et de stéréo pour déterminer la profondeur d'un point. Ainsi, il est possible d'utiliser les deux afin d'obtenir un système plus robuste.

1.2.3.2 Algorithmes de vision par ordinateur

La vision par ordinateur, et plus particulièrement la reconnaissance d'objets, a connu deux phases majeures. La première correspond aux algorithmes de reconnaissance classiques. La deuxième correspond aux algorithmes plus récents basés sur l'apprentissage profond. La transition entre les deux époques s'est opérée entre 2012 et 2014. Dès 2001, Viola et Jones ont proposé le premier algorithme capable de détecter des visages en temps réel [33, 34]. À partir de ce moment, la recherche s'est penchée sur la reconnaissance de traits (*features*) afin d'identifier des objets ou des classes d'objets. Un trait est défini comme une information caractéristique d'un objet ou d'une image, tel qu'un point, un contour, une saillance, etc. [35] La méthode Scale Invariant Feature Transform (SIFT) [36] a été développée; en prenant des points clés dans une image et en calculant le descripteur associé, on peut alors identifier une classe. La méthode par histogrammes de gradients orientés (HOG) [37] est un exemple de descripteur qui a été couramment utilisé pendant près d'une décennie afin de détecter des classes d'objets.

Toutefois, les méthodes présentées ci-dessus sont basées sur des features, ou traits, déterminés par l'utilisateur. Ces features sont donc limités par l'expertise du concepteur et la détection d'objets traditionnelle a donc atteint un plateau aux alentours de 2010. En 2012, l'équipe de G.E. Hinton a participé à la compétition appelée ImageNet [38]. Cette compétition consiste à classer et à localiser des objets dans 1,2 million d'images à haute résolution, avec des objets appartenant à 1000 différentes classes. Pour la première fois, une solution utilisant des réseaux de neurones convolutifs (RNC) a été présentée à la compétition, battant tous les records. Alors que la deuxième équipe atteignait un taux d'erreur de 26,2%, l'équipe d'Hinton atteignait plutôt 16,4% [39].

Depuis, la vision par ordinateur a connu un essor fulgurant. Une multitude d'architectures ont été créées : RCNN [40], Fast RCNN [41], Faster RCNN [42], YOLO [43], SSD [44], etc. [45] Certaines de ces architectures se basent sur les précédentes, d'autres utilisent des concepts

différents. Elles ont chacune leurs avantages et leurs inconvénients. Par exemple, YOLO offre un temps de calcul faible en échange d'un taux d'erreur plus élevé, alors que Faster RCNN offre un plus faible taux d'erreur, mais est un peu plus lent [46, 47].

Le plus simple des modèles ci-dessus, RCNN, utilise un réseau de neurones convolutif (RNC) assez classique. L'innovation du modèle est plutôt en lien avec la manière de sélectionner quelles régions de l'image sont transmises au réseau. Traditionnellement, soit on traitait l'image complète, soit on traitait des régions aléatoires de l'image. RCNN propose plutôt d'utiliser de la segmentation d'image simple afin d'obtenir les 2000 régions de l'image les plus importantes. Ces régions sont ensuite transmises à un RNC qui, pour chaque région, extrait les caractéristiques les plus importantes de l'image. Ces caractéristiques sont ensuite envoyées vers un classificateur qui détermine si la région représente un objet connu.

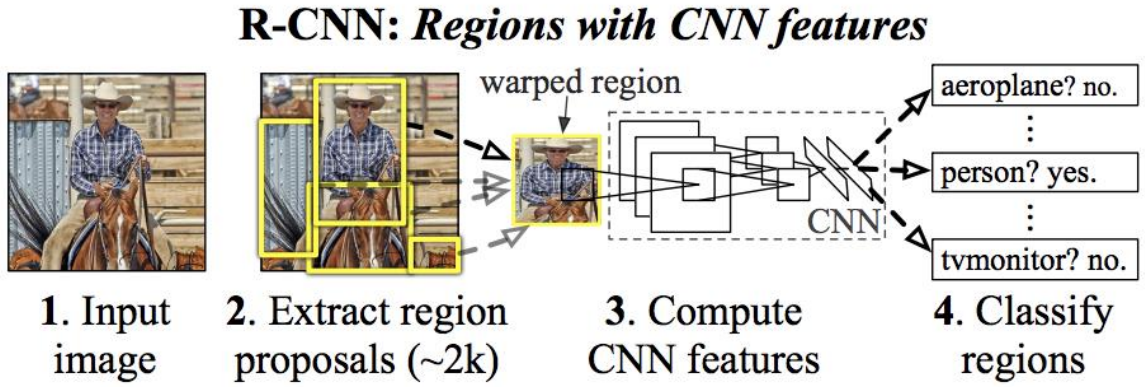


Figure 1.7 Schématisation de RCNN (Tiré de [40])

Fast RCNN propose d'améliorer RCNN en réduisant le nombre de régions qui sont fournies au RNC. En fait, une carte des traits (*convolutional feature map)* est générée à partir de l'image initiale. On utilise ensuite une méthode sélective basée sur cette carte afin de choisir quelles régions sont envoyées au reste de réseau.

Faster RCNN utilise le même principe. Par contre, plutôt que d'utiliser une méthode sélective, Faster RCNN utilise un autre réseau de neurones afin de déterminer quelles régions sont proposées et envoyées vers le reste du RNC pour la classification. Ainsi, le système résultant peut apprendre quelles régions risquent d'être pertinentes en même temps qu'il apprend à classifier les objets d'une image.

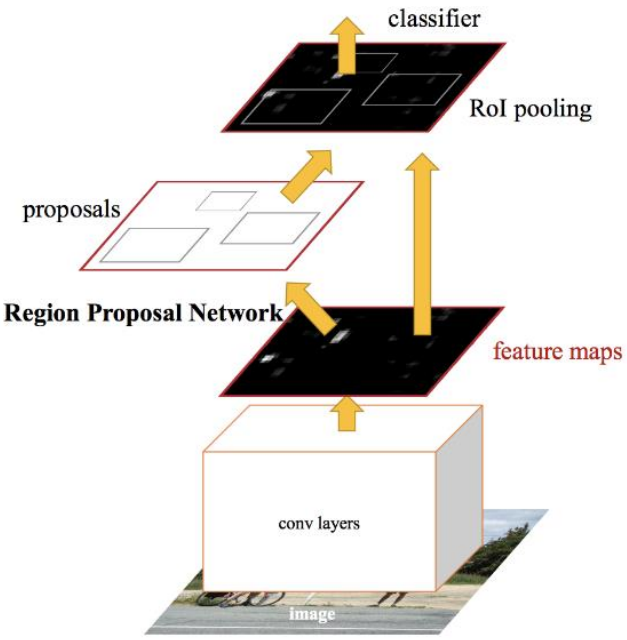


Figure 1.8 Schématisation de Faster RCNN (Tiré de [42])

Tous les algorithmes précédents ont donc une chose importante en commun. Aucun d'eux n'analyse l'image dans son entièreté. En fait, on analyse seulement quelques régions qui sont jugées d'intérêt. L'algorithme *You Only Look Once* (YOLO) prend plutôt l'image d'entrée et la divise en une grille. Pour chaque carreau de la grille, une petite quantité de régions sont générées aléatoirement. Ces régions sont ensuite classifiées par un RNC. Il devient ensuite possible de construire une carte de probabilités d'appartenance à une classe pour l'image entière.

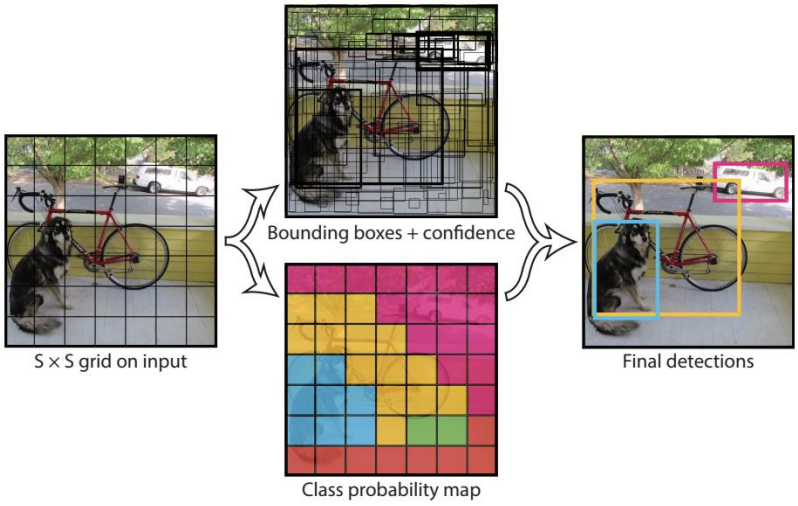


Figure 1.9 Schématisation de YOLO (Tiré de [43])

Le modèle *Single-Shot Detector* (SSD) est semblable à YOLO, c'est-à-dire qu'il n'utilise pas de réseau de proposition de région comme Faster RCNN. Après avoir généré la carte de traits, SSD utilise de simples convolutions 3x3 pour effectuer la classification. Afin d'augmenter sa précision, le modèle effectue ce processus à plusieurs reprises en modifiant la résolution de l'image d'entrée. Avec une résolution plus faible, le modèle peut ainsi détecter des objets plus gros.

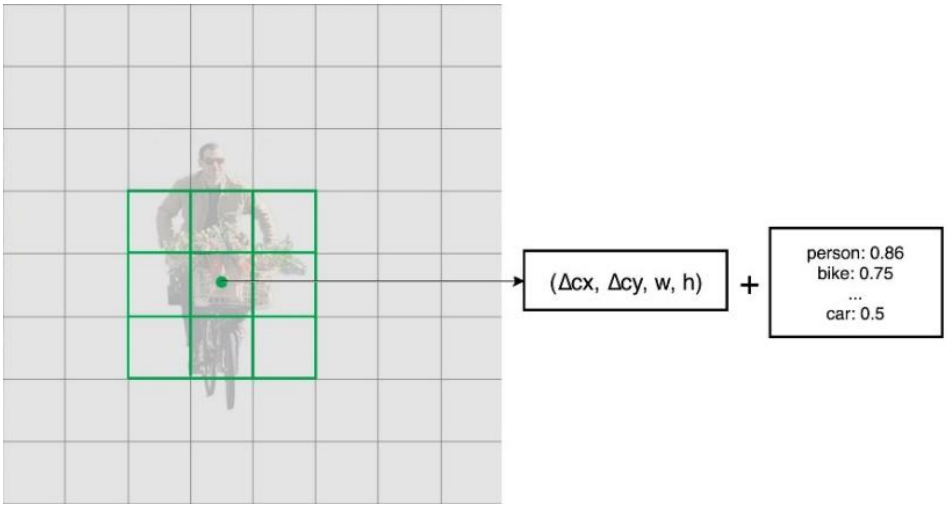


Figure 1.10 Illustration de SSD (Tiré de [46])

1.2.4 Planification de trajectoire et détection d'obstacles

La planification de trajectoire et la détection d'obstacles est une composante nécessaire à tout système robotique. Il existe une multitude de familles d'algorithmes permettant de résoudre ce problème en 2 ou en 3 dimensions. Certains de ces algorithmes sont présentés dans la présente section.

Les algorithmes « insectes » (*bug algorithms*) [48] sont des algorithmes qui suivent une ligne droite entre le robot et la destination. Si un obstacle est rencontré, les contours de l'objet sont suivis jusqu'à ce que l'obstacle soit évité. Si la scène ne change pas, les solutions de cet algorithme sont donc toujours les mêmes. Les champs de potentiel artificiels (*Artificial Potential Field*) [49] se basent sur les principes de l'électromagnétisme. Une charge virtuelle positive est assignée aux objets de la scène, ainsi qu'au robot, alors qu'une charge virtuelle négative est assignée à la destination désirée. On peut ainsi calculer une trajectoire en fonction du gradient des forces potentielles. Cette méthode permet donc d'obtenir une solution optimale, mais est susceptible aux minimums locaux, par exemple si un obstacle est concave.

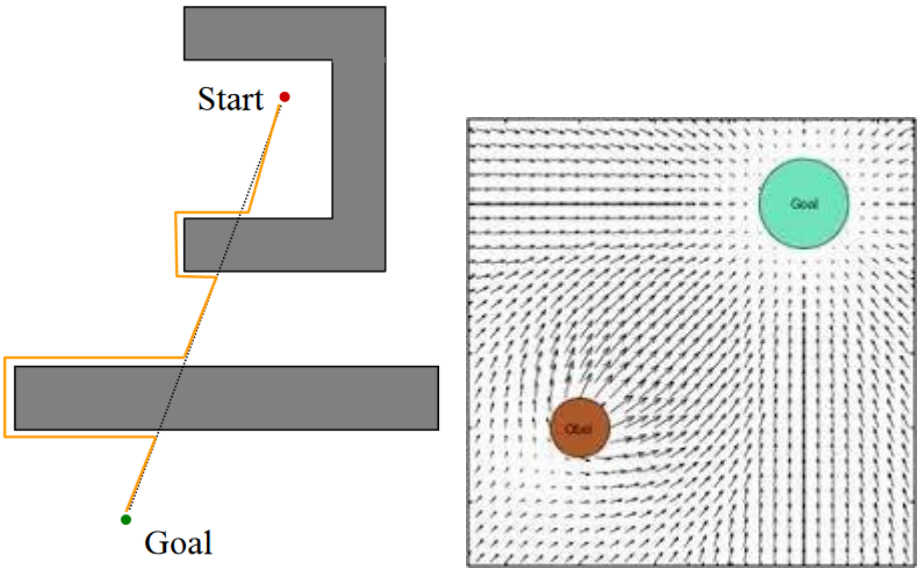


Figure 1.11 Algorithme insecte (gauche) (Tiré de [46]), Champs de potentiel artificiel (droite) (Tiré de [47])

Les histogrammes de vecteurs de champs (*Vector Field Histogram*) [50] se basent sur le principe du sonar. Les obstacles sont ainsi détectés autour de la position initiale du robot, puis un histogramme de la densité d'obstacle en fonction de l'orientation est construit. Le robot prend ensuite la direction ayant la plus faible densité d'obstacle. L'algorithme est donc limité par la vitesse de calcul, car il doit continuellement détecter les obstacles autour du robot. La méthode de recherche d'écart (*Follow the gap*) consiste à trouver l'angle d'écart entre tous les obstacles, puis à choisir le chemin passant par le plus grand écart.

Les algorithmes par échantillonnage utilisent un principe stochastique [51]. On assigne le point de départ ou le point d'arrivée comme point initial de l'algorithme, puis on détermine des vecteurs omnidirectionnels aléatoires à partir de ce point, comme les branches d'un arbre. L'extrémité de ces vecteurs devient un nouveau point de départ, puis on répète le processus jusqu'à atteindre l'objectif. Malgré l'aspect aléatoire, on peut toutefois favoriser certaines directions qui se rapprochent de la cible, par exemple en favorisant les branches qui minimisent la distance euclidienne. Comme la méthode utilise de l'échantillonnage de Monte Carlo, chaque solution est unique, mais il est possible que l'algorithme soit plutôt lent à converger.

Les techniques par algorithme de recherche (*Combinatorial roadmaps and search algorithms*) décomposent d'abord l'espace 3D en formes géométriques simples comme des cubes. Chaque nœud peut être soit occupé s'il contient un obstacle ou le robot, ou libre s'il ne contient rien. On peut alors appliquer un algorithme de recherche (A*, Djikstra, etc.) afin de trouver un chemin passant de l'effecteur vers la cible. Il existe également des versions mélangeant les algorithmes de recherche et les algorithmes par échantillonnage : on échantillonne aléatoirement des points dans l'espace. On peut alors construire une carte où chaque route relie deux points échantillonnés. On peut ensuite appliquer l'algorithme de recherche afin de trouver le chemin optimal.

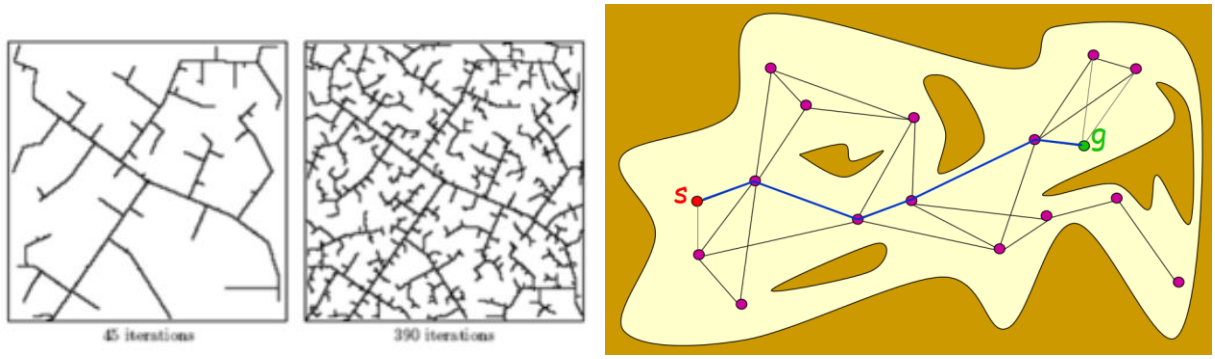


Figure 1.12 Échantillonnage aléatoire (gauche), combinaison d'échantillonnage et de recherche (droite) (Tiré de [51])

Depuis l'avènement de l'apprentissage profond et des réseaux de neurones, des techniques additionnelles ont été développées. Par exemple, des réseaux de neurones ont été utilisés afin de faire de la planification de trajectoire et de l'évitement d'obstacle. Aussi, encore plus récemment, l'apprentissage par renforcement a commencé à susciter un réel intérêt en robotique. Il s'agit de techniques basées sur le principe d'entraînement d'un agent par essai-erreur. On associe ainsi une récompense à chaque essai de l'agent, jusqu'à ce que celui-ci apprenne à définir des trajectoires. [50, 52-54]

1.2.5 Contrôle et cinématique

La cinématique d'un bras robotique permet de représenter la relation entre les angles relatifs de chacun des joints d'un robot, et la position et l'orientation absolue de son extrémité (aussi appelé effecteur). Il s'agit donc d'un concept nécessaire en robotique.

Il existe 2 types principaux de cinématique : la cinématique directe et la cinématique inverse. La cinématique directe est la plus simple pour les bras robotiques sériels ; elle permet de déterminer la pose de l'effecteur en connaissant les angles articulaires du robot. La cinématique inverse, qui est plus complexe pour les bras robotiques sériels, consiste à effectuer l'opération inverse : déterminer les angles des joints qui permettent d'atteindre une pose connue. La Figure 1.13 présente une représentation schématique des cinématiques directe et inverse.

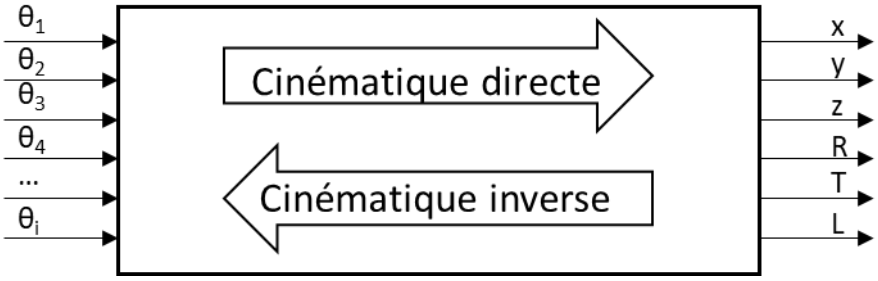


Figure 1.13 Représentation schématique des cinématiques

Où θ représentent les valeurs des angles des joints du bras robotique et xyz/RTL sont respectivement la position cartésienne et l'orientation selon la nomenclature du Roulis/Tangage/Lacet (aussi appelés *Roll/Pitch/Yaw*), par exemple. Il est à noter que l'orientation peut être exprimée selon n'importe quelle autre terminologie.

Afin de passer de l'un à l'autre, on utilise des matrices de transformation permettant de déterminer la position de chacun des joints par rapport au joint précédent. En multipliant chacune de ces matrices, on obtient la matrice de transformation permettant de passer du repère 0 au repère n.

$$T_0^n = \begin{bmatrix} R_0^n & P_0^n \\ 0 & 1 \end{bmatrix} \quad (1)$$

Où R_0^n est une matrice de rotation 3x3 et P_0^n est un vecteur représentant la position de l'effecteur dans le système de coordonnées de base. Afin de déterminer les matrices de transformation, plusieurs méthodes peuvent être utilisées. La plus courante est la méthodologie de Denavit-Hartenberg [55-57].

On obtient donc la pose de l'effecteur avec la cinématique directe f(θ)

$$T_0^n = f(\theta) \quad (2)$$

À l'inverse on obtient les valeurs articulaires en utilisant la cinématique inverse

$$\theta = f^{-1}(T_0^n) \tag{3}$$

Une fois les matrices de transformation obtenues, la cinématique directe est généralement simple à implémenter. La cinématique inverse, toutefois, demeure un défi. Effectivement, comme plusieurs valeurs d'angles peuvent permettre d'atteindre une même pose, on a alors plusieurs solutions permettant d'atteindre une même position avec plusieurs orientations.

Diverses méthodes ont été proposées dans la littérature afin de pallier ce problème.

1.2.5.1 Approche analytique générale

Si la configuration du bras robotique est assez simple, il est possible de déterminer des équations permettant de calculer les valeurs articulaires qui résolvent le problème de cinématique inverse. Dans cette situation, on peut tout de même obtenir plusieurs configurations d'angles possibles. Il est alors usuel de soit utiliser la réponse la plus simple, soit d'appliquer des contraintes supplémentaires sur les valeurs de chaque joint afin de rejeter certaines solutions.

1.2.5.2 Approche analytique par poignet sphérique

Dépendamment de la configuration du bras robotique, il est possible d'appliquer une astuce mathématique afin de simplifier le problème de cinématique inverse. Effectivement, lorsque les systèmes d'axes des 3 derniers joints d'un bras robotique se croisent en un point, on a un bras avec un poignet dit « sphérique ». Alors qu'un bras avec un poignet non-sphérique ne possède souvent pas de solution fermée, un bras avec un poignet sphérique permet d'effectuer un certain découplage entre la position et l'orientation de l'effecteur. On décompose alors le problème de cinématique inverse en 2 problèmes plus simples et il devient possible d'obtenir une forme analytique au problème, ce qui est préférable [58-60]. Le solveur IKFast inclut avec *MoveIt!* utilise une approche analytique par poignet sphérique.

1.2.5.3 Approche numérique par optimisation

La solution traditionnelle au problème de cinématique inverse pour un bras ayant un poignet non-sphérique a traditionnellement été d'employer une approche par optimisation, c'est-à-dire qu'on cherche à minimiser la distance entre la pose actuelle de l'effecteur et la pose désirée. Autrement

dit, nous cherchons à varier les valeurs articulaires de manière à minimiser la distance cartésienne entre les 2 positions, ainsi que la différence angulaire entre les 2 orientations.

Soit *x* la pose de l'effecteur, g*(θ)* une fonction des valeurs articulaires permettant d'atteindre cette pose et *J(θ)* la jacobienne du système

$$x = g(\theta) \rightarrow \frac{\partial x}{\partial \theta} = \frac{\partial g(\theta)}{\partial \theta} = J(\theta) \quad (4)$$

On peut décrire la distance entre la pose désirée et la pose réelle par la fonction de coût F

$$F = \frac{1}{2}\left(x_{désiré} - g(\theta)\right)^T (x_{désiré} - g(\theta)) \quad (5)$$

On peut alors utiliser divers algorithmes d'optimisation, tels que des algorithmes par descente de gradient, des algorithmes évolutifs comme un algorithme génétique, ou des méthodes plus spécifiques comme la pseudo-inverse de la jacobienne afin de minimiser la fonction F [61].

Certaines librairies, comme la librairie *MoveIt!* de ROS, offrent plusieurs méthodes d'optimisation. On y retrouve par exemple KDL qui utilise la méthode de Newton [62] afin de converger, ou TRAC-IK qui utilise la méthode de Newton en parallèle avec une méthode par approche non-linéaire (Optimisation quadratique successive) [63, 64].

1.2.5.4 Approche numérique par réseaux de neurones à propagation avant

Dans les dernières années, les réseaux de neurones, ou l'apprentissage profond en général, ont commencé à être de plus en plus utilisé afin de résoudre des problèmes d'optimisation. Un réseau de neurones consiste en un arrangement d'unités individuelles appelées neurones [65]. Ces unités sont disposées en ce qu'on appelle des *étages* ou des *couches* (ou *layer* en anglais), chaque étage ayant un certain nombre de neurones. Ces étages sont ensuite enchaînés l'un après l'autre.

On retrouve différents types d'étages : l'étage d'entrée (*input layer*), les étages cachés (*hidden layers*) et l'étage de sortie (*output layer*). Le premier, comme son nom le suggère, reçoit l'information à traiter. Le dernier étage permet de recueillir l'information apprise. Enfin, les seconds constituent la partie la plus importante du réseau. Il s'agit des unités qui apprennent à traiter l'information pour obtenir la bonne information de sortie. Dans le cas d'un réseau de

neurones classique, la connexion entre deux neurones de différentes couches est caractérisée par un biais et un poids.

$$sortie = \left(\sum_{i=1}^{N} w_i x_i\right) + b \tag{6}$$

Où N est le nombre d'entrées au présent neurone

w_i représente le poids de la connexion entre l'entrée i et le présent neurone

x_i est l'information de la connexion i

b est le biais

L'entraînement d'un réseau de neurones se fait en deux parties. La première est appelée la propagation avant (*feedforward network*): on envoie des données en entrée, puis, en utilisant l'expression ci-dessus pour chaque neurone, on calcule la sortie associée à cette entrée avec les paramètres actuels.

La deuxième partie est appelée propagation arrière (*backpropagation*). On compare la sortie obtenue lors de la propagation avant à la sortie attendue. On utilise ensuite la différence entre les deux pour ajuster, petit à petit, et exemple après exemple, les poids et les biais de chaque neurone.

Plusieurs articles se sont penchés sur la question de la cinématique inverse en utilisant des réseaux de neurones à propagation avant [66-70]. Le principe consiste à fournir une grande quantité de positions/orientations en entrée au réseau et en sortie les configurations angulaires correspondantes pour que le système apprenne à déterminer la configuration lorsqu'il reçoit une pose en entrée.

1.2.5.5 Approche numérique par apprentissage par renforcement

Depuis 2015, l'apprentissage par renforcement a commencé à être très utilisé en robotique. Le principe est assez simple : le système apprend à contrôler le bras robotique par essai-erreur. Autrement dit, on fournit un environnement à l'agent qui apprend. Celui-ci effectue alors une action comme bouger le 3^{e} joint de 1 radian vers la droite. Une « récompense » est alors calculée en fonction de la nouvelle pose de l'effecteur. Le système cherche donc à apprendre les actions qui lui permettent de maximiser sa récompense [71].

CHAPITRE 2 OBJECTIFS ET QUESTIONS DE RECHERCHE

2.1 Objectifs

L'objectif principal du projet est de développer un système intelligent d'assistance au repas en asservissant un bras robotique de réadaptation.

Cet objectif se divise en 3 sous-objectifs :

SO1 : Concevoir un système de vision capable de reconnaître et détecter la position de la nourriture dans une assiette.

SO2 : Calculer la trajectoire pour un bras robotique de réadaptation à 6 degrés de liberté et résoudre sa cinématique inverse.

SO3 : Tester le système en simulation avec une scène réelle.

2.2 Questions de recherche

Atteindre l'objectif de recherche nous permettra de répondre à la question de recherche qui suit :

Peut-on automatiser le processus d'alimentation d'une personne à mobilité réduite ?

Par processus d'alimentation, on entend repérer la nourriture, la saisir et l'amener à la bouche de l'utilisateur.

CHAPITRE 3 MÉTHODOLOGIE

La section actuelle vise à présenter le processus utilisé afin de développer le système autonome d'assistance au repas. Le cahier des charges sera d'abord présenté afin d'aborder les aspects essentiels du projet. Ensuite, les différents choix techniques seront mis de l'avant : structure du système, simulation du système, éléments de vision par ordinateur, planification de trajectoire et cinématique inverse, et tous les autres décisions prises. Enfin, les détails d'implémentation seront présentés, suivis par une description des tests effectués.

3.1 Cahier des charges

L'objectif du projet étant d'établir un système intuitif d'assistance au repas pour une personne à mobilité réduite, la première étape consiste donc à étudier le fonctionnement normal du processus d'alimentation. On peut alors déterminer quelles sont les étapes importantes à répliquer avec le système robotique.

Il est possible de déconstruire la tâche d'alimentation d'une personne normale en cinq grandes étapes [72] présentées dans la colonne de gauche de la Figure 3.1. Le système robotique à développer devra ainsi permettre de répliquer ces étapes. La dernière colonne de la Figure 3.1 illustre les étapes correspondantes effectuées par un système robotique.

3.1.1 Survol du processus d'alimentation

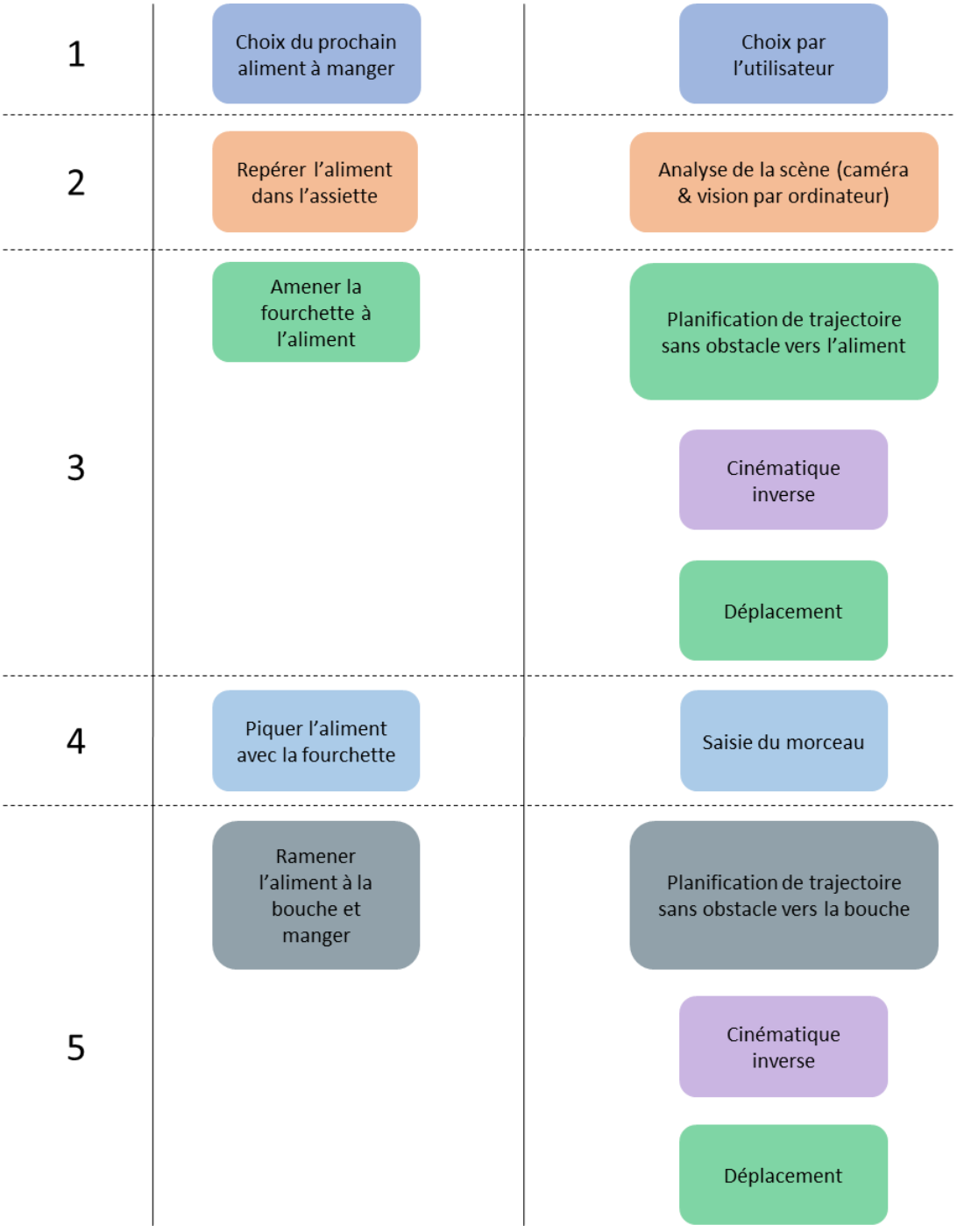


Figure 3.1 Comparaison des processus d'alimentation réel et robotique

3.1.2 Fonctions du système

On définit ici les fonctionnalités du système qui doivent absolument être présentes pour que celui-ci puisse répondre aux objectifs du projet. On définit les fonctions suivantes :

1. Repérer des objets (nourriture) d'intérêt dans la scène (Figure 3.1, étape 2)
2. Approcher et saisir la nourriture
3. Éviter les obstacles dans la scène
4. Ramener la nourriture vers l'utilisateur

Également, on peut identifier certaines sous-tâches que le système doit être en mesure d'accomplir afin d'atteindre les sous-objectifs du projet. Celles-ci sont présentées au Tableau 3.1.

Tableau 3.1 Sous-tâches à accomplir

SO1 : Concevoir un système de vision capable de reconnaître et détecter la position de la nourriture dans une assiette.
1.1. Prendre une image de la scène (couleur et profondeur).
1.2. Utiliser un algorithme de reconnaissance d'objet pour reconnaître la nourriture et sa position dans le référentiel de la caméra.
1.3. Convertir les coordonnées 3D du référentiel de la caméra vers le référentiel du bras robotique.
SO2 : Effectuer un calcul de trajectoire pour un bras robotique de réadaptation à 6 degrés de liberté et résoudre sa cinématique inverse.
2.1. S'assurer que l'algorithme sélectionné est en mesure de résoudre la cinématique inverse d'un bras robotique à 6 DDL avec poignet non-sphérique.
2.2. Être en mesure d'imposer un angle d'approche de la nourriture.
2.3. Ajouter un système d'évitement d'obstacles.
SO3 : Tester le système en simulation avec une scène réelle.
3.1. Créer un environnement de simulation communiquant avec les autres modules

3.2 Structure adoptée

Le logiciel structurel sélectionné dans le cadre de ce projet est ROS. Effectivement, ROS offre les avantages suivants :

1. Offre une documentation détaillée et abondante
2. Simplifie la structure du projet en regroupant chaque module sous un seul logiciel

3. Facilite la communication entre les modules du projet
4. Présente une multitude de librairies pertinentes au projet :
    a. Interface pour les caméras stéréo & librairies de traitement d'image
    b. *MoveIt!* pour la planification de trajectoire et l'évitement d'obstacles
    c. Compatibilité avec une multitude de bras robotiques d'assistance
5. Se retrouve dans de plus en plus de projets de recherche en robotique : Près de 5000 citations de ROS dans des publications scientifiques en 2018 seulement

De plus, le bras robotique de réadaptation utilisé est le bras MICO de Kinova (Figure 3.2). Il s'agit d'un bras adapté aux chaises roulantes pour les personnes souffrant d'un handicap du haut du corps. Il est léger et consomme peu d'énergie, en plus d'être un bras à 6 DDL, ce qui signifie qu'il peut atteindre n'importe quel point dans son espace de travail avec n'importe quelle orientation. Il s'agit également d'un bras fréquemment utilisé et disponible dans le laboratoire de recherche des professeurs Sofiane Achiche et Maxime Raison.

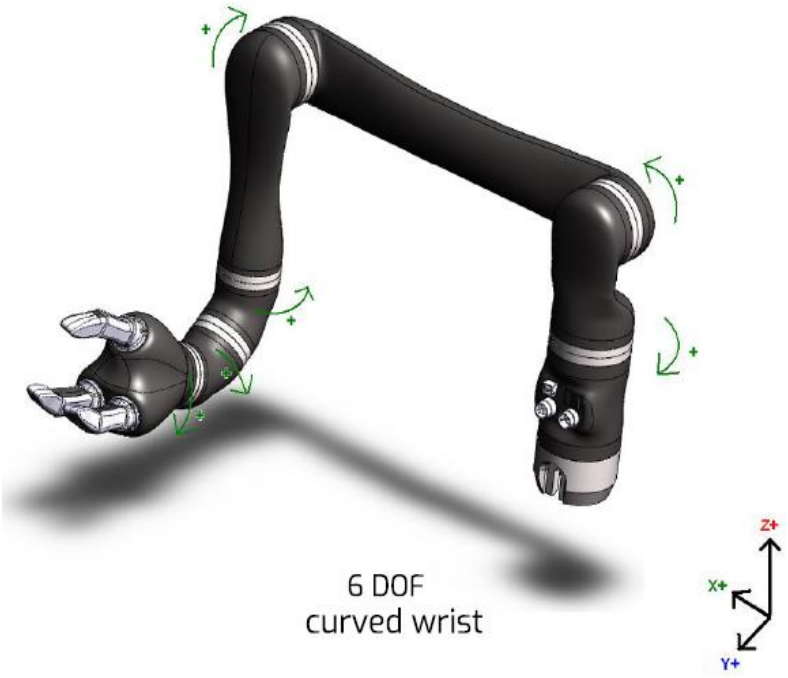


Figure 3.2 Bras MICO à 6 degrés de liberté de Kinova (Tiré de [23])

L'utilisation de ROS a également permis de simplifier le changement de référentiel des coordonnées du système. Effectivement, comme la position relative des objets d'intérêts détectés par la caméra est différente de la position par rapport au bras robotique, il est nécessaire d'effectuer un changement de référentiel. La librairie *tf* [73] permet d'effectuer ce changement de manière simple et efficace en gardant en mémoire la relation entre les différents référentiels. Ainsi, afin d'exprimer des coordonnées dans un référentiel différent, il suffit de fournir les coordonnées initiales, le référentiel de base et le référentiel cible (Voir Annexe A).

3.3 Simulation du système

ROS offre également une compatibilité avec le simulateur *Gazebo* [74], et Kinova offre un modèle du bras MICO qui peut être utilisé avec ROS et Gazebo. Ainsi, dans le cadre du projet, un bras simulé avec Gazebo a été utilisé (Figure 3.3). Plusieurs raisons ont motivé ce choix :

1. L'utilisation d'une simulation accélère le processus de développement, car le simulateur peut utiliser un référentiel de temps plus rapide que le monde réel. Par exemple, un déplacement qui prend 15 secondes en temps réel pourrait en prendre 5 en simulation, ce qui permet de réduire le temps de développement.
2. Un simulateur neutralise le risque d'endommager le bras utilisé.
3. Les robots simulés avec Gazebo respectent la physique du monde réel, c'est-à-dire que leur masse, leur inertie, et leurs autres paramètres physiques sont respectés. Aussi, les forces réelles comme la gravité ou le frottement sont simulées. Ainsi, la réponse obtenue en simulation est représentative de la réponse obtenue sur un système réel.
4. Le passage du bras simulé au bras réel est extrêmement simple. En fait, il suffit de changer quelques lignes de code afin de spécifier au système qu'on utilise un bras réel plutôt qu'un bras virtuel.
5. Cette fin de projet se passe en plein cœur de la pandémie COVID-19, il a donc été jugé plus sécuritaire d'utiliser un bras simulé afin d'éviter les déplacements non nécessaires au laboratoire de Polytechnique Montréal.

La simulation du système avec Gazebo permet donc de répondre au SO3.

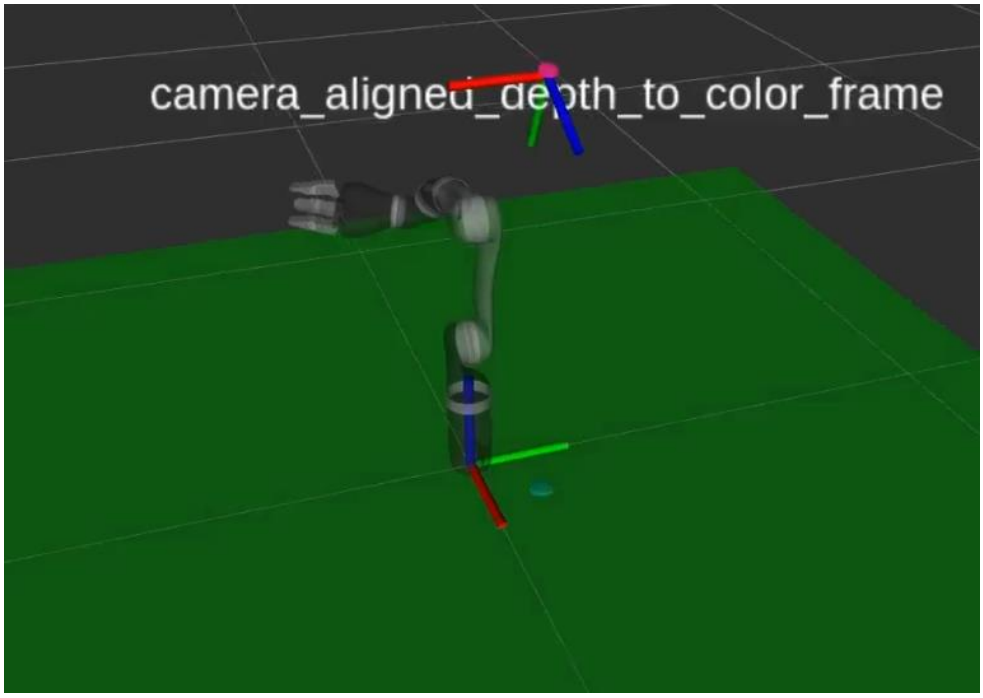


Figure 3.3 Simulation montrant la position du bras et de la caméra

3.4 Vision par ordinateur

Les choix effectués en lien avec la vision par ordinateur servent à répondre au SO1

3.4.1 Choix de caméra

La caméra sélectionnée pour le projet est la caméra D415 d'Intel. Effectivement, celle-ci offre une meilleure polyvalence au niveau des informations qu'elle fournit (stéréo, infrarouge, nuages de points) et de sa compatibilité avec ROS. Aussi, il existe une documentation abondante et riche pour son utilisation. Enfin, cette caméra était disponible au laboratoire de recherche du professeur Sofiane Achiche.

Tableau 3.2 Résumé des caractéristiques des caméras

	Kinect	**Tara**	**D415**
Prix($CAD)	135	336	182
Dimensions(mm)	249 x 66 x 67	100 x 30 x 35	99 x 20 x 23
Résolution(pixels)	1920 x 1080 à 30 fps	1504 x 480 à 60 fps	1920 x 1080 à 30 fps
Portée(m)	0.5 – 4.5	0.5 – 3.0	0.16 – 10.0
Calibration	Non nécessaire	Une seule fois	Une seule fois
Types d'images	Images couleur & nuage de points (infrarouge)	Images couleur & nuage de points (stéréo)	Images couleur & nuage de points (stéréo et infrarouge)
Compatibilité	Simple à utiliser en développement ou avec ROS	Simple à utiliser en développement ou avec ROS	Simple à utiliser en développement ou avec ROS

3.4.2 Choix d'algorithme de reconnaissance d'image

L'algorithme de reconnaissance d'objets utilisé est basé sur une combinaison de Faster RCNN, avec l'architecture élaborée pour le ResNet à 50 couches de neurones (ResNet50)**.** Il s'agit d'une implémentation opensource en Tensorflow offerte par Google [75]. Les algorithmes offerts sont entraînés sur l'ensemble de données COCO [76], qui est une base de données regroupant plus de 330 000 images de 80 classes d'objets différents, incluant des personnes, des voitures, ou même des objets de la vie courante comme des carottes ou des pommes. Il est d'ailleurs possible d'ajouter de nouvelles catégories personnalisées au modèle. L'algorithme Faster RCNN a également l'avantage d'offrir un excellent équilibre entre vitesse d'exécution et précision. Effectivement, selon la documentation offerte sur 25 modèles offerts, celui-ci est dans le top 5 des plus performants au niveau de la précision(mAP), tout en étant, au moment du choix final d'algorithme, le plus rapide de ces 5 modèles. Il est à noter que le modèle 3 n'était pas disponible au moment du développement de cette partie du projet.

Tableau 3.3 Top 5 des algorithmes de reconnaissance d'objet

Modèle	**Métrique de performance (mAP)**	**Temps d'analyse pour une image (ms)**
Faster RCNN (NAS)	43	1833
Faster RCNN (Inception Resnet)	37	620
SSD (Resnet50, FPN)	35	76
Faster RCNN (Resnet101)	32	106
Faster RCNN (Resnet50)	30	89

3.4.3 Fonctionnement de Faster RCNN

La Figure 3.4 illustre les principaux éléments de Faster RCNN.

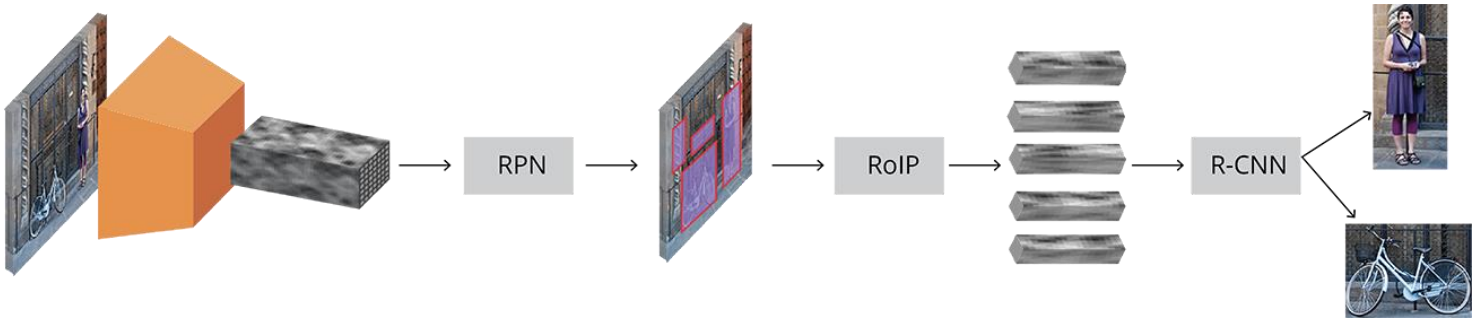


Figure 3.4 Fonctionnement de Faster RCNN (Tiré de [77])

On retrouve d'abord le bloc orange, qui représente l'extracteur de caractéristiques (*feature extractor*) de l'image. Il s'agit d'un sous-modèle de réseaux de neurones convolutifs qui est entraîné sur la base de données de COCO afin de repérer les caractéristiques d'une image qui sont les plus propices à la reconnaissance d'objet. L'extracteur de caractéristique fournit alors une carte convolutive des caractéristiques principales de l'image. On pourrait par exemple imaginer que l'extracteur repérerait les contours ou les régions saillantes d'une image.

Cette carte est transmise à la prochaine étape du réseau, le réseau de proposition de régions (*region proposal network – RPN*). Ce réseau est également entraîné afin de prendre en entrée une carte convolutive de caractéristiques et de retourner en sortie les régions de l'image qui sont les plus susceptibles de contenir des objets. On impose d'ailleurs un maximum de régions que l'algorithme peut fournir en sortie, ce qui permet de proposer uniquement les régions ayant de fortes probabilités de contenir un objet.

L'étape suivant consiste à la mise en commun des régions d'intérêt (*region of interest pooling – RoIP)*. Pour chaque région d'intérêt obtenue à l'étape suivante, l'algorithme effectue un échantillonnage afin d'obtenir les caractéristiques les plus importantes de cette région. L'objectif de cette étape est d'uniformiser la taille des régions afin d'être compatible avec le réseau RCNN.

Ces caractéristiques sont enfin transmises au réseau RCNN qui effectue la classification des objets présents dans les régions d'intérêt [77].

3.5 Choix de planificateur de trajectoire et évitement d'obstacles

En supplément à ROS et au bras robotique de Kinova, le planificateur de trajectoire *MoveIt!* [78], compatible avec ROS et cité plus de 250 fois depuis sa création, a été utilisé. *MoveIt!* fonctionne en complément à ROS et offre une multitude de fonctionnalités permettant de résoudre les problèmes de cinématique, de planification de trajectoire, d'évitement d'obstacles et de perception 3D. Autrement dit, le système offert par *MoveIt!* permet de gérer la partie concernant le contrôle du robot.

Au niveau de la planification de trajectoire, *MoveIt!* offre plusieurs planificateurs [79] :

1. Open Motion Planning Library (OMPL) : Planificateur par échantillonnage stochastique
2. Stochastic Trajectory Optimization for Motion Planning (STOMP) : Planificateur combinant les méthodes d'optimisation et d'échantillonnage stochastique
3. Search-Based Planning Library (SBPL) : Planificateur par algorithme de recherche
4. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) : Planificateur par optimisation par gradient

Pour ce projet, le planificateur par défaut OMPL a été utilisé. Parmi les quatre planificateurs ci-dessus, OMPL est le plus rapide et réduit donc le temps passé à l'étape de planification de

trajectoire. Effectivement, en comparaison, STOMP est plus lent puisqu'il ajoute une composante d'optimisation à l'échantillonnage stochastique. SBPL, lui, a le potentiel d'être plus rapide qu'OMPL. Toutefois, l'intégration de ce planificateur à *MoveIt!* n'est pas entièrement complétée et a donc plus de chances de présenter certains *bugs*. Enfin, CHOMP est également encore en cours d'intégration à *MoveIt!*. De plus, comme CHOMP effectue des calculs de gradient, il est aussi plus lent qu'OMPL. OMPL est d'ailleurs le planificateur par défaut pour la configuration *MoveIt!* offerte par Kinova.

Pour ce qui a trait à l'évitement d'obstacles, une combinaison d'une fonctionnalité de la caméra Intel Realsense avec une autre librairie ROS a été utilisée. Effectivement, la caméra stéréo peut fournir un nuage de points représentant la disposition en trois dimensions de la scène observée. La librairie *Octomap* [80], elle, permet de transformer un nuage de points en obstacles d'une scène. Ainsi, en combinaison avec les données fournies par la caméra d'Intel, il est possible de fournir les coordonnées en trois dimensions des obstacles à éviter à *MoveIt!* [81]. *MoveIt!* s'occupe alors de planifier une trajectoire évitant ces obstacles avec le planificateur choisi précédemment.

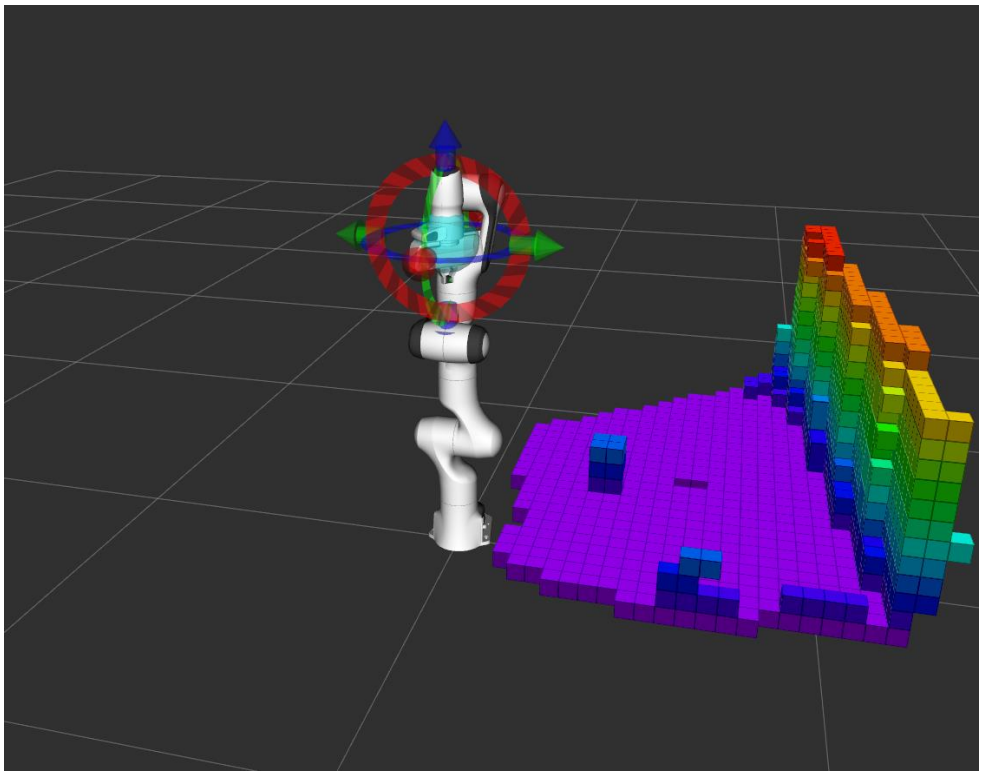


Figure 3.5 Exemple de scène construite à partir d'un nuage de points (Tiré de [81])

Quant au problème de cinématique inverse, Kinova fournit une adaptation du solveur TRAC-IK [82]. Ce solveur est une alternative aux solveurs par inversion de jacobienne. En fait, TRAC-IK combine l'algorithme classique d'inversion de jacobienne par la méthode de Newton à une approche d'optimisation non-linéaire, et exécute les deux algorithmes en parallèle. Dès qu'un des deux algorithmes converge, la solution trouvée est utilisée.

Ainsi, les algorithmes de planification de trajectoire et de résolution de la cinématique inverse permettent de répondre au SO2.

3.6 Autres choix et simplifications

Enfin, afin de permettre au projet d'aboutir dans des délais réalistes, certains autres choix ont dû être faits.

1. La méthode de sélection d'un objet dans la scène se fait actuellement de manière programmable, c'est-à-dire qu'il s'agit d'une variable à changer dans le code à exécuter.
2. Lorsque plusieurs objets de la même classe se retrouvent dans la scène et que l'utilisateur veut en saisir un, l'objet le plus proche de la base du bras robotique sera toujours celui saisi.
3. L'effecteur sélectionné est une fourchette. Effectivement, il a été déterminé qu'un ustensile serait utilisé afin d'éviter de salir le bras robotique et de simplifier la tâche de préhension. Toutefois, une cuillère permet difficilement de saisir de la nourriture solide, et requiert un système de stabilisation et de minimisation des vibrations beaucoup plus avancé dues aux déplacements de l'effecteur d'une position à l'autre.
4. Suite à la saisie de l'aliment à manger, l'effecteur le ramène vers une position constante qui est près de l'emplacement où se trouverait la bouche de l'utilisateur. Cette simplification réduit le temps de calcul et évite d'avoir deux caméras (ou une caméra mobile).
5. Comme on désire utiliser des images réelles avec le bras simulé, il a fallu assigner une position arbitraire à la caméra par rapport au bras robotique. Celle-ci a été placée au-dessus du bras, avec une translation latérale et une orientation de 60 degrés par rapport à l'horizontale de manière à simuler une caméra située au-dessus de l'épaule de l'utilisateur. Ce choix a peu d'impact sur les résultats du système; tant que la position et l'orientation de

la caméra par rapport au bras robotique est connue, il est possible d'effectuer le changement de référentiel sans problème.

3.7 Résumé des choix

La Figure 3.6 présente un résumé des choix effectués dans le cadre du projet, ainsi que l'étape de la tâche d'alimentation qui leur est associée. La mise en commun des modules est gérée par ROS. La Figure 3.7 présente un schéma de la manière dont les différentes parties du système sont reliées.

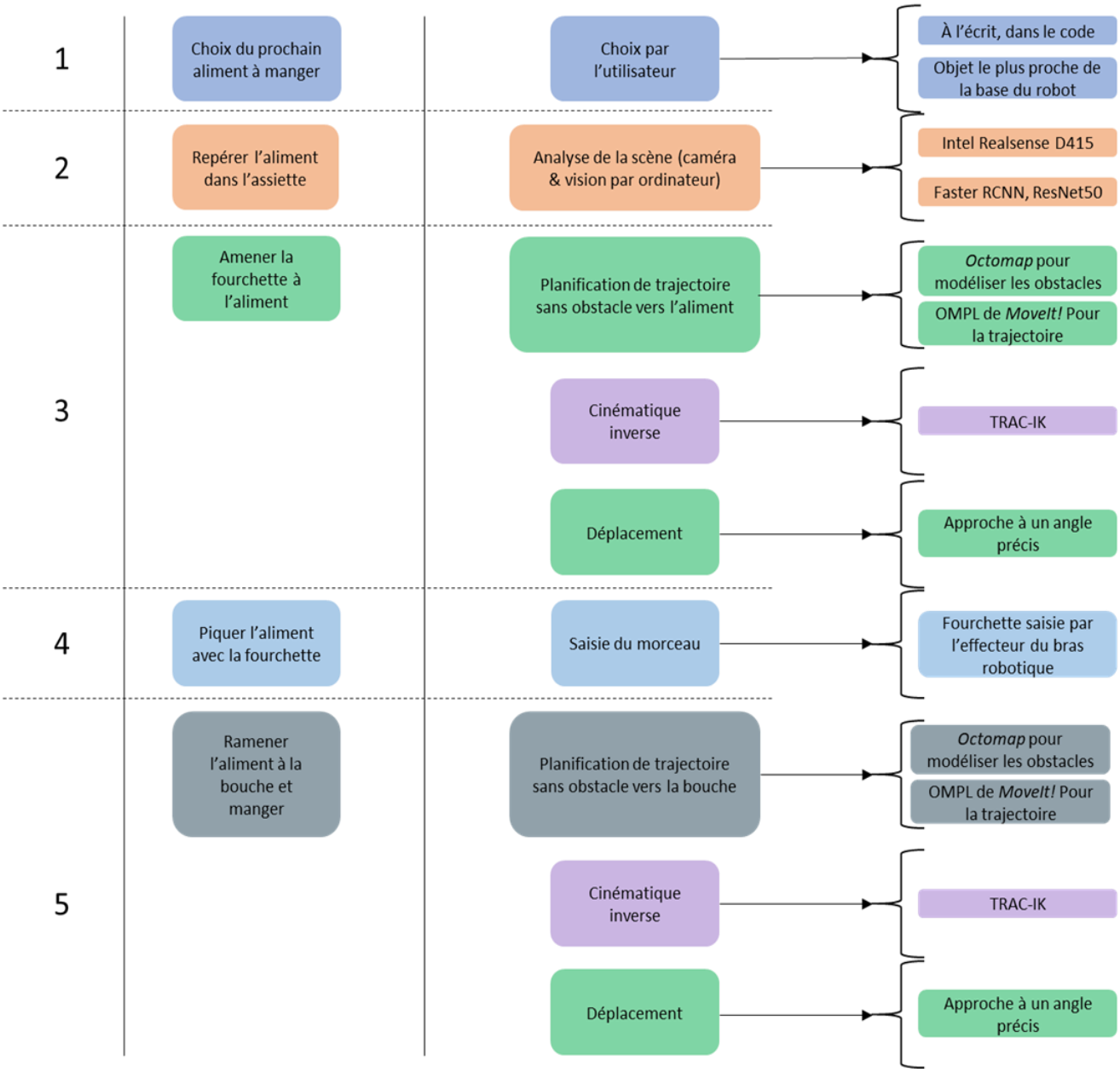


Figure 3.6 Détails des choix effectués et leur étape associée

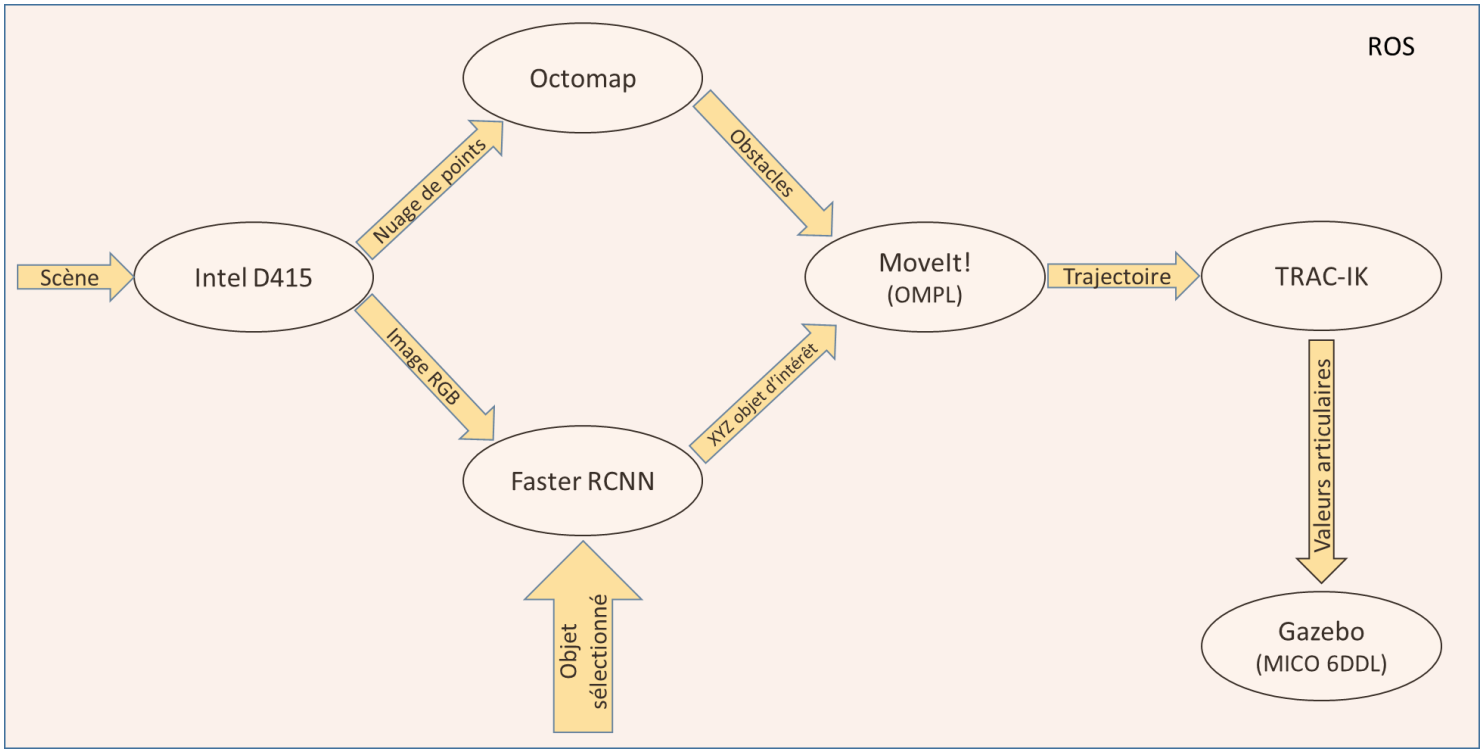


Figure 3.7 Schématisation du système global

La Figure 3.7 présente les différents modules du système final, et schématise le flot de données d'un module à un autre.

3.8 Implémentation et intégration

3.8.1 Détection d'objet

La première étape de fonctionnement du système consiste à analyser la scène à l'aide de la caméra Intel D415. On prend d'abord une photo en couleur de la scène. Cette image est ensuite redimensionnée vers 300 pixels par 300 pixels. Cette nouvelle dimension permet à l'image d'être analysée par l'algorithme de reconnaissance d'images Faster RCNN. L'algorithme fournit alors en sortie la position dans l'image, en termes de pixels, de tous les objets du type désiré. Par exemple, si on recherche des carottes, on aura en sortie une liste de toutes les carottes trouvées avec leur position. Cette liste est ordonnée en ordre de confiance, c'est-à-dire que plus l'algorithme est certain qu'il s'agit d'une carotte, plus la confiance est élevée. Il est d'ailleurs possible de modifier le seuil limite que l'algorithme utilise afin d'afficher les objets. Dans le cadre du projet, un seuil de confiance de 90% a été utilisé.

Une fois qu'on a la position (X_{pixel}, Y_{pixel}) dans l'image des objets d'intérêt, il reste à traduire ces coordonnées en coordonnées du monde réel ($X_{caméra}$, $Y_{caméra}$, $Z_{caméra}$), selon le référentiel de la caméra. On commence d'abord par trouver la distance entre l'objet et la caméra en utilisant la carte de profondeur générée par la caméra D415. En utilisant les coordonnées du centre géométrique de l'objet trouvé, cette carte permet d'obtenir la distance en mètres entre le foyer optique de la caméra et l'objet dans l'image.

On peut ensuite utiliser une déprojection afin de passer de (X_{pixel}, Y_{pixel}, $Z_{caméra}$) à ($X_{caméra}$, $Y_{caméra}$, $Z_{caméra}$). L'annexe B présente les détails pour la déprojection avec la caméra Intel Realsense D415.

$$(X_{pixel}, Y_{pixel}, Z_{caméra}) \xrightarrow{Déprojection} (X_{caméra}, Y_{caméra}, Z_{caméra}) \tag{7}$$

Également, en utilisant la carte de profondeur et la librairie *Octomap*, on génère une carte de tous les objets et les obstacles présents dans la scène. Dans la carte, ceux-ci sont représentés par des cubes d'un centimètre de longueur.

3.8.2 Planification de trajectoire et cinématique inverse

Une fois les coordonnées des objets d'intérêt trouvées dans le référentiel de la caméra, il reste à les transformer dans le référentiel du bras robotique. On utilise alors la librairie *tf* de ROS (Voir Annexe A).

$$(X_{caméra}, Y_{caméra}, Z_{caméra}) \overset{tf}{\Rightarrow} (X_{robot}, Y_{robot}, Z_{robot}) \tag{8}$$

Si plusieurs objets de la classe recherchée sont présents dans la scène, on calcule alors la distance euclidienne entre chaque objet et la base du robot, et on sélectionne l'objet le plus proche. Par définition du changement de référentiel effectué, les coordonnées de la base du robot sont (0, 0, 0).

$$distance = \sqrt{(X_{robot} - 0)^2 + (Y_{robot} - 0)^2 + (Z_{robot} - 0)^2} \tag{9}$$

On veut toutefois que le système robotique approche la nourriture selon un angle θ_{cible} précis afin de faciliter l'action de piquer la nourriture avec la fourchette tenue par l'effecteur. Cet angle a été choisi de façon arbitraire afin d'être semblable à l'angle d'approche qu'un être humain utilise. L'angle de RTL en degrés (0, 230, 110) a été utilisé. En connaissant l'orientation souhaitée pour l'approche, on peut imaginer un vecteur $\vec{v}$ de longueur égale à la longueur de la fourchette et

d'orientation égale à θ_{cible}. En soustrayant ce vecteur à la position de l'objet désiré dans la scène, on obtient une position intermédiaire à laquelle le robot doit se rendre avant d'approcher l'objet.

$$(\mathrm{X}_{robot}, \mathrm{Y}_{robot}, \mathrm{Z}_{robot}) - \vec{v} = (\mathrm{X}_{intermédiaire}, \mathrm{Y}_{intermédiaire}, \mathrm{Z}_{intermédiaire}) \quad (10)$$

Deux informations sont alors transmises à *MoveIt!*: les coordonnées de la destination intermédiaire et la carte des obstacles générée précédemment. Le planificateur de trajectoire OMPL calcule alors une trajectoire évitant les obstacles afin d'atteindre la position intermédiaire. Une fois que la série de points composant la trajectoire a été générée, les points sont transmis au solveur TRAC-IK, un à la fois. Les angles des joints du bras robotique pour chaque position sont envoyés au bras MICO simulé avec Gazebo afin d'exécuter la trajectoire jusqu'à atteindre la position intermédiaire. On calcule ensuite la cinématique inverse une autre fois afin d'approcher l'objet, tout en maintenant l'orientation spécifiée.

Enfin, OMPL et TRAC-IK sont utilisés une dernière fois afin de calculer la trajectoire et la cinématique inverse permettant de ramener la nourriture piquée vers la position initiale, située près de la bouche de l'utilisateur.

3.9 Tests

Afin de s'assurer du fonctionnement du système, il est nécessaire de tester le système dans son entièreté, ainsi que ses sous-modules. La présente section vise à présenter les tests effectués dans le cadre de ce travail.

3.9.1 Test - Cinématique inverse

Trois solveurs pour la cinématique inverse du bras MICO sont fournis par Kinova afin d'interfacer avec *MoveIt!*.

1. KDL : Approche par la méthode de Newton
2. IKFast : Tente de trouver une solution analytique au problème de cinématique inverse
3. TRAC-IK : Combinaison de la méthode de Newton en parallèle d'une optimisation quadratique successive

Afin d'utiliser le solveur le plus approprié, il est nécessaire de les comparer. Le test proposé consiste à générer aléatoirement 1000 positions désirées pour l'effecteur. En partant de la position d'initialisation du bras robotique, on exécute ensuite chaque solveur pour chacune des 1000 positions. Si l'algorithme arrive à converger vers une solution, l'essai est un succès. On mesure le temps d'exécution pour chaque essai réussi et on impose une limite de temps de 30 secondes pour chaque tentative, afin d'assurer un délai réaliste pour le système robotique.

3.9.2 Test - Planification de trajectoire avec cinématique inverse

Afin de vérifier si la combinaison du planificateur de trajectoire et du solveur de cinématique inverse fonctionne adéquatement, on génère une scène simple avec des obstacles. À partir de la position d'initialisation, on demande ensuite au système d'atteindre une position valide, c'est-à-dire qu'il existe des trajectoires permettant de se rendre au point désiré, avec l'orientation désirée. En répétant le test 100 fois, il est possible d'avoir une bonne idée de la répétabilité de la tâche et, ainsi, de valider si le système est fiable ou non. Le test est un succès si le système arrive à trouver une solution permettant d'atteindre la position désirée en moins de 30 secondes. La Figure 3.8 présente la scène utilisée. L'objectif est d'amener l'effecteur à saisir le cylindre situé derrière les barres horizontales.

La combinaison de TRAC-IK avec OMPL a été utilisée.

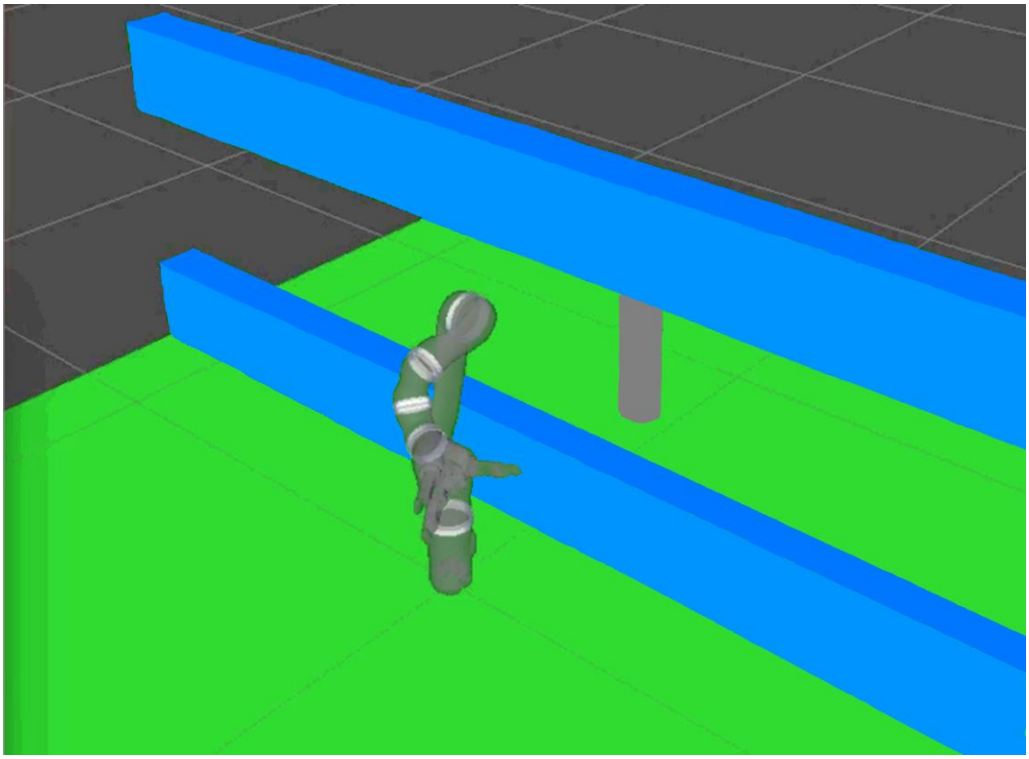


Figure 3.8 Scène avec obstacles

3.9.3 Test - Système complet

Comme l'objectif est d'aller saisir de la nourriture de manière intuitive, on présente le scénario suivant : on construit une scène complexe, où plusieurs carottes et plusieurs brocolis sont disposés dans une assiette devant le système robotique. On demande d'abord au système d'aller saisir une carotte et de la ramener à l'utilisateur, puis d'aller chercher un brocoli et de le ramener.

On observe alors les mesures suivantes :

1. Succès ou échec de la tâche
2. Temps de détection des objets dans la scène
3. Temps de calcul de trajectoire vers l'objet
4. Temps total de la tâche

On définit le succès de la tâche comment étant de réussir à identifier les objets d'intérêt dans la scène et d'être en mesure d'aller les saisir et les ramener vers la bouche de l'utilisateur.

En répétant chaque test 5 fois en gardant la même configuration de la scène, il est possible d'obtenir une idée de la répétabilité de l'exécution. Au début de chaque essai, le système a été complètement réinitialisé. Ainsi, on s'assure que chaque essai est complètement indépendant.

La Figure 3.9 présente la scène d'intérêt.

Figure 3.9 Scène d'intérêt avec carottes et brocolis

CHAPITRE 4 RÉSULTATS & DISCUSSION

4.1 Résultats - Cinématique inverse

Tableau 4.1 Résultats de test - Cinématique inverse

	KDL	**IKFast**	**TRAC-IK**
Nombre de succès	372	642	814
Pourcentage de réussite	37,2%	64,2%	81,4%
Temps moyen pour les succès (secondes)	0, 546	0,118	0,132

Plusieurs informations intéressantes peuvent être relevées à partir du test effectué. La première est qu'aucun des solveurs n'est infaillible. Effectivement, KDL obtient un pourcentage de réussite faible avec 37,2%, alors que IKFast et TRAC-IK performent un peu mieux avec 64,2% et 81,4%, respectivement.

Ces résultats étaient toutefois attendus. Effectivement, KDL utilise la méthode de Newton [83] afin de prendre une approche de résolution de la cinématique inverse par optimisation. Or, la performance de cette méthode est limitée dans le cas de fonctions complexes comme la cinématique inverse d'un manipulateur à 6 DDL. Par exemple, lorsque la dérivée de la fonction à optimiser est difficile à obtenir, l'algorithme l'approxime en calculant la pente de la droite reliant deux points près du point d'intérêt. Or, cette approximation converge plus lentement que la méthode de Newton pure. Également, à cause des contraintes du test (limite de 30 secondes par essai, limite du nombre d'itérations, etc.), il arrive que certains essais qui convergeaient lentement vers une solution soient comptés comme des échecs parce qu'ils prenaient trop de temps. Selon les résultats, IKFast échoue dans près de 35% des cas malgré le fait qu'il est censé offrir une solution analytique au problème de cinématique inverse. L'algorithme tente de déterminer une solution analytique pour la cinématique inverse à partir de la géométrie du bras robotique. Ainsi, le solveur est adapté aux bras robotiques à poignet sphérique, alors que le bras MICO utilisé possède un poignet non-sphérique. L'astuce de résolution qui est utilisée par IKFast ne s'applique donc pas parfaitement au bras utilisé. Autrement dit, le solveur approxime le bras robotique à poignet non-sphérique par un bras

robotique à poignet sphérique, ce qui ne permet pas toujours d'obtenir une représentation réelle du système. Enfin, TRAC-IK utilise également la méthode de Newton, mais effectue également une optimisation quadratique successive en parallèle. Cette méthode est donc la plus robuste dans le cas d'un système complexe[84] et permet d'obtenir une solution dans la majorité des cas.

Également, une partie du taux d'erreur des solveurs est reliée aux limitations au niveau des valeurs articulaires de bras MICO. En effet, comme les approches convergent tranquillement vers une solution, il est possible qu'ils convergent vers une solution qui doit en réalité être rejetée parce qu'elle contient des angles invalides. Aussi, comme le MICO est un bras à 6 DDL, il possède certaines singularités, c'est-à-dire certaines configurations où l'effecteur devient bloqué dans certaines directions. On retrouve par exemple une singularité si deux joints successifs sont alignés. Ces situations peuvent donc être rencontrées par les solveurs et les solutions doivent être rejetées.

Ces taux de succès nous laissent donc présumer qu'il est possible que, dans le système réel, certaines configurations posent problème au solveur et que le système n'arrive pas à calculer la cinématique inverse. Également, on remarque que IKFast et TRAC-IK sont de 4 à 5 fois plus rapides que KDL, avec des temps moyens d'exécution de 0,118 secondes et 0,132 secondes, comparé à 0,546 secondes pour KDL.

D'ailleurs, le solveur recommandé par Kinova est TRAC-IK. Suite à ce test, il semble que les résultats confirment cette recommandation et que TRAC-IK est le solveur le plus approprié pour le système robotique, car il obtient le pourcentage de réussite le plus élevé, tout en étant à peine 0,015 seconde plus lent qu'IKFast, en moyenne. Évidemment, comme les tests effectués n'incluent pas la totalité des configurations de paramètres possibles, la conclusion précédente sert à guider le choix plutôt qu'à affirmer avec certitude que TRAC-IK est supérieure dans toute situation.

4.2 Résultats - Planification de trajectoire avec cinématique inverse

Tableau 4.2 Résultats de test - Planification de trajectoire avec cinématique inverse

Nombre de succès	76
Nombre d'essais	100
Temps moyen pour les succès (secondes)	0,797

On remarque que le ratio de succès est de 76%. Ce taux de succès signifie que, dans la majorité des cas, le système est en mesure de déterminer une trajectoire et les valeurs articulaires correspondantes afin d'atteindre la position cible selon l'orientation désirée. En fait, selon les observations effectuées dans le cadre de ce test, il semble que la raison pour laquelle le système n'a pas trouvé de solution dans 24% des cas est que la scène est complexe. Effectivement, comme on peut le constater sur la Figure 3.8, il n'y a pas beaucoup de marge de manœuvre afin de passer entre les deux blocs horizontaux pour atteindre le cylindre cible. Comme le contexte du test est plus complexe que le contexte d'utilisation réel du système, il semble raisonnable de croire que le système obtiendrait un plus grand nombre de succès dans une scène plus simple avec peu d'obstacles imposants, comme ce serait le cas dans un scénario d'alimentation.

Également, on remarque que le temps moyen pour le calcul de la trajectoire dans le cas des succès est de 0,797 seconde, dont, selon le test précédent, environ 0,132 seconde sont dédiées au calcul de la cinématique inverse. On remarque donc que la majorité du temps de calcul est utilisée par le planificateur de trajectoire OMPL. Or, comme le tout prend moins d'une seconde, le résultat est jugé acceptable pour la tâche d'alimentation à accomplir.

4.3 Résultats - Système complet

Le Tableau 4.3 présente les résultats pour ce test. Il est à noter que le temps total n'inclut pas seulement le temps de calcul et de déplacement du bras robotique, mais également le temps requis pour que l'utilisateur approuve manuellement chacune des étapes du processus en appuyant sur une touche. Effectivement, dans le cadre de cette expérience, avant chaque étape, une confirmation était requise par l'utilisateur, ce qui ne serait pas le cas pour la solution finale. Ainsi, il semble raisonnable de dire que le temps réel requis pour la tâche, dans le cas d'une automatisation complète, serait plus court de quelques secondes.

En observant les résultats ci-dessus, on remarque d'abord que le système est assez fiable, car tous les essais effectués se sont terminés par un succès. De plus, le temps d'exécution de la tâche est semblable pour chaque essai, avec un écart-type de 0,44. On remarque également que l'étape limitante du système, en termes de temps d'exécution, consiste au déplacement du bras robotique. Effectivement, on voit que la détection d'objet et la planification de trajectoire, ensemble, ne

représentent que 18,3% du temps total d'exécution, qui est de 90,22 secondes en moyenne. Ce temps de calcul est d'ailleurs du même ordre de grandeur que ce qui a été obtenu au test de la section 4.2. La différence peut être attribuable au fait que le système doit maintenant prendre une décision pour sélectionner l'objet le plus proche, ce qui n'était pas nécessaire au test précédent.

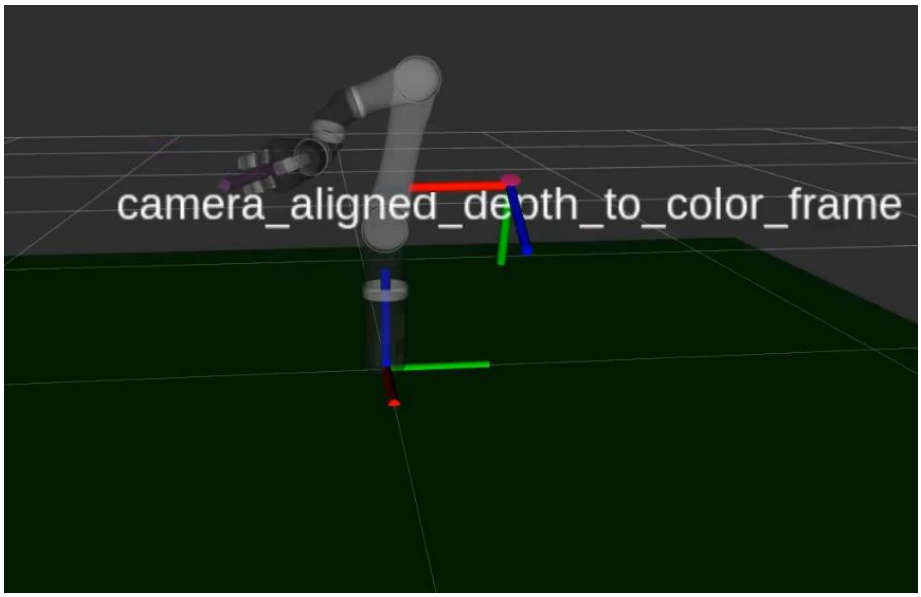
Tableau 4.3 Résultats de test – Système complet

Essai	**Succès**	**Temps de détection des carottes (secondes)**	**Temps de détection des brocolis (secondes)**	**Temps de calcul vers carotte (secondes)**	**Temps de calcul vers brocoli (secondes)**	**Temps total (secondes)**
1	Oui	7,11	6,89	0,99	0,97	89,71
2	Oui	8,10	7,09	0,98	0,98	91,00
3	Oui	7,61	6,66	0,98	0,96	89,98
4	Oui	7,72	6,91	1,01	0,99	90,38
5	Oui	7,34	6,98	0,96	1,02	90,04
MOYENNE		7,58	6,91	0,99	0.98	90,22
ÉCART-TYPE		0,34	0,14	0,02	0.02	0,44
POURCENTAGE MOYEN DU TEMPS TOTAL		16,1%		2,2%		100%

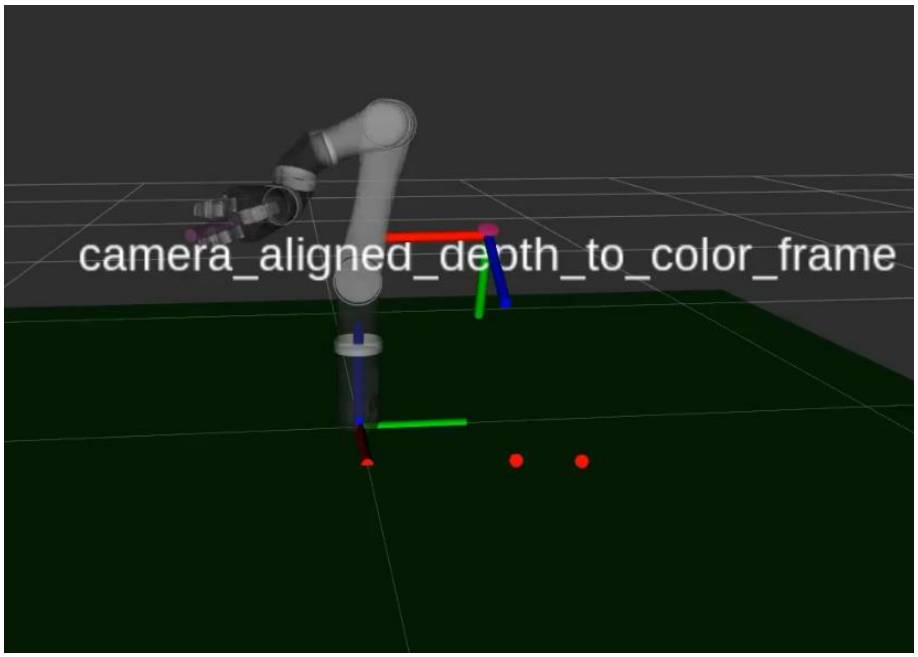
Ainsi, le temps moyen de calcul avant que l'utilisateur observe un déplacement du manipulateur est d'environ 8 secondes. Or, selon [85], 10 secondes est environ la limite pour que l'attention de l'utilisateur commence à diverger et qu'il s'impatiente. Ainsi, il semble que le temps de calcul du système, quoique près de cette borne, soit tout de même assez court pour que l'utilisateur reste concentré sur la tâche à accomplir. D'ailleurs, selon la même source, l'utilisateur devrait être en

mesure d’effectuer une tâche simple en environ une minute afin de ne pas s’impatienter. Or, le temps total de la tâche test est d’environ 90 secondes, ce qui correspond en réalité à deux tâches simples : l’utilisateur peut d’abord manger une carotte, puis un brocoli. Ainsi, le temps d’exécution de la tâche est bien en dessous de la limite. Le système semble donc assez rapide pour offrir une expérience positive à l’utilisateur.

Les images de la Figure 4.1 présentent le déroulement de la séquence test pour un essai. Les points rouges représentent le centre géométrique des carottes, alors que les points verts représentent le centre géométrique des brocolis. On remarque également que l’effecteur choisi, la fourchette, est représenté par un cylindre situé dans le préhenseur du bras robotique.

1		Position de départ. La fourchette est représentée par le cylindre saisi par l’effecteur.

2



Carottes détectées (points rouges).

Temps de détection : 7,58 s

Calcul de trajectoire

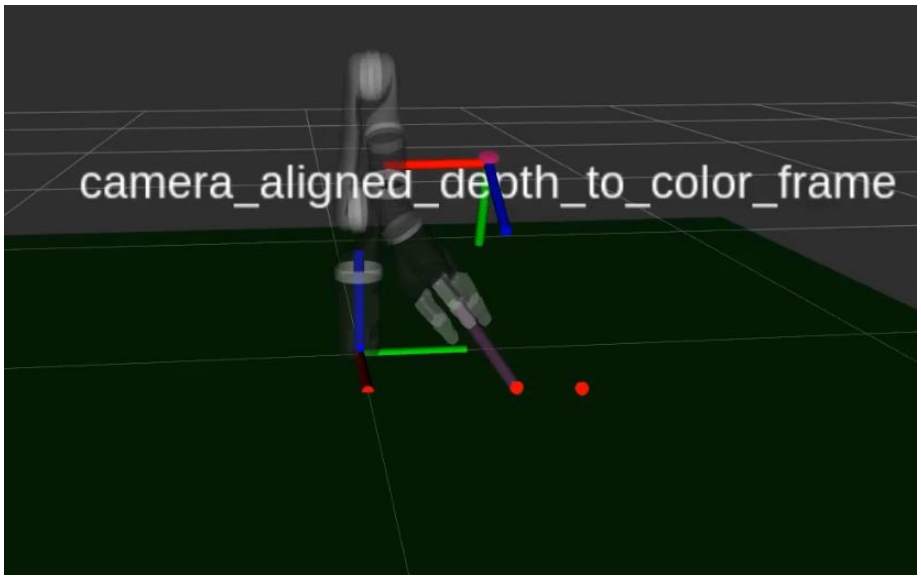
Temps de calcul : 0,99 s

3

Approche de la carotte la plus proche à l'angle spécifié.

Temps de déplacement: 13 s

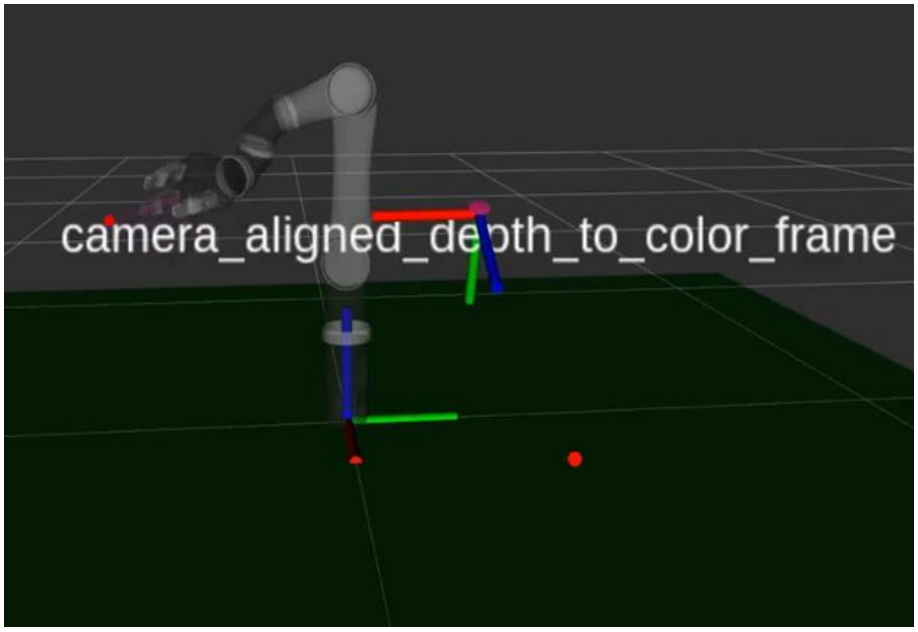
4



La fourchette pique la carotte.

Temps de déplacement: 3 s

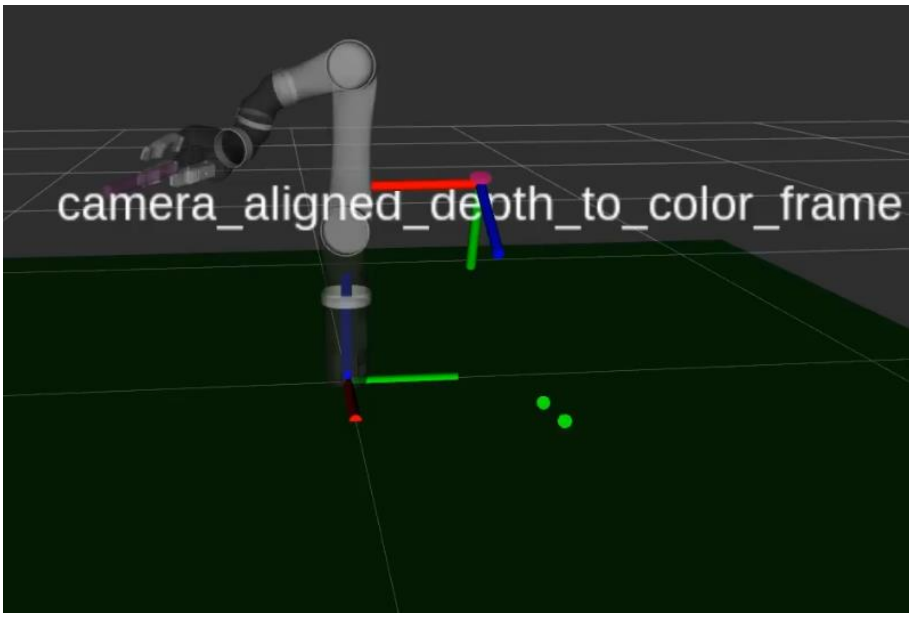
5



La carotte est ramenée à la bouche de l'utilisateur.

Temps de déplacement: 14 s

6



Brocolis détectés.

Temps de détection : 6,91 s

Calcul de trajectoire.

Temps de calcul: 0,98 s

7

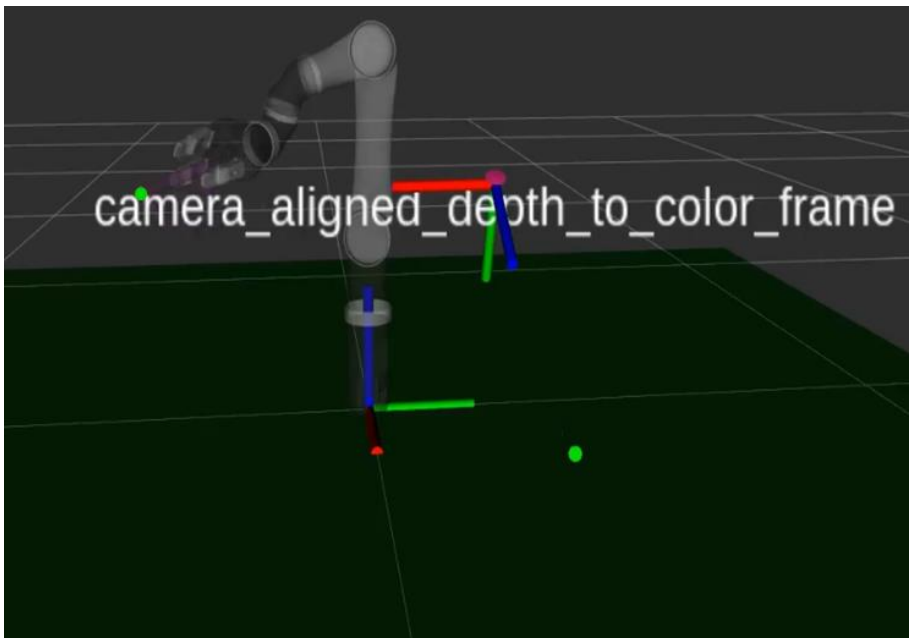
Approche du brocoli le plus proche à l'angle spécifié.

Temps de déplacement: 12 s

8

La fourchette pique le brocoli.

Temps de déplacement: 3 s



Le brocoli est ramené à la bouche de l'utilisateur.

Temps de déplacement: 13 s

Figure 4.1 Séquence d'exécution du test

4.4 Limitations et améliorations

Sous sa forme actuelle, le système présente plusieurs limitations.

L'angle d'approche de la nourriture est immuable; il est le même pour tous les types de nourriture. D'ailleurs, cet angle a été choisi de façon arbitraire et n'est donc peut-être pas optimal. Il serait possible d'effectuer une étude afin de tester différents angles d'approche pour différents types de

nourriture et, ainsi, d'avoir une approche adaptée à l'aliment désiré. Dans le même ordre d'idée, plutôt que de devoir spécifier l'orientation de l'effecteur souhaitée lors de l'approche des aliments, il pourrait être intéressant d'entraîner un agent afin de reconnaître automatiquement la méthode d'approche appropriée en fonction de la scène, de l'effecteur sélectionné, et de l'aliment à aller saisir. Un agent d'apprentissage par renforcement pourrait alors être particulièrement intéressant afin d'apprendre à accomplir la tâche.

Également, malgré le fait que le système actuel tient compte des obstacles présents lors de la prise de photo de la scène, il ne tient pas compte d'obstacles qui pourraient être en mouvement. Ainsi, si, par exemple, une personne déplace son bras entre l'effecteur et la nourriture après le calcul de trajectoire, il est possible qu'il y ait une collision. Dans la même veine, la trajectoire vers la nourriture est actuellement calculée entre la position initiale du bras et la position située à une fourchette de longueur de l'aliment. Si un obstacle se trouve entre cette position de pré-saisie et l'objet en tant que tel, le système n'en tiendra pas compte et on pourrait avoir une collision.

Toutefois, le plus gros questionnement est par rapport au fait qu'un système simulé a été utilisé plutôt qu'un système réel avec un bras robotique réel. En effet, même si une caméra et des images réelles ont été utilisées, et que, selon Kinova, le passage du système sous Gazebo vers un système réel est censé être très simple[86], il reste que le système d'assistance n'a pas pu être testé dans la vraie vie.

D'autres améliorations peuvent être apportées au système. La plus évidente consiste à modifier la méthode de sélection d'aliments. À l'heure actuelle, la sélection se fait à l'aide d'une modification écrite dans le code du système. Or, on pourrait aisément imaginer une sélection par reconnaissance vocale ou à travers une interface utilisateur. Cette amélioration permettrait de compléter l'expérience utilisateur et de rendre le système encore plus efficace et intuitif. Ce serait donc probablement la prochaine fonctionnalité du projet sur laquelle il faudrait travailler, afin de véritablement offrir un système efficace et intuitif pouvant être utilisé dans la vie de tous les jours par les personnes souffrant de troubles de mobilité du haut du corps.

Une autre amélioration « matérielle » consisterait à ajouter une deuxième caméra stéréo pointant vers le visage de l'utilisateur, ou un système de caméra monté à l'extrémité du bras robotique. Ainsi, il deviendrait possible de repérer la position de la bouche de l'utilisateur en temps réel et donc de s'assurer que la nourriture est bien présentée au bon endroit. Une caméra montée sur le

bras robotique permettrait aussi de faire varier le point de vue d'analyse de la scène et, ainsi, d'obtenir une meilleure idée des obstacles et des distances. Il serait aussi possible d'ajouter différents outils pour saisir la nourriture. Le système actuel utilise une fourchette; une cuillère pourrait être ajoutée, ce qui permettrait de varier le type de nourriture qu'il est possible de saisir. Par exemple, le système pourrait être ajusté afin d'aller chercher de la soupe. Il faudrait simplement s'assurer que l'orientation de la cuillère soit prise en considération et demeure constante, de manière à éviter les renversements.

4.5 Retour sur les objectifs et questions de recherche

Le Tableau 4.4 présente un rappel des sous-objectifs du projet de recherche.

Dans l'ensemble, les objectifs du projet de recherche ont été atteints. Effectivement, il a été possible de démontrer la faisabilité d'un système abordable et autonome d'assistance au repas utilisant un bras robotique de réadaptation. Dans tous les essais effectués, le bras MICO a été en mesure d'aller chercher des carottes et des brocolis en utilisant un outil (la fourchette) avec un temps d'exécution moyen de 90,22 secondes. Ainsi, le système développé est capable de reconnaître et de repérer la nourriture dans une assiette, d'amener l'effecteur du bras MICO à 6 degrés de liberté à cette position, puis de ramener la nourriture à la bouche de l'utilisateur.

La caméra Intel Realsense D415, en combinaison avec Faster RCNN, un algorithme de déprojection et la librairie *tf*, a permis d'atteindre le SO1 et d'accomplir les sous-tâches qui y étaient associées.

La librairie *MoveIt!*, combinée aux algorithmes OMPL, *Octomap* et TRAC-IK, a permis de résoudre le problème de calcul de trajectoire, d'évitement d'obstacle et de cinématique inverse, répondant ainsi au SO2.

Enfin, le système a pu être testé dans l'environnement de simulation Gazebo, en utilisant ROS pour assurer la communication entre les différentes parties du projet. Ce test répond au SO3.

La question de recherche principale est rappelée ci-dessous.

> *Peut-on automatiser entièrement le processus d'alimentation d'une personne à mobilité réduite, c'est-à-dire repérer la nourriture, la saisir et l'amener à la bouche de l'utilisateur ?*

Suite aux résultats de tests, il est possible de conclure qu'il est effectivement possible d'automatiser le processus d'alimentation. Le système développé constitue une preuve de concept fonctionnelle pour un système autonome d'assistance au repas.

Tableau 4.4 Résumé des objectifs de recherche

Objectif principal : Développer un système intelligent d'assistance au repas en asservissant un bras robotique de réadaptation.	
SO1 : Concevoir un système de vision capable de reconnaître et détecter la position de la nourriture dans une assiette.	
1.1. Prendre une image de la scène (couleur et profondeur).	✓
1.2. Utiliser un algorithme de reconnaissance d'objet pour reconnaître la nourriture et sa position dans le référentiel de la caméra.	✓
1.3. Convertir les coordonnées 3D du référentiel de la caméra vers le référentiel du bras robotique.	✓
SO2 : Effectuer un calcul de trajectoire pour un bras robotique de réadaptation à 6 degrés de liberté et résoudre sa cinématique inverse.	
2.1. S'assurer que l'algorithme sélectionné est en mesure de résoudre la cinématique inverse d'un bras robotique à 6 DDL avec poignet non-sphérique.	✓
2.2. Être en mesure d'imposer un angle d'approche de la nourriture.	✓
2.3. Ajouter un système d'évitement d'obstacles.	✓
SO3 : Tester le système en simulation avec une scène réelle.	
3.1. Créer un environnement de simulation communiquant avec les autres modules.	✓

CHAPITRE 5 CONCLUSION ET RECOMMANDATIONS

En résumé, le présent mémoire présente une preuve de concept d'un système autonome d'assistance au repas combinant la reconnaissance et la localisation d'objets dans l'espace au contrôle du bras robotique de réadaptation MICO de Kinova. Le résultat est un système qui pourrait faciliter la vie quotidienne des personnes souffrant de troubles de mobilité du haut du corps. L'algorithme de résolution de cinématique inverse TRAC-IK a été utilisé, offrant un taux de succès de 81,4% et un temps d'exécution moyen de 0,132 seconde. En combinaison avec l'algorithme de calcul de trajectoire et d'évitement d'obstacle OMPL, le système arrive à calculer les valeurs articulaires nécessaires pour atteindre une destination cible dans une scène complexe avec obstacles dans 76% des cas, le tout en aussi peu que 0,797 secondes en moyenne. Également, le système final proposé est fiable, ayant réussi avec succès les cinq essais effectués avec un temps moyen d'exécution de la tâche requise d'environ 90 secondes, dont 18,3% seulement est utilisé pour le traitement de l'image et la planification de trajectoire. Enfin, le temps d'exécution de la tâche est globalement assez rapide pour offrir une expérience utilisateur satisfaisante. Le temps de calcul pour repérer les aliments d'intérêt et calculer la trajectoire vers ceux-ci se rapproche de la limite de 10 secondes du temps d'attention d'un utilisateur, mais le temps d'exécution de la tâche en soit est bien en deçà du temps requis pour que l'utilisateur trouve que la tâche s'effectue lentement.

Toutefois, le système possède quelques limitations. Le choix d'orientation pour l'approche des aliments n'est pas autonome, mais imposé. Une étude de l'angle d'approche optimal en fonction du type d'aliment, ainsi qu'un agent d'apprentissage par renforcement, pourraient permettre de remédier à cette limitation. De plus, le système d'évitement d'obstacles n'est pas parfait puisqu'il ne tient pas compte des obstacles en mouvement, des obstacles qui apparaissent après la prise de photo de la scène, ou des obstacles entre l'aliment à saisir et la position de présaisie. Le système proposé n'a pas non plus été testé avec un bras réel; il est donc possible que le comportement du système dans la vraie vie varie un peu par rapport au comportement observé en simulation.

Outre remédier aux limitations ci-dessous, les travaux futurs par rapport au projet devraient d'abord se pencher sur la méthode de sélection des aliments, qui se fait actuellement de manière écrite dans le code. Le système d'assistance deviendrait alors encore plus intuitif et simple à utiliser. Dans le même ordre d'idée, en ajoutant une deuxième caméra pointant vers le visage de l'utilisateur, il

deviendrait possible de lui éviter d'avoir à déplacer son visage pour manger, puisque le système pourrait alors suivre sa bouche.

Malgré ces améliorations possibles, les résultats montrent tout de même que le système développé a le potentiel d'aider de manière significative les personnes souffrant de troubles de mobilité du haut du corps dans leur vie quotidienne et qu'il est donc possible de leur offrir un système autonome d'assistance au repas.

RÉFÉRENCES

[1] I. d. l. s. d. Québec, "L'Enquête québécoise sur la santé de la population, 2014-2015 : pour en savoir plus sur la santé des Québécois," Gouvernement du Québec, Institut de la statistique du Québec, Québec, 2016. [En ligne]. Disponible: https://www.stat.gouv.qc.ca/statistiques/sante/etat-sante/sante-globale/sante-quebecois-2014-2015.pdf

[2] D. M. Canada. (2020) Découvrir la dystrophie musculaire. [En ligne]. Disponible: https://muscle.ca/fr/

[3] S. Ishii et S. Tanaka, "Meal assistance robot for severely handicapped people. Robotics and Automation," *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, 1995.

[4] M. J. Topping, "An overview of the development of Handy 1, a rehabilitation robot to assist the severely disabled," *Journal of intelligent and robotic systems* 2002.

[5] M. J. Topping et J. K. Smith, "The development of handy 1. A robotic system to assist the severely disabled.," *Technology and Disability*, 1999.

[6] R. Soyama, "The development of meal-assistance robot 'My Spoon'," *Proceedings of the 8th International Conference on Rehabilitation Robotics*, 2003.

[7] R. Soyama et S. Ishii, "8 Selectable Operating Interfaces of the Meal-Assistance Device "My Spoon"," *Advances in Rehabilitation Robotics*, 2004.

[8] M. Partners. (2014) Mealtime Partners. [En ligne]. Disponible: http://www.mealtimepartners.com

[9] W. K. Song et J. Kim, "Novel Assistive Robot for Self-Feeding," dans *Robotic Systems - Applications, Control and Programming*, A. Dutta, Édit. Kanpur, India, 2012.

[10] T. Koshizaki et R. Masuda, "Control of a meal assistance robot capable of using chopsticks," *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, 2010.

[11] J. J. Villarreal et S. Ljungblad, "Experience centred design for a robotic eating aid," *2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2011.

[12] N. E. Robotic. (2019) Neater Eater Robotic: Neater Solutions. [En ligne]. Disponible: http://www.neater.co.uk/

[13] H. Tomimoto *et al.*, "Meal Assistance Robot with Blink Operation Interface," *International Conference on Industrial Application Engineering 2016*, 2016.

[14] Y. Ohshima *et al.*, "Meal support system with spoon using laser range finder and manipulator," *2013 IEEE Workshop on Robot Vision (WORV)*, 2013.

[15] K. M. Tsui et D.-J. Kim, ""I want that": Human-in-the-loop control of a wheelchair-mounted robotic arm," *Applied Bionics and Biomechanics*, 2011.

[16] Y.-S. L.-K. Cio, "Asservissement d'un bras robotique d'assistance à l'aide d'un système de stéréo vision artificielle et d'un suiveur de regard," Département de Génie Mécanique, École Polytechnique de Montréal, 2017.

[17] Y. S. Law-Kam-Cio *et al.*, "Proof of concept of an assistive robotic arm control using artificial stereovision and eye-tracking," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 2019.

[18] A. Aronson *et al.*, "Eye-Hand Behavior in Human-Robot Shared Manipulation," *2018 ACM/IEEE International Conference*, 2018.

[19] C. Perera *et al.*, "Electrooculography Signal based Control of a Meal Assistance Robot," *IASTED International Conference on Biomedical Engineering (BioMed 2016)*, 2016.

[20] C. Bousquet-Jette *et al.*, "Fast scene analysis using vision and artificial intelligence for object prehension by an assistive robot," *Engineering Applications of Artificial Intelligence* 2017.

[21] T. Bhattacharjee *et al.*, "Towards Robotic Feeding: Role of Haptics in Fork-based Food Manipulation," *IEEE Robotics and Automation Letters*, 2019.

[22] D. Gallenberger *et al.*, "Transfer depends on Acquisition: Analyzing Manipulation Strategies for Robotic Feeding," *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2019.

[23] K. Inc., "Kinova MICO Robotic Arm User Guide," 2018. [En ligne]. Disponible: https://www.kinovarobotics.com/sites/default/files/ULWS-RA-MIC-UG-INT-EN%20201804-1.0%20%28KINOVA%20MICO%E2%84%A2%20Robotic%20arm%20user%20guide%29_0.pdf

[24] T.-Y. Lin *et al.*, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2018.

[25] S. A. I. Laboratory. (2018) Robot Operating System. [En ligne]. Disponible: https://www.ros.org

[26] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," *ICRA Workshop on Open Source Software*, 2009.

[27] D. Coleman *et al.*, "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study," *ArXiV*, 2014.

[28] Microsoft. (2020) Kinect for Windows. [En ligne]. Disponible: https://developer.microsoft.com/en-us/windows/kinect/

[29] e.-c. Systems. (2020) Tara - USB 3.0 Stereo Vision Camera [En ligne]. Disponible: https://www.e-consystems.com/3D-USB-stereo-camera.asp

[30] O. S. R. Foundation, "taraxl-ros-package," 2019. [En ligne]. Disponible: http://wiki.ros.org/taraxl-ros-package

[31] Intel. (2019) Intel® RealSense™ Depth Camera D415. [En ligne]. Disponible: https://www.intelrealsense.com/depth-camera-d415/

[32] PC-Canada. (2020) Intel RealSense D415 Webcam - 30 fps - USB 3.0 - 1920 x 1080 Video. [En ligne]. Disponible: https://www.pc-canada.com/item/82635ASRCDVKHV.html

[33] P. Viola et M. Jones, " Rapid object detection using a boosted cascade of simple features " *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001.

[34] P. Viola et M. Jones, " Robust real-time face detection " *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, 2001.

[35] S. Umbaugh, *Computer Imaging: Digital Image Analysis and Processing*, 2005.

[36] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, 2004.

[37] N. Dalal et B. Triggs, " Histograms of oriented gradients for human detection " *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.

[38] S. V. Lab. (2016) ImageNet. [En ligne]. Disponible: http://www.image-net.org/

[39] A. Krizhevsky, I. Sutskever et G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems* 2012.

[40] R. Girshick *et al.*, " Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation " *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[41] R. Girshick, " Fast R-CNN " *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[42] S. Ren *et al.*, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2015.

[43] J. Redmon *et al.*, " You Only Look Once: Unified, Real-Time Object Detection " *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[44] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," *European Conference on Computer Vision 2016*, 2015.

[45] K. He *et al.*, " Deep Residual Learning for Image Recognition " *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[46] Z. Zou *et al.*, "Object Detection in 20 Years: A Survey", 2019. [En ligne]. Disponible: https://arxiv.org/abs/1905.05055

[47] L. Jiao *et al.*, " A Survey of Deep Learning-Based Object Detection " *IEEE Access*, 2019.

[48] H. Choset, "Robotic Motion Planning: Bug algorithms," éd, 2019.

[49] R. Siddiqui. (2018) Path Planning Using Potential Field Algorithm. [En ligne]. Disponible: https://medium.com/@rymshasiddiqui/path-planning-using-potential-field-algorithm-a30ad12bdb08

[50] M. Zohaib *et al.*, "Control Strategies for Mobile Robot With Obstacle Avoidance," *ArXiV*, 2013. [En ligne]. Disponible: https://arxiv.org/abs/1306.1144

[51] P. Abbeel, "Sampling-Based Motion Planning," éd. UC Berkeley, 2012.

[52] S. Pluzhnikov, "Motion Planning and Control of Robot Manipulators," Institutt for teknisk kybernetikk Trondheim, Norway, 2012.

[53] L. Yang *et al.*, "Survey of Robot 3D Path Planning Algorithms," *Journal of Control Science and Engineering*, 2016.

[54] H. Bharadwaj et V. Kumar, "Comparative study of neural networks in path planning for catering robots," *Procedia Computer Science*, vol. 133, p. 417-423, 2018.

[55] R. S. Hartenberg et J. Denavit, "A kinematic notation for lower pair mechanisms based on matrices," *ASME Journal of Applied Mechanics* 1955.

[56] R. S. Hartenberg et J. Denavit, *Kinematic synthesis of linkages*. New York, 1965.

[57] C. R. Roche, C. P. Tonetto et A. Dias, "A comparison between the Denavit–Hartenberg and the screw-based methods used in kinematic modeling of robot manipulators," *Robotics and Computer-Integrated Manufacturing*, 2011.

[58] D. L. Pieper, "The Kinematics of Manipulators Under Computer Control," Stanford University, 1968.

[59] R. Featherstone, "Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles," *The International Journal of Robotics Research*, vol. 2, nº. 2, p. 35-45, 1983.

[60] K. Elashry et R. Glynn, "An Approach to Automated Construction Using Adaptive Programing," dans *Robotic Fabrication in Architecture, Art and Design*, McGee, Édit., 2014, p. 51-66.

[61] S. Schaal, "Jacobian Methods for Inverse Kinematics and Planning," éd. USC, Max Planck, 2019.

[62] A. Gil, J. Segura et N. Temme, *Numerical Methods for Special Functions*: SIAM, 2007.

[63] J. F. Bonnans *et al.*, *Numerical Optimization -- Theoretical and Practical Aspects*. Berlin: Springer Verlag, 2006.

[64] MoveIt! (2018) TRAC-IK Kinematics Solver. [En ligne]. Disponible: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/trac_ik/trac_ik_tutorial.html

[65] C. Woodford, "Neural Networks," dans *ExplainThatStuff!* vol. 2020, éd, 2019.

[66] P. Jha, "A Neural Network Approach for Inverse Kinematic of a SCARA Manipulator," *International Journal of Robotics and Automation*, 2014.

[67] A. R. J. Almusawi, S. Kapucu et L. C. Dülger, "A New Artificial Neural Network Approach in Solving Inverse Kinematics of Robotic Arm (Denso VP6242)," *Computational Intelligence and Neuroscience*, 2016.

[68] Z. Xu *et al.*, "Dynamic Neural Networks Based Kinematic Control for Redundant Manipulators with Model Uncertainties," *Neurocomputing*, 2018.

[69] Z. Bingul, H. M. Ertunc et C. Oysu, "Comparison of Inverse Kinematics Solutions Using Neural Network for 6R Robot Manipulator with Offset," *2005 ICSC Congress on Computational Intelligence Methods and Applications*, 2005.

[70] A. K. Srivastava, "An Optimization Technique to Solve the Inverse Kinematics of Robot Manipulator," *International conference on Manufacturing Excellence*, 2013.

[71] B. Osinski et K. Budek. (2018) What is reinforcement learning? The complete guide [En ligne]. Disponible: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

[72] Connectability, "Organization and Sequencing through Task Analysis," éd, 2018.

[73] T. Foote, " tf: The transform library " *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013.

[74] N. Koenig et A. Howard, " Design and use paradigms for Gazebo, an open-source multi-robot simulator " *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[75] Google. (2016) Tensorflow detection model zoo. [En ligne]. Disponible: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

[76] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," *European Conference on Computer Vision*, 2014.

[77] J. Rey, "Faster R-CNN: Down the rabbit hole of modern object detection," éd: Tryolabs, 2018.

[78] I. A. Sucan et S. Chitta. MoveIt. [En ligne]. Disponible: moveit.ros.org

[79] P. Robotics. (2019) Planners Available in MoveIt. [En ligne]. Disponible: https://moveit.ros.org/documentation/planners/

[80] A. Hornung *et al.*, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.

[81] M. Lautman et R. Luthra. (2018) Perception Pipeline Tutorial. [En ligne]. Disponible: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/perception_pipeline/perception_pipeline_tutorial.html

[82] P. Beeson. (2018) Track_ik. [En ligne]. Disponible: https://bitbucket.org/traclabs/trac_ik/src/master/

[83] V. Ryaben'kii et S. Tsynkov, *A Theoretical Introduction to Numerical Analysis*: CRC Press, 2006.

[84] V. Dehdari, "Comparing three popular optimization algorithms by considering advantages and disadvantages of each of them using a field case study," *CCG Annual Report 13*, 2011.

[85] J. Nielsen. (2009) Powers of 10: Time Scales in User Experience. [En ligne]. Disponible: https://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/

[86] Kinova. (2018) MoveIt! For Kinova robots. [En ligne]. Disponible: https://github.com/Kinovarobotics/kinova-ros/wiki/MoveIt

[87] J. Cashbaugh et C. Kitts, "Automatic Calculation of a TransformationMatrix Between Two Frames," *IEEE Access*, 2018.

[88] R. Gourdeau, "ELE8203 - Robotique: Modélisation des robots manipulateurs," éd. École Polytechnique de Montréal, 2016.

[89] I. Realsense. (2019) Projection in Intel RealSense SDK 2.0. [En ligne]. Disponible: https://dev.intelrealsense.com/docs/projection-in-intel-realsense-sdk-20

ANNEXE A CHANGEMENT DE RÉFÉRENTIEL AVEC LA LIBRAIRIE *TF*

Soit O les coordonnées d'un point dans un système de coordonnées d'origine et D les coordonnées du même point dans un système de coordonnées de destination.

Afin d'effectuer le changement de référentiel, on cherche la matrice de transformation M pour laquelle

$$D = M * O$$

$$\begin{bmatrix} x_D \\ y_D \\ z_D \\ 1 \end{bmatrix} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_O \\ y_O \\ z_O \\ 1 \end{bmatrix}$$

La sous-matrice 3x3 $\begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix}$ représente la rotation entre les 2 systèmes de coordonnées, alors que la sous-matrice $\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$ représente la translation entre les systèmes. La quatrième ligne de M représente les facteurs d'échelle et de perspective. Dans ce cas-ci, on considère la perspective comme orthonormale et le facteur d'échelle comme étant 1:1 puisque la transformation est homogène [87].

Avec la libraire *tf* de ROS, la matrice M peut être trouvée en effectuant une série de transformations indépendantes à partir d'un repère initial.

Soit

$$T_{a,b,c} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{y,\beta} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_{z,\gamma} = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Où $T_{a,\,b,\,s}$ représente une translation de a, b, c le long des axes x, y, z respectivement

Et $R_{x,\alpha}$, $R_{y,\beta}$, $R_{z,\gamma}$ sont respectivement des translations de α, β, γ le long des axes x, y et z [88].

Supposons qu'on effectue les transformations dans l'ordre suivant

$$M = T_{a,b,c} * R_{x,\alpha} * R_{y,\beta} * R_{z,\gamma}$$

On peut alors développer l'équation selon les matrices définies précédemment afin d'obtenir

$$M = \begin{bmatrix} \cos(\beta)\cos(\gamma) & -\cos(\beta)\sin(\gamma) & \sin(\beta) & a \\ \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\sin(\gamma) & -\sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & -\sin(\alpha)\cos(\beta) & b \\ -\cos(\alpha)\sin(\beta)\sin(\gamma) + \sin(\alpha)\sin(\gamma) & \cos(\alpha)\sin(\beta)\sin(\gamma) + \sin(\alpha)\cos(\gamma) & \cos(\alpha)\cos(\beta) & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

On remarque alors que M a bien la forme attendue. Ainsi, en partant d'un repère initial, si on connait la série de rotations et de translation requis pour atteindre le repère désiré, il est possible de trouver M de manière analytique.

Ainsi, la librairie *tf* permet de calculer les matrices de transformation d'un repère à l'autre et de les garder en mémoire afin d'éviter d'avoir à les recalculer à chaque fois.

ANNEXE B DÉPROJECTION POUR LA CAMÉRA INTEL REALSENSE D415

Afin de passer des coordonnées 2D aux coordonnées 3D avec une caméra stéréo, il est nécessaire de connaître les paramètres intrinsèques de la caméra, qui sont uniques à chaque modèle. Pour la caméra D415, ces paramètres sont programmés dans l'appareil et accessibles avec la librairie *librealsense* fournie par Intel.

Également, par défaut, la caméra D415 n'utilise pas de distorsion de l'image, qui peut normalement être introduite à cause des lentilles utilisées. Ce paramètre simplifie donc le problème de projection, qui pourrait être plus complexe si un modèle de distorsion était utilisé [89].

On définit :

largeur et hauteur	Le nombre de lignes et de colonnes de la matrice représentant l'image
f_x et f_y	La longueur focale de l'image, qui est fonction de la largeur et la hauteur de l'image
pp_x et pp_y	Les coordonnées en pixels du centre de projection de l'image
X_{pixel} et Y_{pixel}	Les coordonnées en pixels du point 2D à convertir en 3D
D	La distance en mètres entre le point à convertir et le centre de projection de l'image

On peut alors effectuer une déprojection simple du point 2D vers son équivalent 3D

$$X_{3D} = D * \left(\frac{X_{pixel} - pp_x}{f_x}\right)$$

$$Y_{3D} = D * \left(\frac{Y_{pixel} - pp_y}{f_y}\right)$$

$$Z_{3D} = D$$