

**Conception et implémentation du contrôle pour le
manipulateur d'une plateforme robotique mobile**

par

Alexandre FRANCOEUR

Rapport technique élaboré dans le cadre du cours projets spéciaux GPA791
PRÉSENTÉ À Vincent DUCHAINE

MONTRÉAL, LE 7 AVRIL 2020

**ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC**

TABLE DES MATIÈRES

	Page
INTRODUCTION	1
CHAPITRE 1 DESCRIPTION DE LA PROBLÉMATIQUE	3
1.1 Mise en contexte	3
CHAPITRE 2 DEXTÉRITÉ : CONTRÔLE DU BRAS ROBOTIQUE OVIS	5
2.1 Définition de l'objectif	5
2.2 Choix technologiques	6
2.2.1 Matériel	6
2.2.2 Logiciel	6
2.3 Implémentation	7
2.3.1 Description du robot	7
2.3.2 Génération du paquet de support pour MoveIt	7
2.3.2.1 Génération du module d'extension IKFast	7
2.3.3 Simulation avec le simulateur industriel	8
2.3.4 Simulation avec Gazebo	9
2.3.5 Implémentation du contrôle linéaire	9
2.3.5.1 Contrôle du manipulateur à l'aide de la souris 3D	10
2.3.6 Essais sur le manipulateur	11
2.3.6.1 Configuration de la communication contrôleur-ordinateur	11
2.3.6.2 Interface matériel logiciel	12
2.3.6.3 Détection du contrôleur via le protocole USB	12
2.3.6.4 Essai du pilote informatique <i>kinova_ros</i> sur un ordinateur portable	12
2.3.6.5 Modification du pilote informatique <i>kinova_ros</i> pour l'architecture <i>aarch64</i>	13
2.3.7 Intégration de la pince à la solution	13
CHAPITRE 3 RÉSULTATS ET DISCUSSION	15
3.1 Description du bras robotique Ovis	15
3.2 Simulation	15
3.2.1 Simulateur industriel	15
3.2.2 Simulateur Gazebo	15
3.3 Algorithme	16
3.4 Interface matériel logiciel	18
CONCLUSION ET RECOMMANDATIONS	19
BIBLIOGRAPHIE	20

ANNEXE I ROBOTS ACTUELLEMENT SUR LE MARCHÉ 21

LISTE DES FIGURES

	Page
Figure 0.1 La plateforme robotique Markhor équipée du bras robotique Ovis.....	1
Figure 2.1 Méthode d'essais de la RoboCup Rescue.....	5
Figure 2.2 Ovis simulé par le simulateur industriel	8
Figure 2.3 Le système d'axes de la souris 3D	10
Figure 2.4 La pince Robotiq simulée avec Gazebo suite à une collision légère	14
Figure 3.1 Ovis et sa caméra simulée par Gazebo et visualisée dans RViz	16
Figure 3.2 Liaisons entre les différents noeuds logiciel de la solution	17

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

IMU	Inertial measurement unit
LiDAR	Light Detection and Ranging
NIST	National Institute of Standards and Technology ¹
ROS	Robot Operating System ²
SDF	Semantic Description Format ³
TORK	Tokyo Opensource Robotics Kyokai Association ⁴
UI	Interface utilisateur
URDF	Unified Robot Description Format ⁵
USAR	Urban Search and Rescue

1. <https://www.nist.gov>

2. <https://www.ros.org/>

3. <http://sdformat.org/>

4. <https://opensource-robotics.tokyo.jp/?lang=en>

5. <http://wiki.ros.org/urdf>

INTRODUCTION

Le club Capra conçoit et fabrique des robots terrestres depuis ses débuts en 1998. En 2017, le club était à la recherche d'un nouveau défi. Suite à plusieurs recherches et discussions, il fut décidé que le club participerait à la compétition RoboCup Rescue. La RoboCup Rescue a pour mission de stimuler la recherche et le développement de robots pour la recherche et sauvetage en milieu urbain⁶. Les épreuves de la compétition sont basées principalement sur les normes d'essais développées par l'organisme de standardisation américain «National Institute of Standards and Technology» (NIST)⁷.

Nous sommes présentement en cours de conception de notre deuxième prototype pour cette compétition. Il est conçu pour être plus robuste et plus performant que le prototype précédent. Celui-ci adopte une géométrie similaire à celle des robots actuellement disponibles sur le marché, soit quatre chenilles motrices pouvant être inclinées en fonction des besoins. À l'annexe I, se trouvent des images de certains robots qui ont servi d'inspiration.

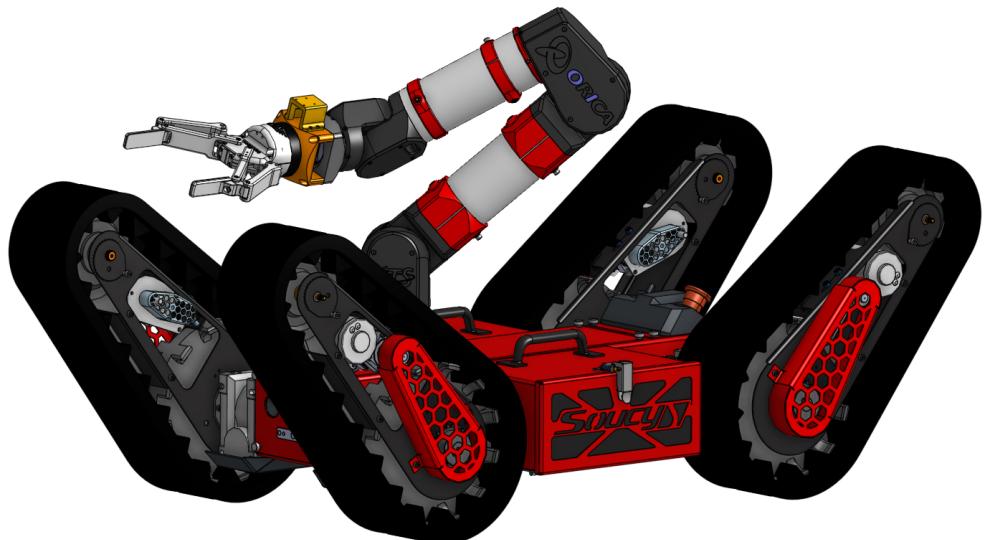


Figure 0.1 La plateforme robotique Markhor équipée du bras robotique Ovis

6. Traduit de l'anglais : «Urban Search and Rescue (USAR)»

7. Jacoff, A. (2014)

Le sujet traité par ce rapport est le contrôle du bras manipulateur qui est attaché à la plate-forme robotique. Pour ce faire, la problématique sera décrite puis analysée afin de définir l'objectif, l'objectif sera posé, les contraintes technologiques seront énoncés puis la mise en oeuvre de la solution sera décrite.

CHAPITRE 1

DESCRIPTION DE LA PROBLÉMATIQUE

1.1 Mise en contexte

Les services d'urgence sont appelés à travailler dans des conditions extrêmes. Dans le but de réduire leur exposition à certains types de danger tel que les radiations, sols instables, températures élevées, explosifs, ceux-ci utilisent des robots. L'utilisation de robots permet aux répondants de rester à une distance sécuritaire du danger. Toutefois, le temps de réponse et l'efficacité d'une intervention dépendent principalement des deux variables suivantes soit l'aisance de contrôle et la capacité pour l'opérateur à visualiser l'environnement qui entoure le robot.¹

Par conséquent, le contrôle du robot devra être conçu de manière à être intuitif. Ceci dans le but de permettre à un opérateur n'ayant aucune expérience préalable, de maîtriser le bras robotique très rapidement. Poursuivant l'idée explorée par la marine américaine pour le contrôle du périscope de leurs sous-marins², nous allons utiliser du matériel pré-existant et facile d'utilisation. Le but d'utiliser un composant en vente libre est qu'en raison des chaînes d'approvisionnement globales, il est possible de se procurer et de remplacer rapidement le matériel peu importe l'emplacement géographique où se déroule la mission.

Il n'y aura pas d'étude visant à qualifier la charge cognitive résultante de ce projet. Ceci parce que selon nos recherches, il n'existe pas à l'heure actuelle de méthode permettant de mesurer avec précision la charge mentale imposée à un opérateur. Certaines méthodes d'analyses existent tel que le NASA Task Load Index³ mais celle-ci n'est pas en mesure de réduire suffisamment le biais du répondant. Une autre méthode qui semble plus probante

1. Massey, K., Sapp, J. & Tsui, E. (2009)

2. Vergakis, B. (2017)

3. Hart, S. G. (1986)

est la mesure par oculométrie⁴ mais nous ne l'aborderons pas dans ce rapport étant donné que ce n'est pas le sujet de ce projet.

4. St-Onge, D. (2020)

CHAPITRE 2

DEXTÉRITÉ : CONTRÔLE DU BRAS ROBOTIQUE OVIS

2.1 Définition de l'objectif

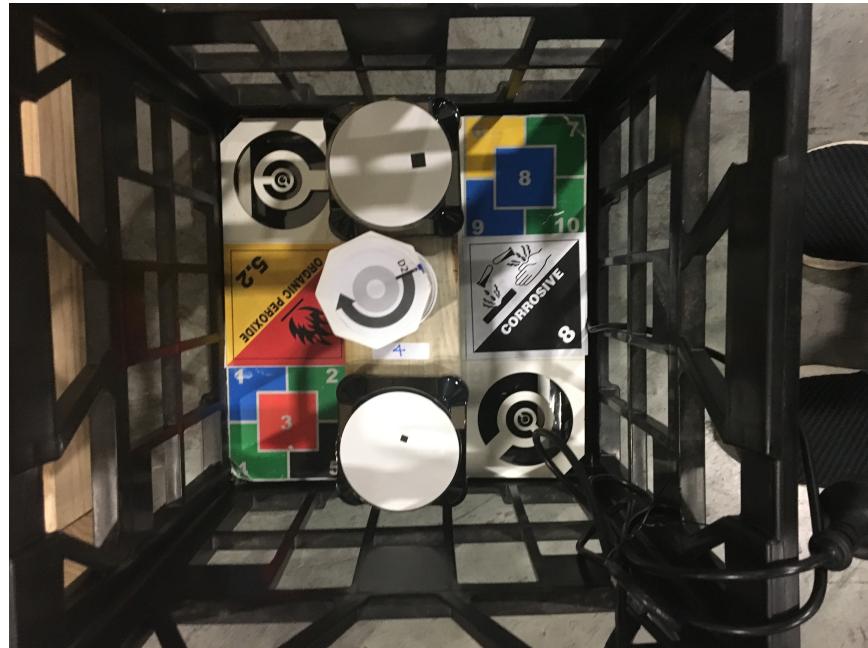


Figure 2.1 Méthode d'essais de la RoboCup Rescue

L'objectif est de contrôler le bras robotique à l'aide d'instructions de mouvement cartésien. Le bras devra tenir compte de l'environnement qui l'entoure afin d'éviter une collision. La méthode d'essais qui sera utilisé à la RoboCup Rescue consiste à faire pivoter un tube de plastique PVC de 180 degrés puis de le retirer dans l'axe d'un autre tube de plus gros diamètre, le tuyau doit ensuite être déposé dans la caisse de lait. La figure 2.1 montre le banc d'essais qui met à l'épreuve les robots sur leur capacité à interagir avec les éléments de leur environnement. En ajout à l'objectif principal, on définira l'objectif plus ambitieux de contrôler le bras robotique à l'aide d'une souris 3D. La pince devra aussi être contrôlable en position, la vitesse d'ouverture/fermeture ainsi que la force de préhension seront contrôlables par les paramètres de l'interface utilisateur (UI) pour le moment.

2.2 Choix technologiques

2.2.1 Matériel

Le matériel suivant est déjà en possession du club Capra :

1. Actuateurs et contrôleur de Kinova
2. Pince 2 doigts 140 mm de Robotiq
3. Bras robotique Ovis
4. Souris 3D - SpaceMouse de 3Dconnection
5. Ordinateur embarqué - Nvidia Jetson Xavier

2.2.2 Logiciel

Les solutions logicielles suivantes sont présentement utilisées par le club Capra :

1. Ubuntu 18.04 LTS
2. ROS Melodic Morenia
3. Interface utilisateur capra_web_ui qui fonctionne sur Windows

Les solutions logicielles suivantes seront explorées au cours de ce projet :

1. MoveIt¹
2. jog_control²
3. kinova_ros\kinova_driver
4. Simulateur robotique Gazebo
5. ROS industrial_robot_simulator

1. Sucan, I. A. & Chitta, S.

2. Tajima, R. (2018)

2.3 Implémentation

2.3.1 Description du robot

La première étape de la programmation du bras robotique consiste à formuler la description (URDF). Cette description contient les éléments suivants :

- L'origine et l'orientation des liens ;
- La nature et les limites des axes ;
- Les relations entre les liens et les axes ;
- Les descriptions géométriques pour l'affichage et le calcul des collisions.

Une fois cette description écrite et validée, il est possible de générer une description plus complexe (SDF) à l'aide de l'assistant de configuration MoveIt.

2.3.2 Génération du paquet de support pour MoveIt

L'assistant est simple à utiliser, il suffit de suivre les étapes énoncées dans le tutoriel, puis l'assistant génère un paquet de support utilisable par MoveIt.

Deux options s'offrent maintenant, simuler à l'aide du simulateur industriel ou simuler à l'aide du logiciel Gazebo. Ce dernier étant complexe à configurer et plutôt demandant en ressources, la première approche fut choisie afin d'effectuer des essais préliminaires.

2.3.2.1 Génération du module d'extension IKFast

La génération du module IKFast pour MoveIt requiert l'installation de la librairie Open-Rave. La procédure d'installation et de configuration étant ardue, l'utilisation d'une image docker pré-configurée simplifie grandement la tâche. Le tutoriel MoveIt sur la création du module IKFast³ est précis, il suffit de suivre les instructions. Le seul problème rencontré

3. ros-planning.github.io/moveit_tutorials/doc/ikfast/ikfast_tutorial.html

est lors de la configuration des permissions, le tutoriel dit de fermer la session et de se reconnecter, ce n'était pas suffisant, un redémarrage est nécessaire.

2.3.3 Simulation avec le simulateur industriel

La configuration requise afin de simuler le bras robotique à l'aide du simulateur industriel n'est pas évidente à réaliser. Principalement parce que la documentation n'a pas suivi les mises à jour. En s'inspirant de la configuration d'autres manipulateurs industriels, il fut possible d'arriver au résultat visé.

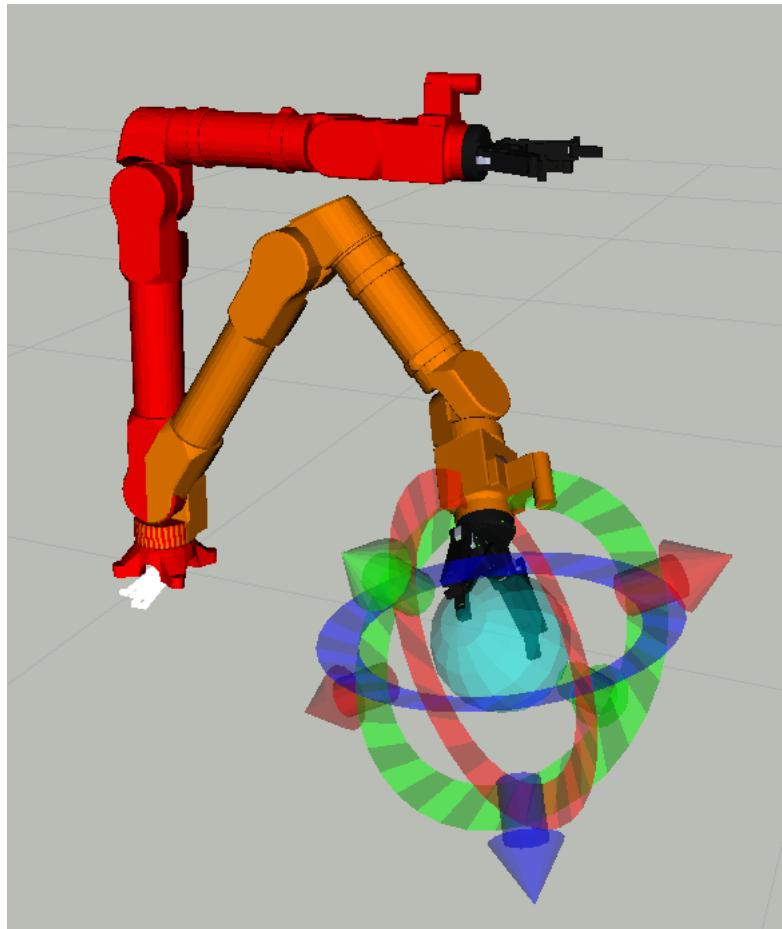


Figure 2.2 Ovis simulé par le simulateur industriel

Sur la figure 2.2, on voit la position actuelle simulée (rouge), la position à atteindre simulée (orange) ainsi que le marqueur interactif (sphère cyan). Toutefois, on remarque que les joints de la pince robotique ne sont pas contrôlés (blanc). Ceci est dû à la lacune suivante du simulateur industriel. Le simulateur industriel est conçu pour un seul manipulateur, il n'émule donc pas plus d'un contrôleur logiciel. Les doigts de la pince ne sont pas localisés dans l'espace et se retrouvent affichée par défaut à la base du manipulateur.

2.3.4 Simulation avec Gazebo

En raison des lacunes que présente le simulateur industriel, il fut décidé que pour la suite du projet, il serait préférable de configurer la simulation Gazebo. Pour configurer la simulation Gazebo, il est possible d'utiliser le *MoveIt Setup Assistant*, celui-ci génère un fichier URDF de base compatible avec Gazebo. Toutefois, ce fichier n'est pas modulaire et donc plus difficile à modifier et entretenir à l'avenir. Il fut donc décidé de continuer l'approche modulaire et d'étendre le contenu de la définition Xacro⁴ afin d'y inclure les paramètres spécifique à Gazebo tel que l'inertie, l'emplacement des capteurs de force et de la caméra du poignet. Il est aussi nécessaire d'ajouter le *plugin* pour simuler le contrôleur logiciel du robot.

2.3.5 Implémentation du contrôle linéaire

La beauté de l'écosystème ROS est bien entendu sa communauté vouée à la recherche et au développement de la robotique. Il est souvent aisément de trouver une entité qui à dû traiter un problème identique ou similaire et qui publie sa solution librement. En cherchant une solution pour le déplacement linéaire d'un robot manipulateur, on a trouvé que le Tokyo Opensource Robotics Kyokai (TORK) avait déjà travaillé sur cette problématique. Leur solution *jog_control* étant librement accessible, il a donc suffit d'adapter le fichier de lancement de cette suite logicielle afin de la faire interagir avec le manipulateur.

4. wiki.ros.org/xacro

2.3.5.1 Contrôle du manipulateur à l'aide de la souris 3D

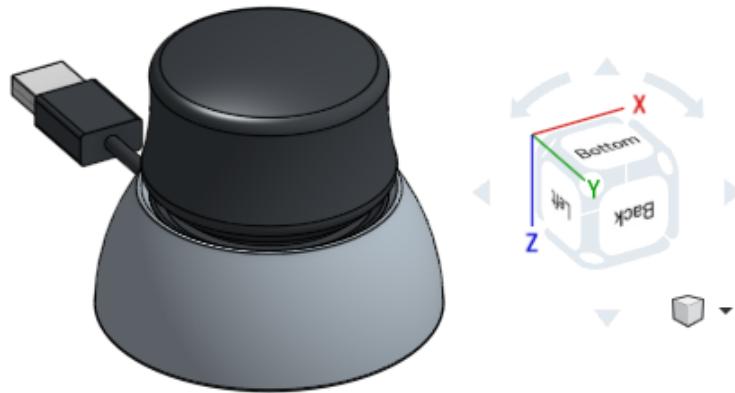


Figure 2.3 Le système d'axes de la souris 3D

Le logiciel *jog_control* contenait déjà un exemple permettant de contrôler un manipulateur à l'aide d'une manette de jeux vidéos. L'exemple fut testé à l'aide d'une manette Xbox afin de valider qu'il était fonctionnel. Par la suite, l'adaptation avec la souris 3D fut entreprise. L'exemple étant écrit en python, le prototype d'adaptation pour le contrôle avec la souris 3D l'est aussi. Ayant en tête le projet actuel, ce travail fut effectué à la session A2019⁵. Le prototype calcule d'abord une multiplication matricielle afin de passer des axes de la souris 3D tel que montrés à la figure 2.3 vers le référentiel local à l'effecteur.

La réécriture complète du logiciel fut débutée. Cette fois-ci, le logiciel est écrit en C++ afin d'augmenter la rapidité d'exécution. La réécriture comprend aussi un remodelage des différentes fonctions et variables membre de la classe. La réécriture ne fut pas sans heurt, la librairie ROS pour le C++ est plus complexe que celle pour python. Il fallut aussi trouver une librairie afin d'effectuer les calculs matriciels nécessaires pour le calcul des rotations, la librairie Armadillo⁶ fut sélectionnée en raison de sa documentation exhaustive et de sa facilité d'intégration avec ROS. Quelques recherches ont été nécessaire afin de configurer

5. github.com/tork-a/jog_control@b1b27c9

6. Sanderson, C. & Curtin, R. (2016)

avec succès l'éditeur *Visual Studio Code*⁷ afin d'être en mesure de déverminer le nouveau code.

Le fonctionnement général de l'algorithme est le suivant :

1. Les valeurs des axes de la souris 3D sont reçus ;
2. La valeur de l'axe le plus significatif est retenue ;
3. Les axes sont transformés de manière à refléter le système d'axes au manipulateur ;
4. Les valeurs transformées sont publiées afin d'être exécutées par le contrôleur du bras robotique.

2.3.6 Essais sur le manipulateur

2.3.6.1 Configuration de la communication contrôleur-ordinateur

Le contrôleur Kinova nécessite une configuration de départ. Étant donné la nature de notre projet par rapport à ceux de la clientèle typique de Kinova, il fut difficile de trouver l'information afin de configurer correctement le contrôleur. Les problèmes rencontrés étaient les suivant : la version du logiciel⁸ livrée avec les actuateurs n'était pas la plus récente malgré que le *firmware* des composants fut mis à jour ; la configuration réseau nécessite que le contrôleur ait une adresse MAC propre à lui, la procédure de configuration de l'adresse MAC n'était pas disponible en ligne, il fut nécessaire de contacter le support technique. Finalement, en suivant la procédure de configuration pour utiliser le contrôleur Kinova par le port réseau, le logiciel est maintenant capable de communiquer avec le contrôleur et les actuateurs. Toutefois, la connexion n'est possible que de pair à pair, il semble impossible d'utiliser un commutateur Ethernet *managed* entre les deux appareils.

7. github.com/RoboGnome/VS_Code_ROS

8. Kinova Development Center

2.3.6.2 Interface matériel logiciel

Avant de procéder à essayer le logiciel, on doit connecter le logiciel au matériel. Pour ce faire, les options sont d'utiliser le pilote informatique fourni par la solution *kinova_ros* ou d'utiliser la solution développée par le club Walking Machine aussi de l'ÉTS. Il fut choisi d'essayer la solution développée par Kinova étant donné qu'à première vue celle-ci semblait plus près de la solution recherchée.

2.3.6.3 Détection du contrôleur via le protocole USB

Lors de la première tentative de connexion entre l'ordinateur portatif et le contrôleur Kinova, le contrôleur n'était pas détecté. Suite à plusieurs recherches, le problème est causé par l'algorithme qui régule la consommation d'énergie et qui gère le bus USB. Cet algorithme permet en condition d'utilisation régulière d'économiser l'énergie en mettant en veille un périphérique qui n'est pas utilisé. Le contrôleur Kinova n'étant pas un produit grand public, celui-ci ne réagit pas de la manière attendu par l'algorithme. Le port USB est donc mis en veille, la communication avec le contrôleur est donc impossible. La solution la plus simple est de désactiver la mise en veille automatique pour tous les ports USB⁹. Une solution plus élégante est de modifier le fichier *.udev mais le temps étant limité cette approche sera explorée seulement si un problème de consommation élevée est détecté¹⁰.

2.3.6.4 Essai du pilote informatique *kinova_ros* sur un ordinateur portable

Afin d'essayer la solution fournie par Kinova, l'écriture d'un fichier de lancement ainsi que d'un fichier de configuration est nécessaire. Le fichier écrit pour le bras Ovis est calqué de celui existant pour lancer un bras Jaco de Kinova. Il est nécessaire d'ajuster quelques paramètres et le fichier lance le pilote informatique afin d'établir la communication avec le bras Ovis.

9. <https://askubuntu.com/a/1161074>

10. <https://askubuntu.com/a/525916>

2.3.6.5 Modification du pilote informatique *kinova_ros* pour l'architecture *aarch64*

Le pilote informatique fourni par Kinova est compatible seulement avec un ordinateur de bureau régulier possédant habituellement l'architecture *x32* ou *x86*. La plateforme Markhor est contrôlée par l'ordinateur embarqué, celui-ci repose sur l'architecture *aarch64*. Le pilote informatique fourni n'est pas compatible avec la solution matérielle utilisée cela pose problème. Suite à plusieurs recherches, un autre dépôt logiciel fourni par Kinova contient les fichiers compilés pour l'architecture visée. Il est donc nécessaire d'adapter le pilote informatique afin qu'il soit apte à localiser et référencer ces fichiers correctement lors de la compilation et l'exécution du logiciel. La solution à ce problème consiste à ajouter les fichiers précompilés (.so) dans un dossier au nom de l'architecture spécifique du Jetson soit : "aarch64-linux-gnu". Le fichier d'instructions au compilateur¹¹ étant écrit de manière paramétrique, aucune modification ne fut nécessaire.

2.3.7 Intégration de la pince à la solution

Grâce à l'initiative ROS-Industrial, la pince Robotiq utilisée vient avec un paquet logiciel ROS¹². Ceci réduit grandement la complexité d'intégration de la pince à la solution. Toutefois, ce paquet ne comporte pas de support pour la simulation dans Gazebo. La figure 2.4 montre les doigts de la pince Robotiq simulés qui se comportent de manière erratique suite à une collision légère, bien entendu ceci ne correspond pas au comportement réel de la pince.

Suite au développement de l'algorithme de contrôle du bras, il apparaît maintenant évident que le contrôle de la pince ne doit pas être intégré à celui-ci. Le contrôleur de la pince existant sera simplement lancé sur l'ordinateur embarqué de la plateforme mobile et le UI fera les appels nécessaires afin de contrôler manuellement l'ouverture/fermeture de la pince.

11. CMake.org

12. github.com/ros-industrial/robotiq

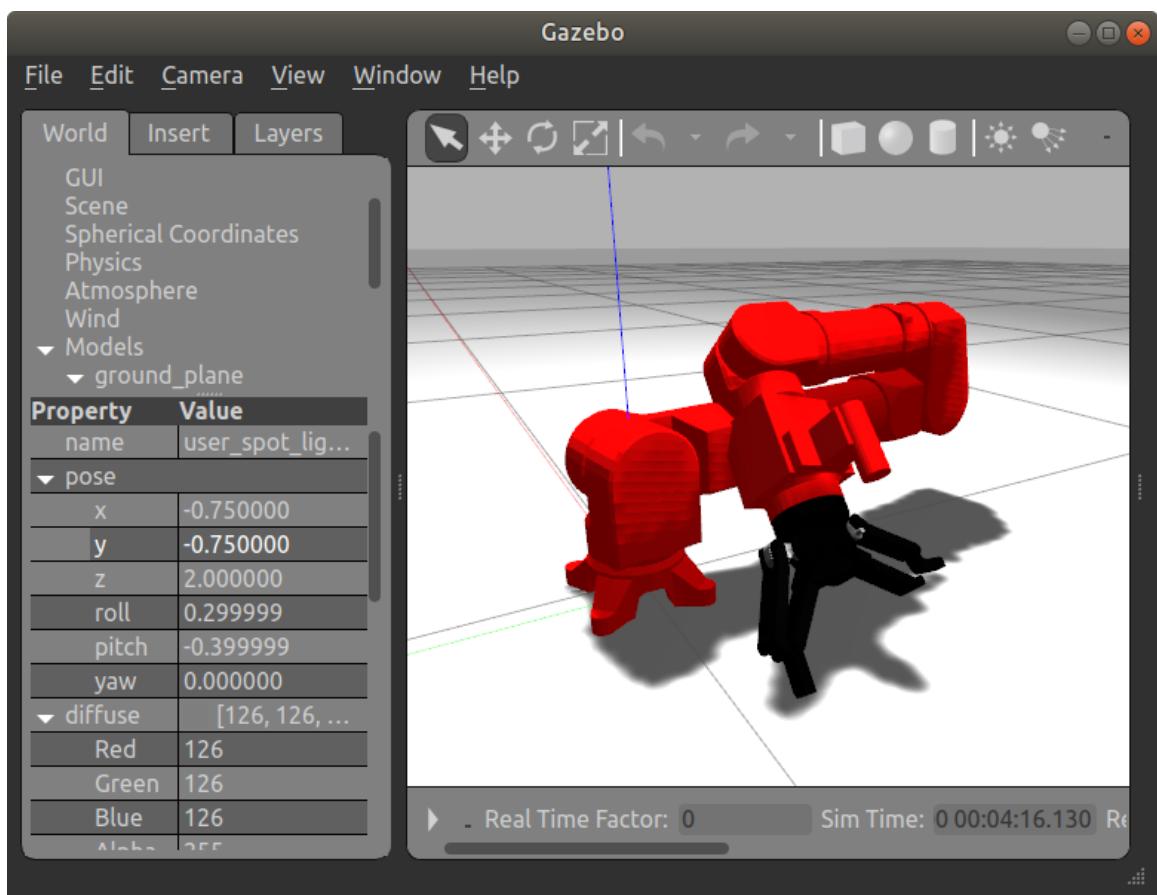


Figure 2.4 La pince Robotiq simulée avec Gazebo suite à une collision légère

CHAPITRE 3

RÉSULTATS ET DISCUSSION

Ce chapitre discute des résultats obtenus par ce projet.

3.1 Description du bras robotique Ovis

La définition du bras représente de manière fidèle le bras physique. Les membrures et joints sont positionnés adéquatement et leurs limites sont bien définies. La description est modulaire puisque la géométrie du bras est appelée à être modifiée au cours des prochaines années.

3.2 Simulation

3.2.1 Simulateur industriel

La simulation du manipulateur par le simulateur industriel est efficace pour effectuer le développement d'un algorithme. Toutefois, l'affichage des mouvements dans RViz est légèrement saccadé, ceci semble être un défaut provenant du simulateur industriel plutôt que de l'algorithme développé. Afin de valider cette hypothèse, un algorithme de contrôle pré-existant fut testé et la simulation répondait de manière similaire.

3.2.2 Simulateur Gazebo

La simulation du manipulateur avec Gazebo est maintenant complétée. Toutefois, la pince Robotiq apparaît mais n'est pas simulée adéquatement tel que mentionné à la section 2.3.7. L'ajout d'un contrôleur logiciel réglerait ce problème, mais puisqu'il n'y a pas de valeur ajoutée pour ce projet la pince restera simulée ainsi. La figure 3.1 montre la visualisation du bras dans RViz, lors de la simulation par Gazebo. On aperçoit l'image de la caméra au

poignet aussi générée par Gazebo. Une vidéo de la simulation est disponible en ligne à <https://youtu.be/ZcYvDAnuGzY>.

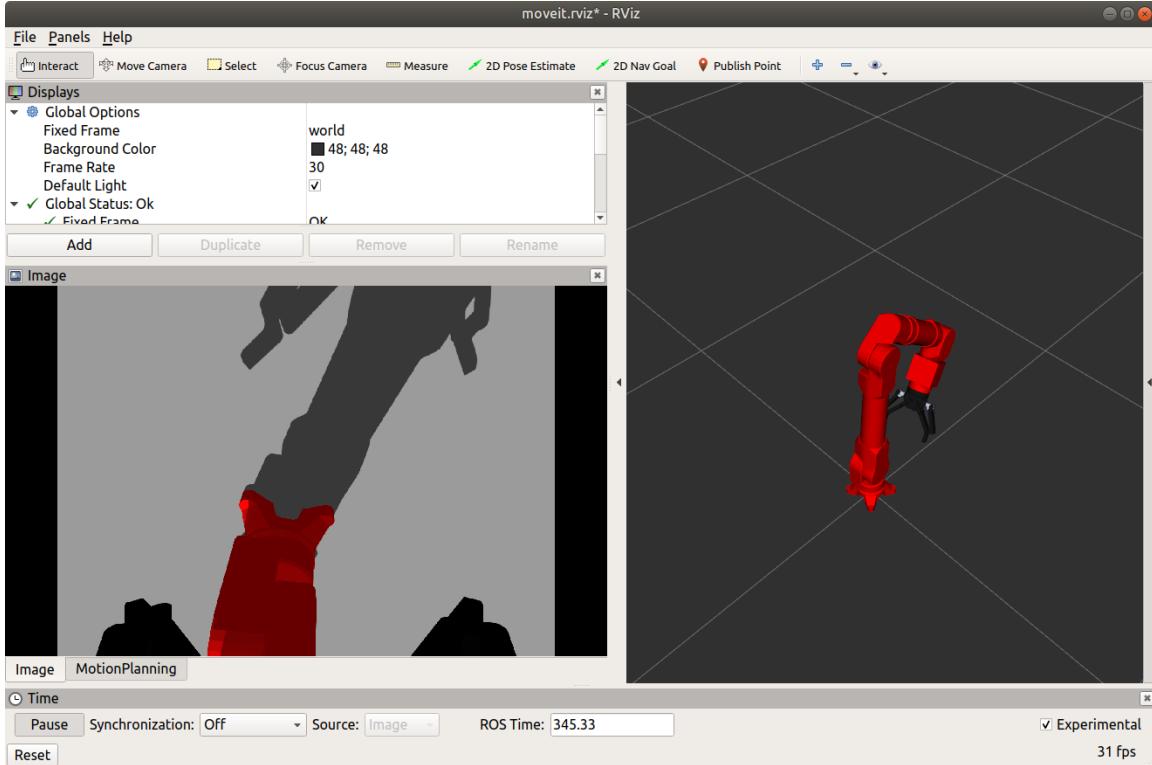


Figure 3.1 Ovis et sa caméra simulée par Gazebo et visualisée dans RViz

3.3 Algorithme

L'algorithme de contrôle en C++ est qualitativement plus rapide que le prototype précédent. Afin de valider que l'algorithme est, tel que souhaité, c'est à dire simple d'utilisation pour un opérateur n'ayant aucune expérience préalable, l'algorithme fut testé par quelques personnes externes au projet. Il n'y a eu aucun commentaires négatifs, l'expérience fut appréciée de tous. L'algorithme comporte toutefois certaines lacunes. Tout comme un robot industriel en mode *jog* qui effectue un mouvement linéaire, lorsque deux axes se trouvent à être parallèles le bras arrive à une singularité. L'algorithme ne gère pas ce cas particulier. Aussi, lorsque la pince du robot approche la limite de la zone de travail, le robot peut être plus difficile à manipuler puisqu'il perd généralement un degré de liberté. Une solution

envisagée pour palier ce problème est d'ajouter un contrôleur qui plutôt que de commander directement le système d'axes de référence, enverrait un vecteur afin de pousser/tirer virtuellement sur le système d'axes à l'outil. Finalement, malgré que l'algorithme permette au bras d'éviter les collisions avec lui-même et les objets connus de son environnement, celui-ci dépend de capteurs externes au bras. Une collision est possible entre le bras et un élément inconnu de l'environnement. Un événement de ce type n'est pas prévisible, mais l'algorithme pourrait être amélioré afin de tenir compte des capteurs de forces présent dans les actuateurs du bras et ainsi éviter d'endommager le manipulateur. En résumé, l'algorithme répond bien aux critères énoncés en début de projet et ce malgré qu'il ne soit fonctionnel que sur une plage réduite de l'enveloppe totale de travail accessible par le manipulateur.

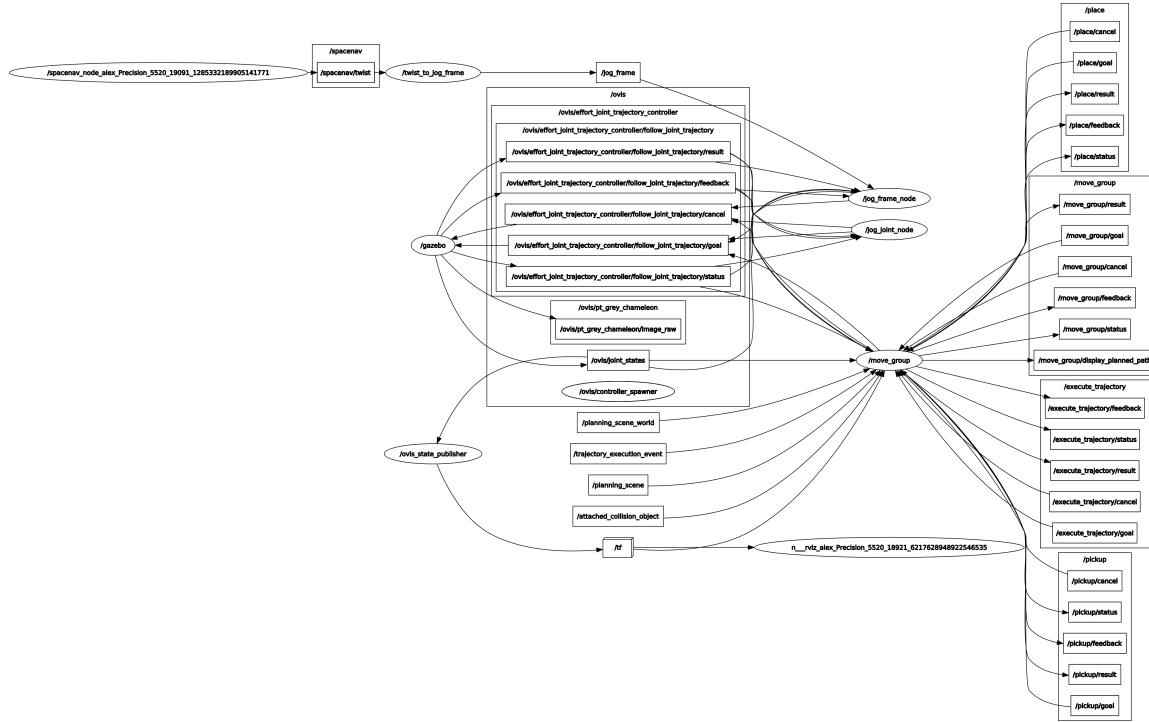


Figure 3.2 Liaisons entre les différents noeuds logiciel de la solution

3.4 Interface matériel logiciel

La communication entre le logiciel et le matériel n'est pas au point. L'ordinateur est apte à communiquer avec le contrôleur Kinova, ceci fut validé par l'envoi de commandes au travers du *UI* fourni par Kinova. Toutefois, la communication entre le logiciel MoveIt et le contrôleur logiciel fourni par Kinova n'est pas complétée. Le contrôleur logiciel Kinova est détecté par le noeud ROS mais n'est pas détecté par MoveIt. Les causes possibles sont les suivantes : la connexion USB est instable, le nom de *topic* attendu par MoveIt ne correspond pas à celui publié par le *driver* Kinova. Ceci est une tâche de configuration assez ardue puisqu'il n'est pas évident à première vue d'établir tous les liens nécessaires. À titre d'exemple, la figure 3.2 montre les liaisons entre les différents noeuds logiciel nécessaires au fonctionnement de la simulation visualisés à l'aide du logiciel *rqt_graph*¹. Malgré que ce point du projet est important pour la participation du club à la compétition. Il n'est pas critique à la validation de ce projet, la simulation étant suffisante à l'évaluation de l'algorithme. Il est important de mentionner que puisque l'algorithme est générique, celui-ci pourrait être testé sur tout autre robot supporté par ROS-Industrial et le logiciel *jog_control*.

1. wiki.ros.org/rqt_graph

CONCLUSION ET RECOMMANDATIONS

En guise de conclusion, ce projet montre qu'il est possible de contrôler un bras robotique à l'aide d'une souris 3D. Il aurait bien entendu été intéressant de réussir à établir la communication entre le logiciel et le matériel, mais la preuve de concept sur simulateur est suffisante en elle-même afin de déclarer ce projet un succès.

Ce projet permis d'appliquer les notions apprises dans les cours GPA546 (Robots industriels) et GPA434 (Ingénierie des systèmes orientés objet). Il est important de mentionner que la matière du défunt cours GPA435 (Systèmes d'exploitation et programmation de systèmes) aurait fort probablement aidé à régler certains problèmes rencontrés au cours de ce projet puisque l'entièreté du logiciel développé ici repose sur Linux.

Dans le cadre d'une mission de recherche et sauvetage, l'algorithme développé par ce projet permettra à l'opérateur de sauver de précieuses minutes sur les manipulations. Malgré qu'elle n'ait pas été mesurée, la réduction de la charge mentale est flagrante par rapport au contrôle joint par joint. L'algorithme habilitera aussi l'opérateur à effectuer des manipulations préalablement impossible. Ce domaine étant lié au militaire, l'exemple le plus fréquent de manipulation sensible est la manipulation d'une bombe tuyau. L'algorithme développé permettant de tirer dans un axe précis, ceci rend l'opération plus sécuritaire car il réduit le risque d'explosion soudaine et non contrôlée.

En terminant, il serait éventuellement intéressant de comparer l'efficacité de l'algorithme développé dans le cadre de ce projet avec l'algorithme jog_arm¹ qui fut ajouté à la documentation officielle de MoveIt récemment.

1. ros-planning.github.io/moveit_tutorials/doc/arm_jogging/arm_jogging_tutorial.html

BIBLIOGRAPHIE

- Hart, S. G. (1986). *NASA Task Load Index (TLX). Volume 1.0; Paper and Pencil Package.* Repéré à <https://ntrs.nasa.gov/search.jsp?R=20000021488>.
- Jacoff, A. (2014). *Guide for Evaluating, Purchasing, and Training with Response Robots Using DHS-NIST-ASTM International Standard Test Methods.* Gaithersburg, MD. Repéré à https://www.nist.gov/sites/default/files/documents/el/isd/ks/DHS_NIST_ASTM_Robot_Test_Methods-2.pdf.
- Massey, K., Sapp, J. & Tsui, E. (2009, 04). Improved situational awareness and mission performance for explosive ordnance disposal robots. *Unmanned Systems Technology XI*, 7332, 73320O. doi : 10.1117/12.820188.
- Sanderson, C. & Curtin, R. (2016). Armadillo : a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2), 26. doi : 10.21105/joss.00026.
- St-Onge, D. (2020). Des équipes de robots pour faciliter l'exploration spatiale. *Substance ÉTS.* Repéré à <https://substance.etsmtl.ca/equipes-robots-faciliter-l-exploration-spatiale>.
- Sucan, I. A. & Chitta, S. MoveIt motion planning software. Repéré à <http://moveit.ros.org/>.
- Tajima, R. (2018). "jog_control" : package of jog control for robot arms. Tokyo Open-source Robotics Kyokai Association. Repéré à https://opensource-robotics.tokyo.jp/?s=jog_control&lang=en.
- Vergakis, B. (2017). The Navy's most advanced submarines will soon be using Xbox controllers. *The Virginian-Pilot.* Repéré à <https://www.stripes.com/news/us/the-navy-s-most-advanced-submarines-will-soon-be-using-xbox-controllers-1.488074>.

ANNEXE I

ROBOTS ACTUELLEMENT SUR LE MARCHÉ



Figure-A I-1 Northrop Grumman Remotec Andros



Figure-A I-2 Telerob telemax PRO