



Stage de fin d'étude

**Développement sous ROS d'une tâche de contrôle d'un
robot yaskawa avec intégration dans une cellule
robotisée multi-constructeur**

Effectué du 22 Mars 2021 au 10 Septembre 2021

Andrea Melissa Pérez Peña

Deuxième année de Master en Électronique Électrotechnique et Automatique
Spécialité Robotique

Organisme d'accueil: AIP-Primeca Toulouse
Tuteurs Professionnels: Michel Taïx et Thierry Germa

Établissement d'enseignement: Université de Montpellier
Tuteur pédagogique: Andrea Cherubini

Année 2020-2021

Résumé

ROS, ou **R**obot **O**perating **S**ystem, est un middleware de type Open Source créé par l'ancien laboratoire de recherche Willow Garage. Conçu pour encourager le développement collaboratif de logiciels robotiques, il a fait sa première apparition en Mai 2010. Depuis lors, le nombre des robots déployés dans le commerce et l'industrie avec ROS continue d'augmenter.

L'objectif de mon stage au sein de l'**A**telier **I**nter-établissements de **P**roductique et **P**ôle de **R**essources **I**nformatiques pour la **MÉCANIQUE**, est de contrôler via ROS le robot collaboratif yaskawa HC10 afin de montrer l'intérêt de la générnicité qu'offre cet environnement pour la robotique industrielle.

Abstract

ROS, or **R**obot **O**perating **S**ystem, is an open source middleware created from the ground up by the former Willow Garage research laboratory. Designed to encourage collaborative development of robotic software, it made its first appearance in May 2010. Since then, the number of robots deployed in commerce and industry with ROS continues to increase.

The objective of my internship at the «**A**telier **I**nter-établissements de **P**roductique et **P**ôle de **R**essources **I**nformatiques pour la **MÉCANIQUE**», is to control through ROS the HC10 yaskawa collaborative robot in order to show the versatility that offers this environment in industrial robotics.

Remerciements

La réalisation de ce rapport a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.

Je voudrais dans un premier temps remercier mes tuteurs de stage, M. TAÏX, enseignant chercheur au LAAS-CNRS et M. GERMA chef de projet Robotique spatiale et Vision chez Magellium, tous deux intervenants auprès de l'AIP-PRIMECA pour leur partage des connaissances dans la robotique, leur disponibilité et surtout leurs judicieux conseils, qui ont contribué à alimenter ma réflexion et ma curiosité. Un grand merci à eux pour la confiance qu'ils m'ont accordée et l'opportunité qu'ils m'ont offerte.

M. BRIAND, directeur de la plateforme. Merci de m'avoir invité à participer à la « ROSCon Fr 2021 », cet événement m'a permis d'observer les différents projets qui se développent autour de ROS ainsi que d'exposer le travail réalisé lors de ce stage.

Mme. FUGIER, secrétaire de la plateforme. Merci pour votre accueil chaleureux, votre gentillesse et votre aide par lesquels ce stage s'est déroulé dans les meilleures conditions.

Mme. GRIMAL, ingénieure informatique. Merci pour les connaissances apportées au sujet des systèmes et des réseaux. En particulier, ceci m'a permis de comprendre les enjeux liés à la gestion des parcs informatiques.

M. MARCO, ingénieur contrôle système. Merci pour vos conseils et le temps consacré pour répondre à toutes mes questions. Votre collaboration et votre bienveillance ont contribué au bon déroulement de mon stage.

A mes très chers parents, Alexandra et William, je vous suis énormément reconnaissant de m'avoir toujours aidé et soutenu tout au long de ma formation.

Je remercie également mon frère Sergio et ma sœur Ariadna, pour leurs encouragements.

Sommaire

Remerciements	3
Table des figures	5
Liste des abréviations	6
Glossaire	6
Introduction	7
1. Présentation de l'organisme d'accueil	8
1.1 L'AIP-Primeca en France	8
1.2 Pôle Occitanie: L'AIP Primeca à Toulouse	9
1.3 RIO: Ros In Occitanie	9
1.3.1 Intégration de ROS dans la plateforme de production en 2020	10
1.3.2 Intégration du Yaskawa HC10	11
2. Robot Operating System	13
3. Yaskawa et ROS-Industrial	14
3.1 Architecture ROS-I pour les robots Yaskawa	15
3.2 Couche contrôleur	16
3.3 Couche « Simple Message »	16
3.4 Couche d'interface ROS-I	17
3.5 Couche de configuration	17
3.6 Couches supplémentaires	17
4. Etude de la problématique et travail effectué	18
4.1 Création d'un environnement de simulation	18
4.2 Génération de trajectoire	19
4.2.1 Génération de trajectoire avec le robot simulé	20
4.2.2 Génération de trajectoire avec le robot réel	21
4.3 L'utilisation des efforts sur le robot Yaskawa HC10	23
4.3.1 Calibration du robot	23
4.3.2 Récupérer le retour en effort	24
4.3.3 Récupérer le retour en position et vitesse de l'organe terminal	26
4.3.4 Mapping de la mémoire yaskawa	26
4.3.5 Création des messages personnalisés	26
5. Validation de la solution	28
5.1 Lecture et affichage des mesures	28
5.1 Simulation d'une tâche	30
Conclusion	34
Bibliographie	35
Annexes	36

Table des figures

1. Pôles du réseau SMART [i]	8
2. Logo formation RIO [ii]	10
3. Robot Staubli RX60 et Robot Kuka KR6 R700	11
4. Environnement de simulation de la plateforme de production [iii]	11
5. Robot Yaskawa HC10 [iv]	12
6. Architecture de communication ROS [v]	13
7. Logo Yaskawa [vi] et ROS-I [vii]	14
8. Architecture ROS-I pour les robots Yaskawa [viii]	15
9. Structure de message défini par le protocole « Simple Message »	17
10. Robot HC10 simulé sur Rviz	19
11. Rqt_graph de la simulation locale	20
12. Rqt_graph de la connexion avec le robot réel	21
13. Représentation de la position courante (en blanc) et la position désirée (en orange) du robot sur Moveit	22
14. Représentation de la position courante du robot après l'exécution de la trajectoire sur Moveit	22
15. Mesure de l'effort après calibration sur la position d'origine du robot affiché sur le boîtier de commande	23
16. Représentation du serveur et client pour récupérer le retour en effort	25
17. Structure du message pour le retour en effort	27
18. Structure du message pour le retour en position	27
19. Structure du message pour le retour en vitesse	27
20. Retour en position affiché sur le boîtier de commande	28
21. Retour en position affiché sur rqt	28
22. Retour en effort affiché sur le boîtier de commande	29
23. Retour en effort affiché sur rqt	29
24. Planification d'une collision potentiel	31
25. Points de passage	32
26. Exécution du chien de garde	33

Liste des abréviations

AIP-PRIMECA	Atelier Inter-établissements de Productique et Pôle de Ressources Informatiques pour la MÉCANIQUE
CAO	Conception Assistée par Ordinateur
CNRS	Centre National de la Recherche Scientifique
FAO	Fabrication Assistée par Ordinateur
ICA	Institut Clément Ader
INP	Institut national polytechnique
INSA	Institut National des sciences appliquées
IRIT	Institut de recherche en informatique de Toulouse
LAAS	Laboratoire d'analyse et d'architecture des systèmes
LIDAR	Light Detection and Ranging o Laser Imaging Detection and Ranging
OS	Operating System
RIO	Ros in Occitanie
ROS	Robot Operating System
ROS-I	Robot Operating System - Industrial
SMART	Systems Manufacturing Academics Resources Technologies
URDF	Unified Robot Description Format
XML	Extensible Markup Language

Glossaire

CoppeliaSim	C'est une plateforme de simulation de robot basée sur une architecture de contrôle distribué.
Fichier .launch	Spécifie les paramètres de configuration, les nœuds à lancer, ainsi que les machines sur lesquelles ils doivent être exécutés.
Gazebo	simulateur 3D, cinématique, dynamique et multi-robot
Lidar	Dispositif qui permet de déterminer la distance entre un émetteur laser et un objet ou une surface à l'aide d'un faisceau laser.
Open Source	Logiciel dont le code source est publié sous une licence open source ou fait partie du domaine public.
Rviz	Outil de visualisation 3D pour ROS

Introduction

Dans une des dernières publications faite par la Fondation Linux au sujet de la transformation de l'industrie à travers le logiciels Open Source, ils ont conclu que « la possession de la propriété intellectuelle des paquets logiciels dont les clients ne se servaient pas, limitait des opportunités commerciales et coûtaient cher en matière de développement et de maintenance. Pour accélérer l'adoption de ces logiciels, travailler ouvertement sur la plomberie commune pourrait présenter plus de possibilités de croissance des entreprises.»

En outre, «Tous se sont rendu compte que la collaboration ouverte offre des possibilités de réduction des coûts, de temps de mise en marché, d'amélioration de la qualité et d'ouverture de nouveaux secteurs de concurrence. La capacité d'atteindre ces résultats sur une base collective fait progresser l'innovation dans les industries respectives. » [1]

Ceci est vrai aussi pour le domaine de la robotique, en effet, dans le but de tirer parti de ces avantages, le projet ROS-Industrial a été conçu à partir d'une collaboration entre Yaskawa Motoman Robotics, le Southwest Research Institute et Willow Garage pour appuyer l'utilisation de ROS pour l'automatisation de processus de fabrication [2].

Dans le cadre du projet RIO (Ros in Occitanie) et avec le soutien du consortium européen ROS-Industrial, l'AIP-Primeca envisage de contrôler tout l'environnement de son atelier (Robots industriel, caméras, cellule de production) via ROS. L'intégration des robots Staubli RX60 et Kuka KR6 R700 a été réalisée en 2020 et la dernière acquisition de l'atelier, le robot Yaskawa HC10, n'est pas encore intégré ni dans son environnement natif ni sous ROS lors de mon arrivée en stage.

Ma mission principale consiste à intégrer le nouveau robot Yaskawa à l'environnement ROS et de développer une tâche de type inspection industrielle afin de valider le travail réalisé.

L'élaboration de ce rapport a pour finalité de présenter le contexte du stage au sein de l'AIP, ainsi que de présenter la méthodologie et les compétences utilisées pour mener à bien les missions qui m'étaient confiées.

1. Présentation de l'organisme d'accueil

1.1 L'AIP-Primeca en France

L'Atelier Inter-établissements de Productique et Pôle de Ressources Informatiques pour la MÉCANIQUE fait partie du réseau national français SMART (System Manufacturing Academics Resources Technologies), qui comprend 10 pôles régionaux en Aquitaine, Auvergne, Dauphiné-Savoie, Franche-Comté, Ile-de-France, Lorraine, Nord-Pas-de-Calais, Pays de la Loire, Rhônes Alpes Ouest et Toulouse. Ces pôles sont illustrés sur la figure 1.



Figure 1. Pôles du réseau SMART [i]

Ce réseau a été mis en place en 2015 pour répondre aux besoins des concepteurs de demain. Des formations et ressources pédagogiques sont proposées dans la finalité de développer et partager des connaissances scientifiques et technologiques autour de l'industrie à une échelle nationale et régionale. En particulier, le réseau s'intéresse à la transformation numérique afin de soutenir l'industrie du futur.

1.2 Pôle Occitanie: L'AIP Primeca à Toulouse

A une échelle régionale, le pôle occitanie propose des formations à des acteurs dans le milieu industriel et académique. Parmi eux, l'université fédérale de Toulouse, l'université Paul Sabatier, l'INSA, l'INP, le LAAS-CNRS, l'IRIT, l'ICA et le cluster Robotics Place.

L'atelier possède des matériels et des logiciels favorisant la mise en œuvre des manipulations à caractère industriel. Les utilisateurs (formateurs, enseignants, et industriels) sont confrontés aux enjeux de la vision industrielle, l'interaction homme-machine, les sciences de la production et la logistique, la conception assistée par ordinateur (CAO), la fabrication assistée par ordinateur (FAO) et la supervision des systèmes de production et usine.

En particulier, la plateforme robotique est composée d'un ensemble des robots de service, 5 robots TurtleBot et 2 robots humanoïdes NAOs de Softbank Robotics. Également, le service possède 2 drones Parrot BEBOP, et des robots de manipulation industriel, le Staubli RX60, le Kuka KR6 R700 et le Yaskawa HC10.

Les robots industriels se situent dans trois des quatre postes de travail disponibles d'une cellule flexible desservie par un système de transport Montrac (système de transport à monorail avec navettes automotrices). Le Staubli et le Kuka sont associés à un système de vision Cognex tandis que le Yaskawa est associé à un LIDAR (laser imaging detection and ranging en anglais) permettant de détecter l'accès à la zone de travail. L'ensemble du matériel mentionné donne origine à la plateforme de production de l'atelier, conçu pour aborder des problématiques robotiques particulières telle que l'informatique industrielle et le contrôle et intégration des systèmes de production robotisés, entre autres. [3]

1.3 RIO: Ros In Occitanie

L'académie ROS In Occitanie a été créée dans le pôle occitanie du réseau SMART grâce au financement du programme de recherche et d'innovation Horizon 2020 de l'Union européenne et grâce au soutien du consortium européen ROS-Industrial.

RIO propose un centre de formation ROS Industrial de 24 places dont l'objectif est de favoriser l'utilisation de l'environnement ROS dans les formations initiales et continues proposées par le service. Son logo est dessiné sur la figure 2.



Figure 2. Logo formation RIO [ii]

L'utilisation et le développement des logiciels open-source devient de plus en plus standardisé dans l'industrie et le middleware représente un outil avantageuse capable de résoudre des problèmes liés aux logiciels propriétaires (closed-source).

Cette notion de collaboration ouverte joue un rôle crucial dans la construction du savoir-faire des ingénieurs et développeurs robotiques de demain. Tout cela, autour d'un environnement commun qui facilite le prototypage rapide d'applications robotiques avancées.

1.3.1 Intégration de ROS dans la plateforme de production en 2020

Les modèles Kuka et Staubli de l'atelier (représentés sur la figure 3) n'étaient pas compatibles avec le middleware. Alors, un driver a été créé pour interpréter l'information transmise entre la couche ROS et l'OS du robot. Le serveur et le client communiquent à travers un protocole de communication par trames, où, chaque message est défini par un identifiant et ses données. Et chaque donnée est accompagnée par un marqueur de début et de fin.

Enfin, la cellule de production Montrac, commandée par des automates industriels Modicon M340 est simulée sur CoppeliaSim et Gazebo. Sur la figure 4, nous montrons la dernière version de l'environnement de simulation (2 postes de travail en fonctionnement).

Les paquets logiciels et les caractéristiques générales et techniques des robots industriels se trouvent sur <https://github.com/aip-primeca-occitanie>.

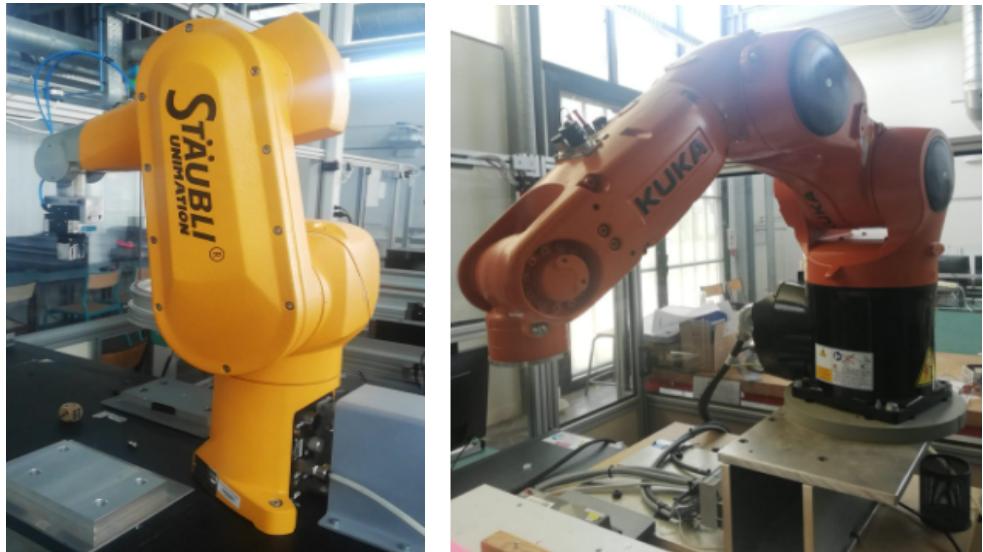


Figure 3. Robot Staubli RX60 et Robot Kuka KR6 R700

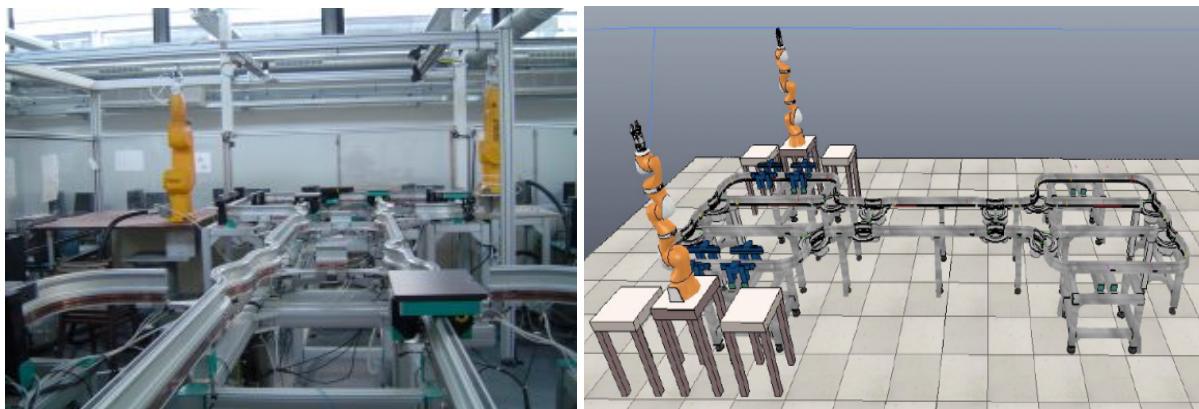


Figure 4. Environnement de simulation de la plateforme de production [iii]

1.3.2 Intégration du Yaskawa HC10

Arrivé à l'atelier en octobre 2020, le modèle HC10 (montré sur la figure 5), est un bras robot collaboratif 6 axes avec un poids de 47 Kg et possède une charge utile maximale de 10 Kg. Afin de garantir un fonctionnement sécurisé, des capteurs de couple sont intégrés sur chaque axe pour contrôler l'effort et limiter la puissance en cas de contact avec un opérateur.

En plus, en fonction de l'application et en combinaison avec des dispositifs de sécurité externe (tapis sensibles ou un lidar dans notre cas) le cobot peut être utilisé sans une barrière de sécurité. Ceci, favorise son installation dans différents lieux de travail, l'utilisation sécurisée du mode collaboratif par la détection de l'opérateur dans l'espace de travail et offre une rentabilité maximale en termes d'espace et des coûts. (Annexe 1)



Figure 5. Robot Yaskawa HC10 [iv]

Pour sa mise en oeuvre, le robot est accompagné de:

- Un boîtier de commande, qui sert à la programmation des tâches complexes.
- Un contrôleur de haute performance YRC1000, qui rend possible des vitesses accrues et une plus grande précision de trajectoire.
- Un outil en bout de bras, qui varie en fonction de l'application, par exemple, une pince, ou un préhenseur.
- Le logiciel Motosim, qui est un simulateur de programmation de robots qui utilise le langage natif INFORM.

Désormais, pour l'intégration sous ROS-I, le robot possède un driver spécifique aux robots Motoman (Gamme des robots Yaskawa), appelé **MotoROS**. Il garantit que tous les paramètres internes du robots soient correctement configurés et établit la communication entre ROS et les contrôleurs de robots Motoman.

Le driver est open source et les paquets associés se trouvent sur <https://github.com/ros-industrial/motoman>.

Dans le but de familiariser le lecteur avec le fonctionnement de ROS-I, nous montrerons plus tard un aperçu de son architecture et la façon dont elle interface avec les différents paquets logiciel (§3).

2. Robot Operating System

La conception d'un robot est complexe, non seulement il faut prendre en compte l'assemblage de pièces mécaniques, électromécaniques ou pièces électroniques. Mais aussi, il faut considérer le logiciel embarqué associé, qui varie en fonction du degré de complexité des tâches à accomplir.

Généralement, les problématiques associées au développement logiciel sont liées à l'implémentation des protocoles de communication entre les éléments qui composent le robot, tels que les capteurs, les actionneurs et l'unité centrale. Par ailleurs, une autre problématique est l'élaboration des modules qui traitent l'information recueillie par les capteurs pour la création des programmes de commande et contrôle.

Dans l'objectif de simplifier ces tâches et optimiser la conception des systèmes robotiques, ROS (ou Robot Operating System) est introduit en 2007, afin de fournir une plateforme standard pour le développement logiciel des robots.

En plus de ce dernier, ROS est considéré comme un méta-système d'exploitation, car il fournit des services proches d'un système d'exploitation (abstraction du matériel, contrôle des périphériques de bas niveau) et aussi, agît comme un middleware, qui se charge de la transmission de messages entre les processus, les paquets logiciels, et des systèmes robotiques différents.

Sur la figure 6, nous montrons l'architecture de communication ROS.

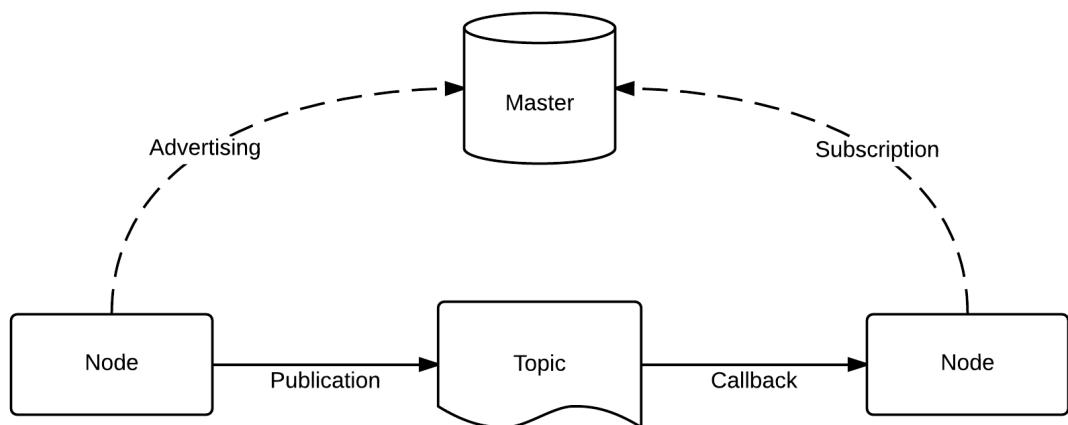


Figure 6. Architecture de communication ROS

Nous expliquons cette architecture dans l'intérêt d'illustrer postérieurement l'échange entre les blocs logiciels utilisés pour la génération de trajectoire (§4.2).

Un nœud (ou Node en anglais) est un processus qui effectue une opération. Il peut échanger avec d'autres nœuds à travers des topics (communication asynchrone) ou des services (communication synchrone).

Lorsqu'un nœud est initialisé, il s'identifie auprès du « master » qui agit comme un intermédiaire entre 2 nœuds souhaitant communiquer.

Pour réaliser cet échange, un des noeuds avertit au driver qu'il veut envoyer de l'information, il sera donc un « publisher ». Et le deuxième avertit qu'il veut recevoir de l'information, il sera donc un « subscriber ». Après, le master enregistrera ces déclarations et établira la connexion. En plus, un nœud peut s'abonner et publier de l'information à d'autres nœuds en même temps.

La communication sous ROS se base sur ce modèle de publisher/subscriber. Cela permet d'avoir un échange inter-processus et inter-machine à condition d'avoir connaissance du même master.

3. Yaskawa et ROS-Industrial



Figure 7. Logo Yaskawa [v] et ROS-I [vi]

« Constatant que les intégrateurs de systèmes et les utilisateurs finaux ont besoin d'apporter des modifications fonctionnelles adaptatives aux applications industrielles traditionnelles de robots dans des processus répétables à grand volume, Yaskawa a été l'un des premiers fabricants de robots à soutenir l'initiative ROS aux États-Unis, rejoindre le ROS-Industrial Consortium (RIC) Amériques pour accélérer le progrès technologique et développer un moteur de communication robuste» [5].

MotoROS ouvre la possibilité aux entreprises, universités et laboratoires de recherche de mettre au point des nouvelles technologies basées sur ROS, qui pourront se transformer éventuellement en usage industriel et commercial.

Les robots Motoman et les générations de contrôleurs de robots plus récents prennent en charge ROS sur différentes couches. Pour nommer quelques-uns, le contrôleur FS100, utilisé pour l'emballage et applications de manipulation. Le contrôleur DX200, utilisé pour la planification de trajectoire et le mouvement coordonné de plusieurs robots. Et le contrôleur YRC1000, utilisé pour des applications de précision.

Nous expliquons ci-dessous la manière dont les contrôleurs interagissent avec l'architecture en couches de ROS-I, le diagramme associé se trouve sur la figure 8.

3.1 Architecture ROS-I pour les robots Yaskawa

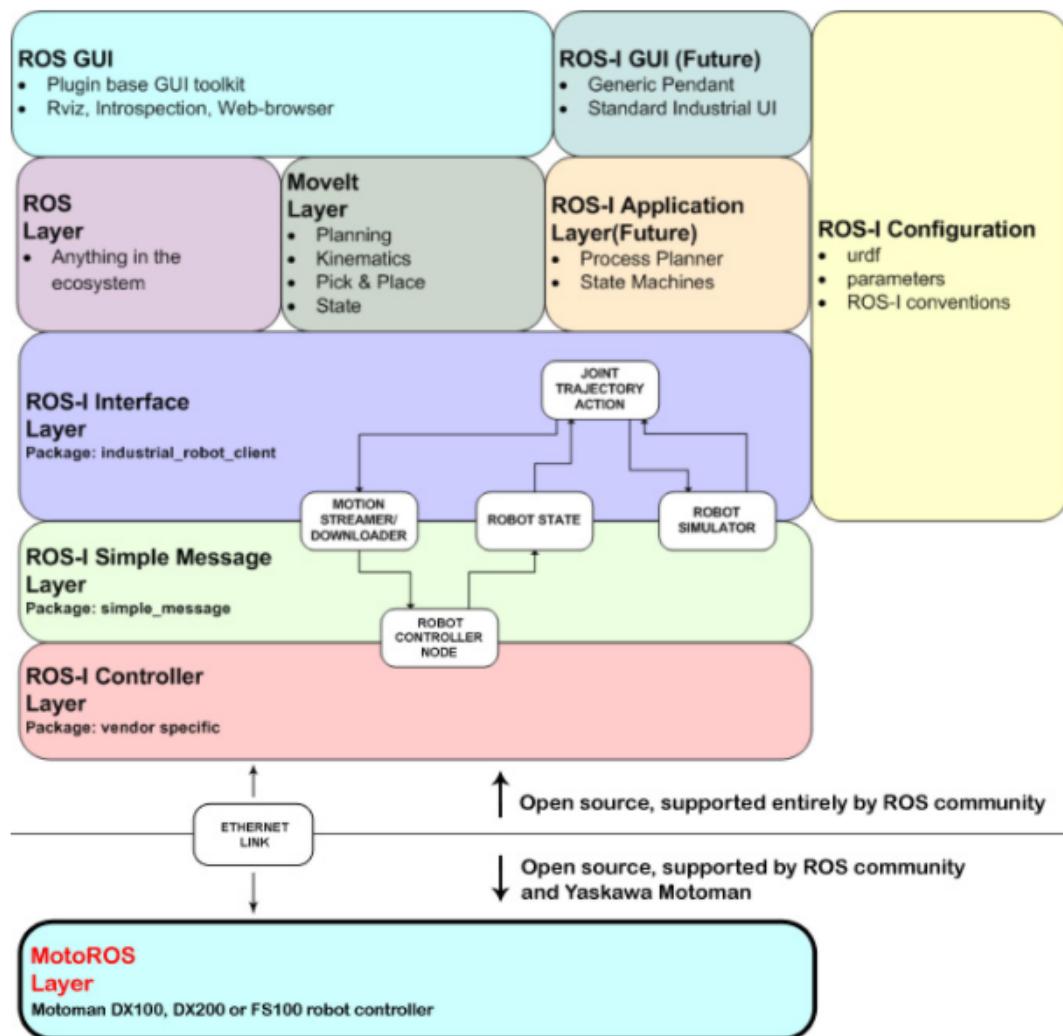


Figure 8. Architecture ROS-I pour les robots Yaskawa [vii]

Tout d'abord, nous soulignons que la « couche MotoROS » et la « couche ROS contrôleur » est la même. Nous allons du bas vers le haut pour introduire cette architecture.

Alors, nous partons d'une couche interface avec le robot via le driver MotoROS. Ensuite, nous passons par la « couche Simple Message » qui enveloppe l'information envoyée depuis l'OS du robot vers ROS et vice versa, selon le protocole de communication, appelé aussi *Simple Message*.

Après, la « couche d'interface » implémente un nœud (`joint_trajectory_action`) qui fournit une interface pour suivre l'exécution de la trajectoire. Pour finir, nous arrivons aux couches d'en haut, correspondant à des modules de contrôle, planification, traitement d'images, etc. Ces couches seront rajoutées une après les autres en fonction du besoin et elles auront accès aux informations intrinsèques du robot (contenues dans la « couche de configuration ») au profit du développement des tâches industrielles.

3.2 Couche contrôleur

Dans cette couche se trouve le driver MotoROS. Il exécute automatiquement une tâche dans le langage natif du contrôleur qui sera au début des toutes les commandes de mouvement.

Le driver communique avec le middleware et transmet des informations au contrôleur en utilisant la structure de message défini par le protocole « Simple Message ».

3.3 Couche « Simple Message »

A ce niveau, un protocole de communication et une interface capable de transmettre la structure de ces messages sont définis. Nous décrivons cette structure car nous nous servirons plus tard pour récupérer des valeurs spécifiques sur le contrôleur (§4.3.2).

Les messages commenceront toujours par un préfixe, suivi d'un en-tête et un corps [7]. Le préfixe, correspond à la somme en octets de l'en-tête et du corps. Il agit comme une somme de contrôle (Checksum en anglais) qui vérifie l'intégrité du message envoyé.

Ensuite, l'en-tête est divisé en trois champs qui spécifient le type de message envoyé. Le premier, appelé `msg_type`, contient l'identifiant du message. C'est-à-dire qu'il définit le type d'information à récupérer. Le deuxième, appelé `comm_type`, indique le type de flux des données. Et, le troisième, appelé `reply_code`, avertit si l'information a été reçue. (Annexe 2)

Pour finir, le corps correspond aux données, déterminées par le type de message et le type de communication. La taille des données peut varier et dans certains cas un message ne peut contenir aucune donnée.

Nous dessinons la structure du message sur la figure 9.

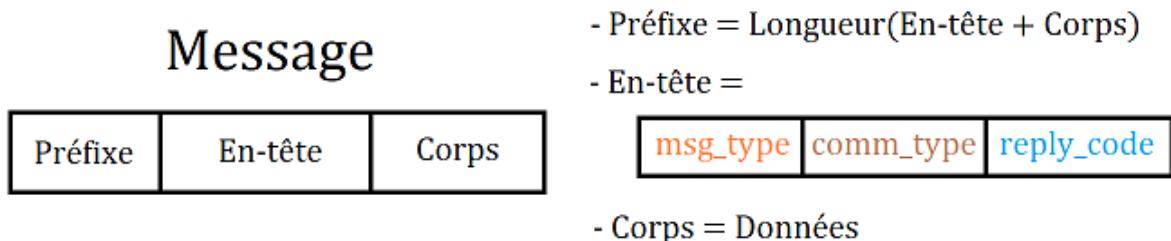


Figure 9. Structure de message défini par le protocole « Simple Message »

3.4 Couche d'interface ROS-I

Dans cette couche s'introduit une interface normalisée pour le contrôle des robots industriels, basée sur la spécification ROS-I. Par conséquent, l'architecture pourra communiquer avec n'importe quel contrôleur possédant un serveur compatible au protocole « Simple Message ».

3.5 Couche de configuration

Grâce à cette couche, nous récupérons la configuration du robot. Généralement, cette information s'enregistre dans les paramètres ROS, aussi appelés « rosparams » lors de l'exécution du fichier .launch (écrit en langage XML) et reste à la disposition de programmes dans l'environnement ROS tant que le « roscore » est en marche.

Le « roscore » lance le « Master » (§2) , ainsi que le serveur de paramètres ROS et un nœud de journalisation.

3.6 Couches supplémentaires

D'autres couches peuvent être ajoutées afin d'étendre les fonctionnalités du robot en fonction de l'application. Le logiciel Gazebo offre une simulation physique très proche de la vie réelle. Le logiciel Moveit offre plusieurs planificateurs qui s'adaptent aisément à une zone de travail contrainte et il existe encore d'autres logiciels qui s'adaptent à la manipulation, détection et le traitement d'images.

4. Etude de la problématique et travail effectué

Comme mentionné au §1.3.2, le robot dispose d'un driver qui respecte la norme ROS-I. Nous choisissons de l'utiliser, contrairement à celui conçu pour le Kuka et le Staubli (§1.3.1), car nous voulons utiliser au maximum l'architecture ROS disponible.

Alors, nous ne nous soucions pas dans un premier temps de la communication entre l'OS du robot et ROS (C'est que nous ferons plus tard dans §4.3.2 et §4.3.3), car elle est déjà établie grâce au driver. Mais, nous nous intéresserons à la simulation du robot, la génération de trajectoire et l'exploitation des fonctionnalités du mode cobot.

Tout au long du stage nous avons été confrontés à différentes problématiques pour atteindre nos objectifs. Nous faisons une analyse et expliquons les choix qui nous ont permis de surmonter ou dans certains cas, de contourner ces difficultés.

4.1 Crédit d'un environnement de simulation

L'environnement de simulation permet de visualiser le comportement du robot lorsque nous voulons tester de nouveaux modèles de commande, ou effectuer la détection de collisions sans mettre en danger l'opérateur ou endommager les éléments qui se trouvent dans la zone de travail.

Le dépôt Motoman (§1.3.2) a des paquets de support qui contiennent la configuration de la gamme des robots Yaskawa compatibles avec ROS-I. Parmi eux, il y a l'URDF (United Robotics Description Format) du modèle HC10, dont nous nous sommes servi afin de générer un paquet de configuration personnalisé pour une utilisation avec MoveIt selon [8] , [9] et [10].

Suivant la norme ROS-I, le paquet s'appelle **motoman_hc10_moveit_config**. Il considère les collisions parmi les axes, la chaîne cinématique, les limites physiques et de vitesse des joints de façon à représenter correctement le robot réel.

Pour initialiser l'environnement simulé, nous exécutons le fichier **demo.launch**. Ce dernier propage les données de configuration du robot dans les rosparams (§3.5) et ouvre une fenêtre de visualisation avec Rviz.

Nous montrons le résultat de cette procédure sur la figure 10. La correspondance entre les axes et les couleurs est la suivante:

- Axe X en rouge.
- Axe Y en vert.
- Axe Z en bleu.

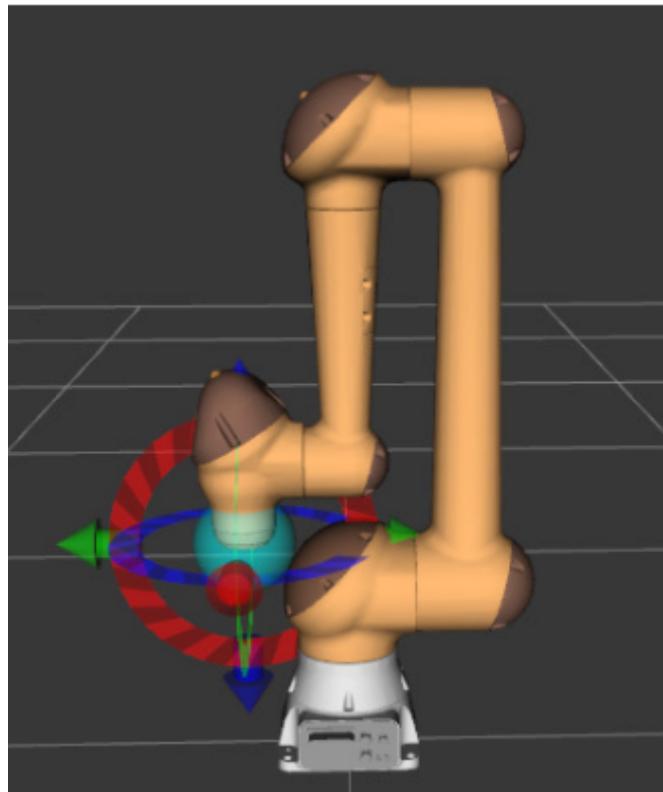


Figure 10. Robot HC10 simulé sur Rviz

4.2 Génération de trajectoire

Les entreprises qui automatisent leur processus de production à l'aide des robots industriels, réalisent souvent des procédures répétitives, qui exigent de la vitesse, de la précision et de la fiabilité.

Pour que le robot se déplace vers une position désirée depuis une position courante quelconque, nous devons considérer les obstacles dans la zone de travail, le type de déplacement (cartésien ou articulaire) et sa vitesse. A ce sujet, Moveit met à disposition plusieurs planificateurs, accessibles depuis le plug in « Motion planning ».

Nonobstant, à part l'évitement d'obstacles nous n'avons pas d'exigences spécifiques pour la planification des mouvements, alors nous utilisons le planificateur par défaut OMPL (Open Motion Planning Library), qui possède plusieurs algorithmes basés sur l'échantillonnage asymptotique optimal (asymptotic optimality en anglais). Cela veut dire que, les solutions retournées se basent sur l'échantillonnage, et le coût des trajectoires converge vers le coût de la solution optimale, car le nombre d'échantillons approche de l'infini.

Par ailleurs, nous avons créé le fichier « **moveit_config** **moveit_planning_execution.launch** » pour commencer à planifier à la fois les mouvements du robot simulé et du robot réel.

Les paramètres passés lors du lancement de ce fichier, déterminent lequel des deux reçoit la consigne de mouvement.

- Pour lancer une simulation locale du robot, nous utilisons la commande:
`roslaunch motoman_hc10_moveit_config moveit_planning_execution.launch`
- Pour se connecter au robot réel nous utilisons la commande:
`roslaunch motoman_hc10_moveit_config moveit_planning_execution.launch sim:=false controller:=yrc1000 robot_ip:=192.168.0.113`

Avec, « sim » le paramètre qui spécifie si nous sommes en simulation ou pas. « Controller », qui indique le nom du contrôleur. Et « robot_ip » qui permet la connexion au réseau du robot.

Nous illustrons sur les figures 11 et 12 l'architecture de communication des nœuds pour le robot simulé et le robot réel respectivement.

4.2.1 Génération de trajectoire avec le robot simulé

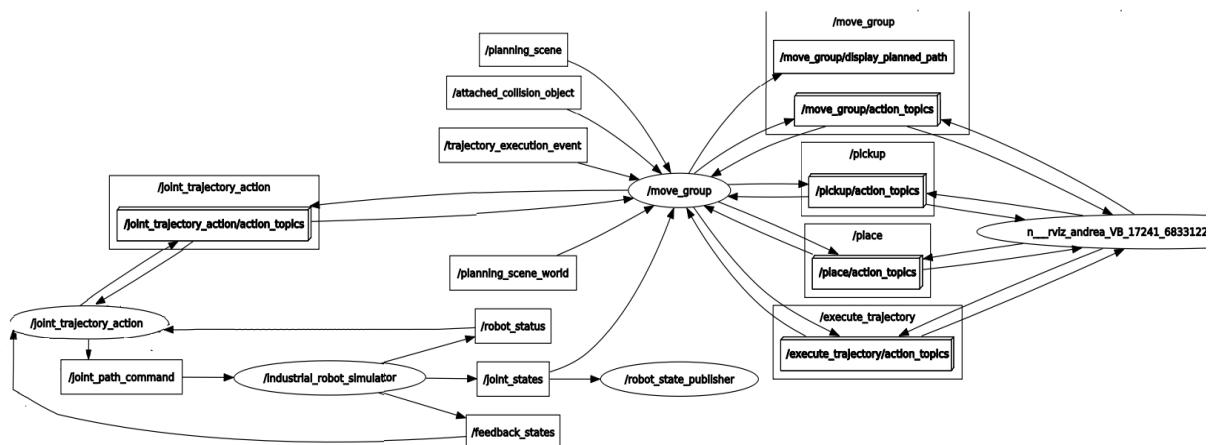


Figure 11. rqt_graph de la simulation locale

Le nœud **industrial_robot_simulator** simule le contrôleur d'un robot industriel en accord avec la norme ROS-I. Actuellement, le simulateur ne prend en charge que les exigences minimales pour la génération de trajectoire.

Nous retrouvons le « topic » **/joint_states**, qui transmet à Rviz la position et vitesse du robot pour créer une visualisation réaliste. **/feedback_states** qui publie la même information au nœud d'action. Et **/joint_path_command** par lequel sont transmis les points intermédiaires pour accomplir la trajectoire.

Enfin, nous avons le noeud **joint_trajectory_action** qui avertit le client de l'achèvement de l'action.

4.2.2 Génération de trajectoire avec le robot réel

En plus de l'exécution du fichier launch, pour envoyer des consignes il faut:

- Mettre le robot en mode distanciel (Remote Mode) car nous envoyons des commandes depuis un appareil externe et non du boîtier d'apprentissage.
- Habiliter l'envoie des commandes de mouvement vers le contrôleur avec l'appel à *rosservice call robot_enable*.

Une fois terminé, l'envoie des commandes se désactive en faisant appel à *rosservice call robot_disable*.

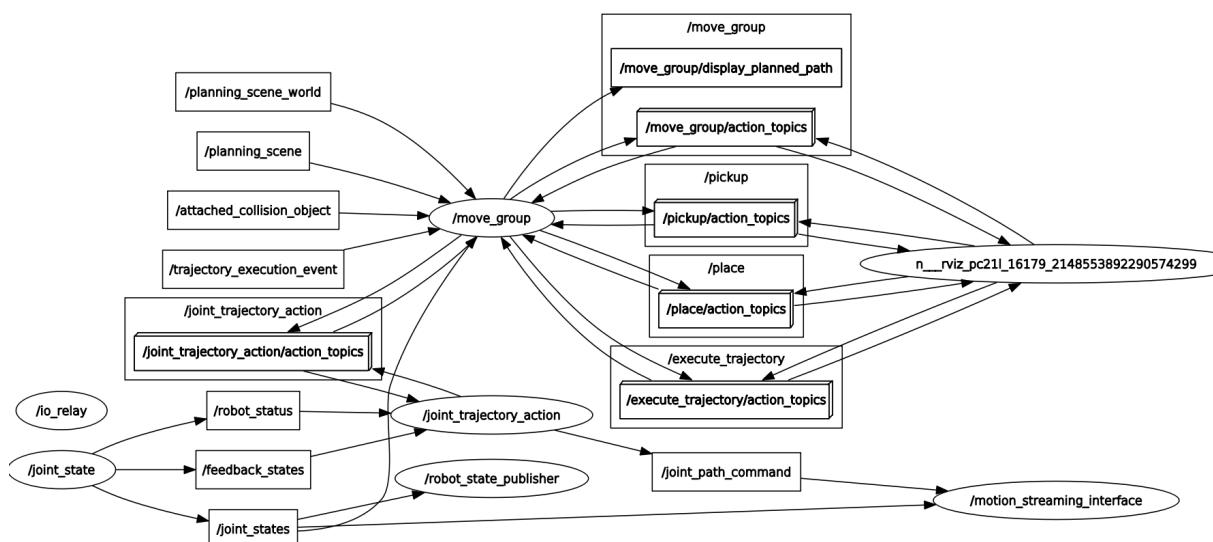


Figure 12. *rqt_graph de la connexion avec le robot réel*

Cette fois-ci, l'état du robot et le retour de position est recueilli depuis le contrôleur. Le noeud **joint_state** se charge de publier cette information aux topics qui supportent le déplacement, `/joint_state` et `/feedback_state`.

Ensuite, le noeud **motion_streaming_interface** transmet la consigne au contrôleur suivant le protocole « Simple Message ».

Nous pouvons voir sur la figure 13 la position courante du robot et sa représentation sur Rviz. La position désirée est dessinée en orange.

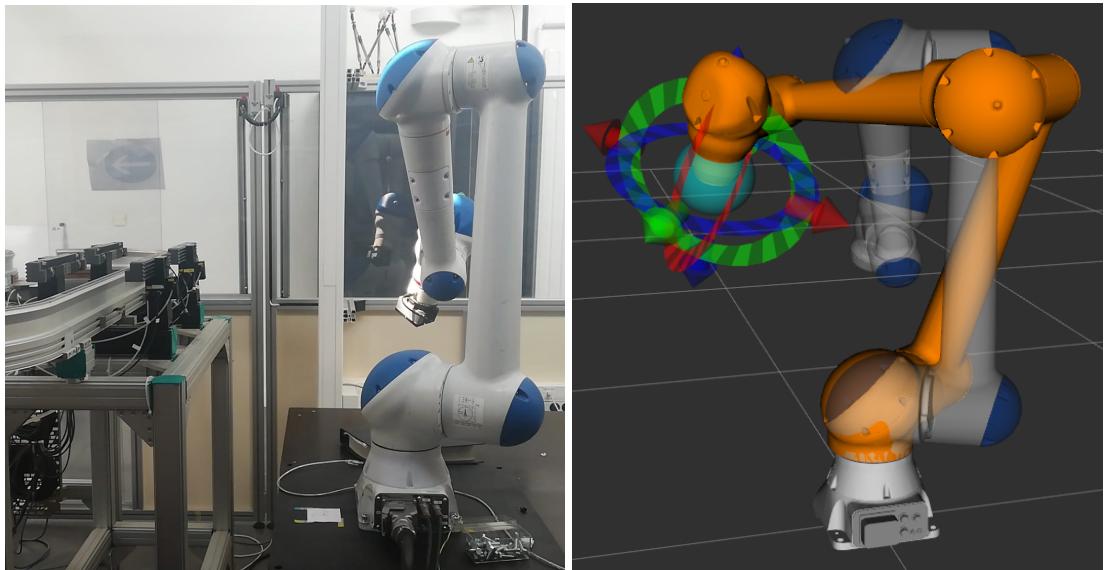


Figure 13. Représentation de la position courante (en blanc) et la position désirée (en orange) du robot sur Moveit

Ensuite, sur la figure 14 nous affichons le résultat après avoir envoyé la consigne.

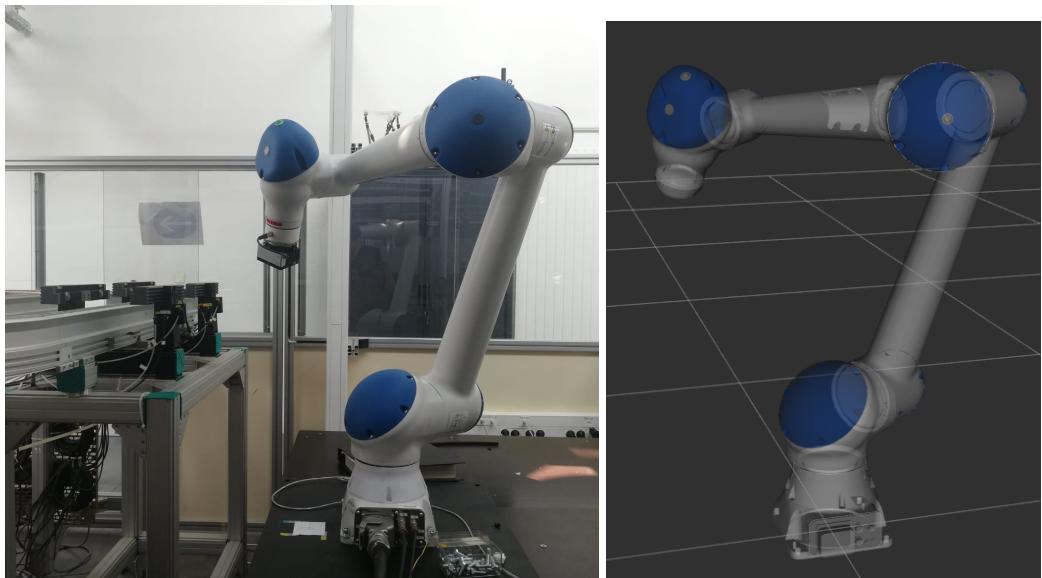


Figure 14. Représentation de la position courante du robot après l'exécution de la trajectoire sur Moveit

4.3 L'utilisation des efforts sur le robot Yaskawa HC10

Le cobot Yaskawa HC10, à comparaison du robot industriel conventionnel, admet une collaboration étroite avec l'homme. Il ne s'introduit pas comme un substitut, mais comme une aide bénéfique pour la production et le personnel.

Le contrôle en effort, utile en tant que mesure de sécurité comme mentionné dans §1.3.2; est aussi avantageux pour l'amélioration de la précision lorsque nous voulons développer une tâche industrielle qui exige une grande dextérité. Ou encore, la maîtrise des forces exercées par l'organe terminal pour ne pas abîmer les objets manipulés.

Nous présentons par la suite les démarches qui nous ont permis d'utiliser l'effort sur le robot.

4.3.1 Calibration du robot

Tout d'abord, nous avons réalisé la calibration des capteurs de couple dans l'objectif d'améliorer la précision des efforts mesurés.

En suivant le manuel d'utilisation, nous déplaçons le robot (À l'aide du boîtier de commande) vers différentes positions, pour que le contrôleur puisse recalculer les valeurs de couple (ou des efforts) en fonction de paramètres intrinsèques du robot, comme le poids, l'inertie, entre autres.

Sur la figure 15, nous montrons une photo du boîtier de commande après calibration.

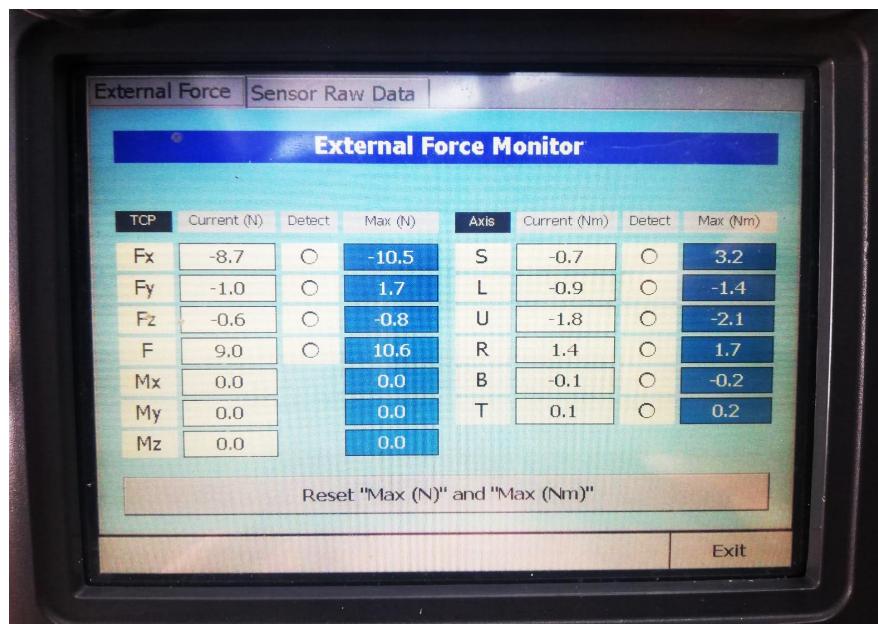


Figure 15. Mesure de l'effort après calibration sur la position d'origine du robot affiché sur le boîtier de commande.

Le couple des axes S-L-U-R-B-T (axes 1 à 6) est mesuré en newton-mètres [Nm]. La force estimée de l'organe terminal (TCP en anglais) se mesure en newtons [N].

4.3.2 Récupérer le retour en effort

Sur l'annexe 2, nous avons un tableau avec les identifiants de messages standard pour le protocole « Simple Message » et un autre avec les identifiants spécifiques au fournisseur.

Aucun de ces identifiants nous permet de récupérer les données liées à l'effort. Pourtant, il est détecté correctement par les capteurs. En plus, pour que la communication s'effectue, « Le modèle client/serveur exige que les deux comprennent la charge de données associée aux différents types de messages (msg_type) et de communications (comm_type) » [11]. (§3.3)

En résumé, nous avons 2 problèmes. D'une part, déterminer où s'enregistrent les données d'effort du robot. Et d'une autre part, transmettre ces données de façon à respecter le protocole « Simple Message » afin de pouvoir les utiliser dans une tâche décrite sur ROS.

Solution

Après une recherche dans les manuels d'utilisation du robot, nous déterminons les registres qui contiennent l'information associée à l'effort. Nous les affichons dans l'annexe 3.

C'est important de souligner que nous ne pouvons pas créer de nouveaux identifiants de message, au moins d'avoir une clé d'accès pour étendre la fonctionnalité du contrôleur. C'est pourquoi, nous récupérons l'effort en utilisant les messages spécifiques à Yaskawa.

Parmi les identifiants (sur l'annexe 2) qui nous donnent accès aux registres, nous avons:

ID	Nom	Commentaire
2003	ROS_MSG_MOTO_READ_IO_BIT	Read single I/O point (any address)
2004	ROS_MSG_MOTO_READ_IO_BIT_REPLY	-
2007	ROS_MSG_MOTO_READ_IO_GROUP	Read whole group (byte) of I/O (any address)

2008	ROS_MSG_MOTO_READ_IO_GROUP_REPLY	-
2012	ROS_MSG_MOTO_READ_MREGISTER	Read M-Register (M000 - M999)

Le protocole de communication exige que la réponse des données suit la convention de services ROS [11]. Alors, un identifiant est nécessaire pour réaliser une demande au contrôleur et un autre pour recevoir une réponse.

Les identifiants 2007 et 2008, ne nous conviennent pas, car même si plusieurs registres peuvent être lus en même temps, ceci est fait à condition que la valeur de chaque registre soit codée sur 1 bit. Cependant, la valeur d'un seul registre d'effort est codée au moins sur 2 octets.

Ensuite, les identifiants 2003, 2004, et 2012 nous permettent de lire correctement des valeurs codés sur 2 octets. Par contre, l'identifiant 2003 n'a pas accès à tous les registres se trouvant sur la plage M000-M999 (avec M = 1000). Nous rappelons que l'information associée à l'effort s'enregistre dans cette plage d'adresses (Annexe 3).

Ainsi, nous utilisons l'identifiant 2012 pour demander au contrôleur l'information contenu dans ses registres. Et l'identifiant 2004 pour récupérer cette information.

La figure 16 illustre l'échange entre le client et le serveur.

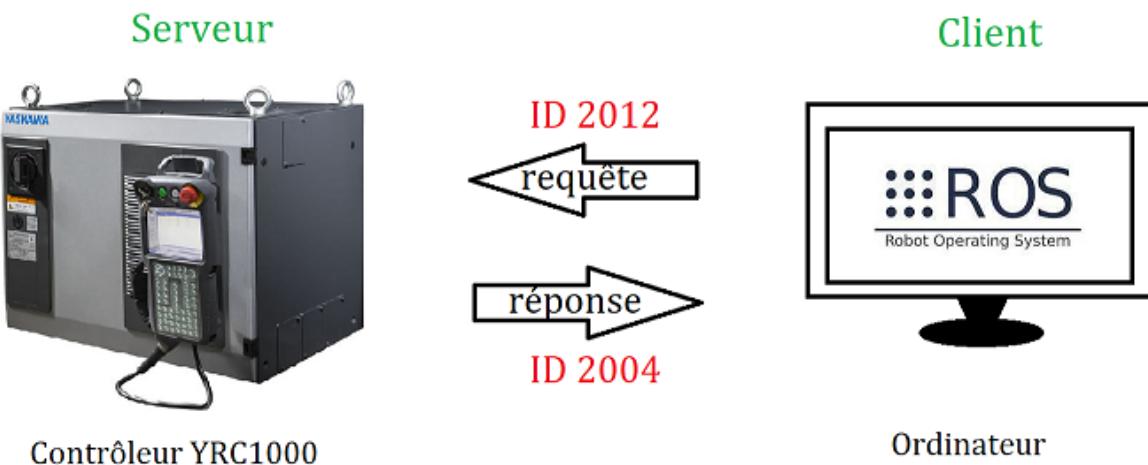


Figure 16. Représentation du serveur et client pour récupérer le retour en effort

Nous relevons que les valeurs des registres d'effort doivent passer par une méthode de conversion pour récupérer les couples en newton-mètres et la force en newtons.

4.3.3 Récupérer le retour en position et vitesse de l'organe terminal

Pour réaliser une manipulation en utilisant les efforts, nous devons maîtriser la force/couple exercé sur l'organe terminal. Néanmoins, le nœud **joint_state** (vu dans §4.2.2) qui se charge de publier l'état du robot, transmet seulement les données de position et de vitesse des joints.

Par ailleurs, la documentation du robot ne contient pas les adresses dans lesquelles les valeurs en position et en vitesse sont enregistrées. Néanmoins, nous pouvons rediriger ces valeurs vers les registres de notre choix, sur la plage M000-M999.

A titre d'exemple, l'annexe 4 montre la fonction permettant de rediriger la position de l'organe terminal.

Comme le retour en position et en vitesse se trouvent dans la même plage d'adresses que le retour en effort, nous utilisons les mêmes identifiants que dans §4.3.2. de façon à recueillir ces données.

Pour finir, même si nous nous intéressons aux données de l'organe terminal, nous décidons aussi de récupérer les valeurs de position et de vitesse des joints à travers les registres.

Dans l'objectif d'avoir un vue d'ensemble de l'état du robot, nous créons des messages personnalisés pour la position, la vitesse et l'effort. (Voir §4.3.5)

4.3.4 Mapping de la mémoire yaskawa

Dans la section §4.3.2 et §4.3.3 nous accédons aux registres internes du robot afin de récupérer son état courant. Pour réaliser cela, nous devons indiquer à chaque fois l'adresse du registre auquel nous voulons accéder.

Du point de vue de la programmation, la tâche de lire et écrire les adresses une par une peut devenir assez complexe. Alors, pour faciliter la lecture du code, nous avons défini des mots clés, pour parcourir les différentes plages d'adresses.

Nous montrons le mapping de la mémoire sur l'annexe 5. Ce document pourra être utilisé comme une base pour de futures utilisations.

4.3.5 Crédation des messages personnalisés

Nous créons des messages personnalisés afin de connaître l'état du robot. Sur les figures 17, 18, 19, nous montrons la structure des messages contenant l'information de l'effort, la position et la vitesse respectivement.

Ces messages s'adaptent à la communication inter-processus propre à l'environnement ROS. (§2)

```

#Couple des axes
float32[] CoupleJoints

#Couple de la TCP
float32[] CoupleTCP

#Forces de la TCP
float32[] ForceTCP

#Force resultante de la TCP
float32 ForceTotaleTCP

```

Figure 17. Structure du message pour le retour en effort

```

#Position des axes en degrés
float32 pos_s
float32 pos_l
float32 pos_u
float32 pos_r
float32 pos_b
float32 pos_t
#Position du manipulateur en mm
float32 pos_x
float32 pos_y
float32 pos_z
#Orientation du manipulateur en degrés
float32 rot_x
float32 rot_y
float32 rot_z
float32 rot_e

```

Figure 18. Structure du message pour le retour en position

```

#Vitesse des axes en 0.0001 deg/sec
float32 vit_s
float32 vit_l
float32 vit_u
float32 vit_r
float32 vit_b
float32 vit_t
#Vitesse de la TCP
float32 vit_tcp

```

Figure 19. Structure du message pour le retour en vitesse

Les variables des joints sont nommées suivant la convention pour un robot 6 axes de Yaskawa. Alors, S-L-U-R-B-T correspond aux axes de 1 à 6 du robot.

Le type *float32* est attribué à une valeur flottante codée sur 32 bits.

Le type *float32[]* est attribué à un tableau des flottantes codées sur 32 bits.

5. Validation de la solution

Nous montrons par la suite les résultats de solutions présentés dans la partie précédente. En passant d'abord, par l'affichage de l'effort et la position mesuré et par la simulation d'une tâche industrielle.

La variation de la vitesse étant assez rapide, nous la validons de manière empirique.

5.1 Lecture et affichage des mesures

Nous montrons sur les figures 20 et 21 le retour en position mesuré sur le boîtier de commande et le logiciel rqt (utilisé sur ROS pour afficher l'information véhiculée à travers le topics). Pareillement, nous montrons le retour en effort sur les figures 22 et 23.

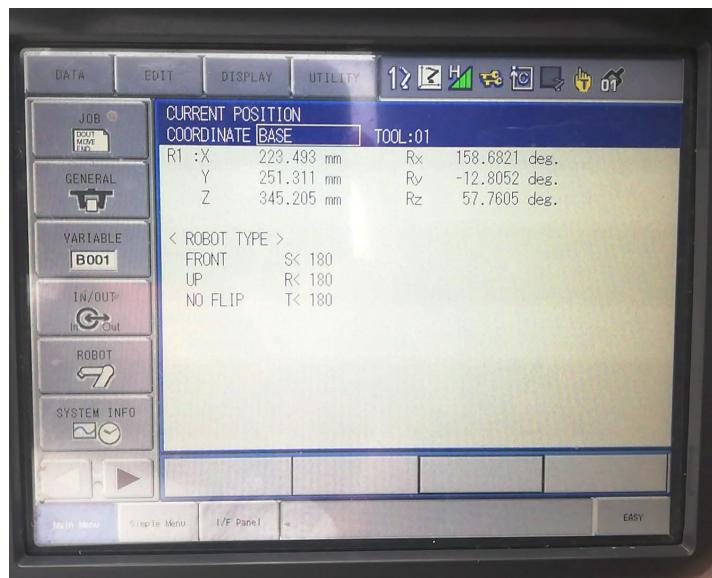


Figure 20. Retour en position affiché sur le boîtier de commande

Default - rqt				
Topic	Type	Bandwidth	Hz	Value
/attached_collision_object	moveit_msgs/AttachedCollisionObject			not monitored
/execute_trajectory/cancel	actionlib_msgs/GoalID			not monitored
/execute_trajectory/feedback	moveit_msgs/ExecuteTrajectoryActionFeedback			not monitored
/execute_trajectory/goal	moveit_msgs/ExecuteTrajectoryActionGoal			not monitored
/execute_trajectory/result	moveit_msgs/ExecuteTrajectoryActionResult			not monitored
/execute_trajectory/status	actionlib_msgs/GoalStatusArray			not monitored
/feedback_states	control_msgs/FollowJointTrajectoryFeedback			not monitored
/joint_effort	motoman_msgs/Effort			not monitored
/joint_path_command	trajectory_msgs/JointTrajectory			not monitored
/joint_position	motoman_msgs/Position	414.04B/s	7.89	
pos_b	float32			-64.404296875
pos_l	float32			-21.534700393676758
pos_r	float32			14.814599990844727
pos_s	float32			30.827600479125977
pos_t	float32			20.4768009185791
pos_u	float32			63.10770034790039
pos_x	float32			223.48800659179688
pos_y	float32			251.3059975585938
pos_z	float32			345.1910095214844
rot_e	float32			0.0
rot_x	float32			158.6822052001953
rot_y	float32			-12.803799629211426
rot_z	float32			57.760501861572266

Figure 21. Retour en position affiché sur rqt

Sur la figure 21, les valeurs des variables pos_x, pos_y, et pos_z correspondent à la position en mm de l'organe terminal et rot_x, rot_y, et rot_z à son orientation en degrés. (Vu dans §4.3.5)

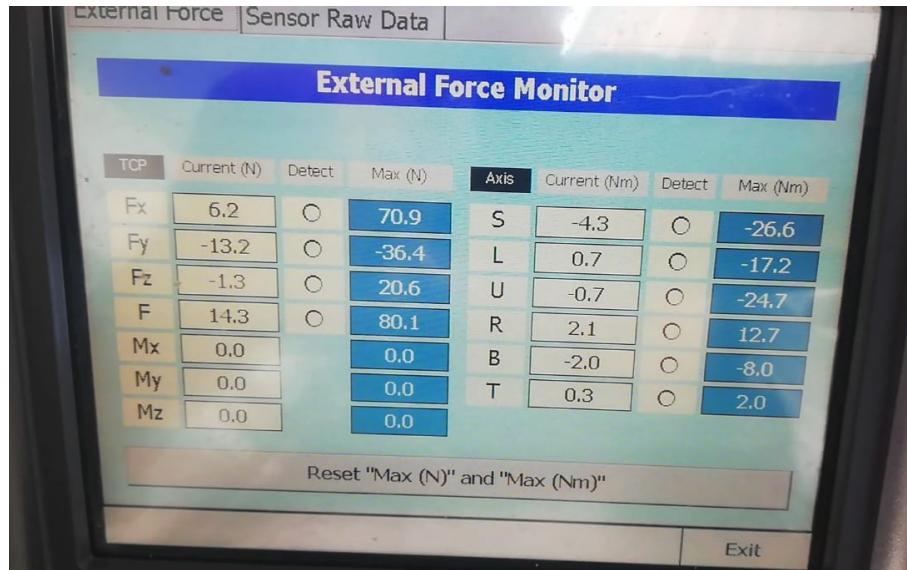


Figure 22. Retour en effort affiché sur le boîtier de commande

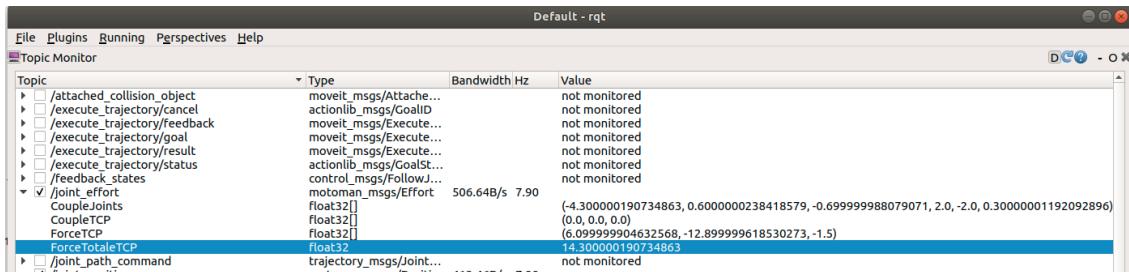


Figure 23. Retour en effort affiché sur rqt

Le retour en effort affiché sur la figure 23, présente une légère différence par rapport aux valeurs affichées sur la figure 22 (Boîtier de commande). La figure 22 contient des valeurs arrondis alors que dans la figure 23, les chiffres après la virgule sont considérés.

Tout de même, la force totale estimée de l'organe terminal (soulignée en bleu dans la figure 23) reste toujours la même. Alors, cette petite variation dans les valeurs ne sera pas considérée comme une erreur de mesure.

5.1 Simulation d'une tâche

Nous désirons créer une tâche d'inspection industrielle qui viendra intégrer le travail réalisé. C'est-à-dire, la génération de trajectoire (§4.2), le retour en position, en vitesse et en effort (§4.3.2 et §4.3.3). A cette fin, nous avons développé des « blocs » élémentaires nous permettant de le faire.

Dans un premier temps, le robot doit être capable d'exécuter ces trajectoires sans risque, alors nous simulons la zone de travail. Ensuite, la tâche doit être capable de se répéter plusieurs fois, c'est pourquoi nous définissons des points de passage. De plus, nous voulons nous assurer de la sécurité de l'opérateur ou des objets se trouvant dans la zone de travail qui n'ont pas été considérées dans la simulation, donc nous créons un chien de garde qui nous indiquera si l'organe terminal exerce un effort au-delà d'un seuil donné lorsqu'une trajectoire est en train d'être exécuté.

Dû au contexte sanitaire et à la fermeture de l'atelier imposant une période de télétravail, l'intégration de tous les blocs n'a pas pu être faite. Néanmoins, nous avons vérifié leur validité.

Detection d'obstacles

Le planificateur Moveit permet de récréer la zone de travail dans la simulation. Les espaces verts sont considérés comme des obstacles. Lorsque nous planifions une trajectoire et que la configuration finale du robot se retrouve dans cette zone, alors les joints sont dessinés en couleur rouge pour indiquer la collision potentielle. Nous pouvons le voir sur la figure 24.

Par ailleurs, la planification échoue car le planificateur ne trouve pas de solutions pour atteindre cette position (rectangle rouge).

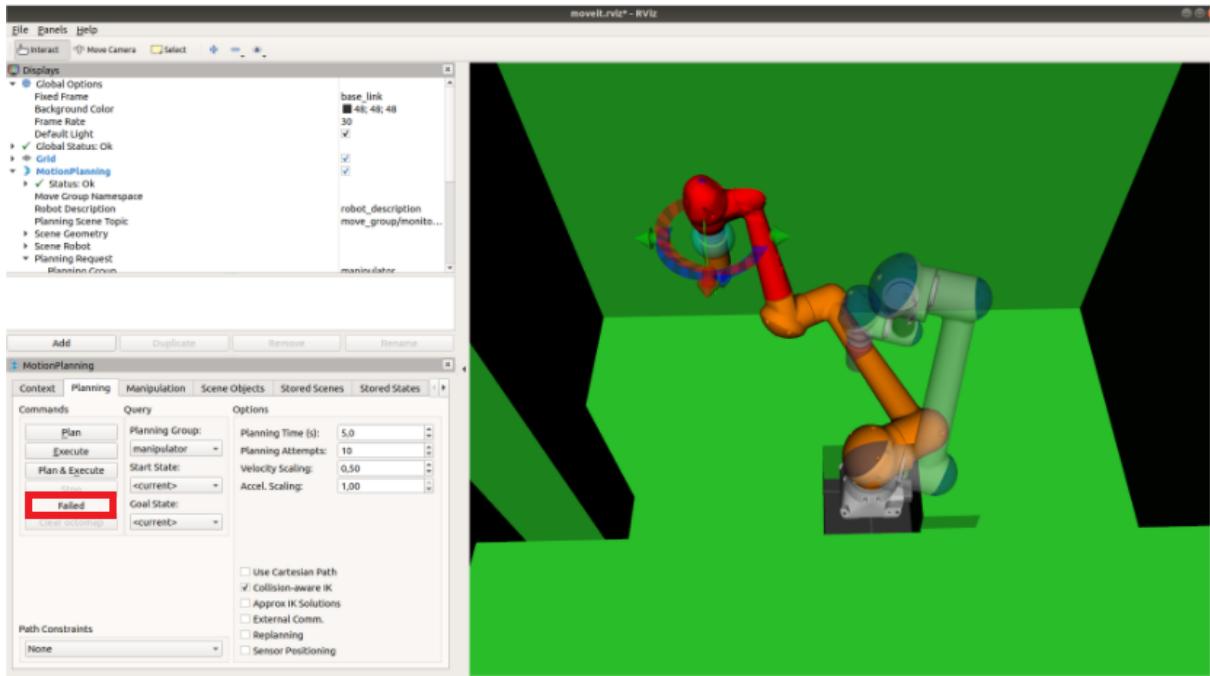


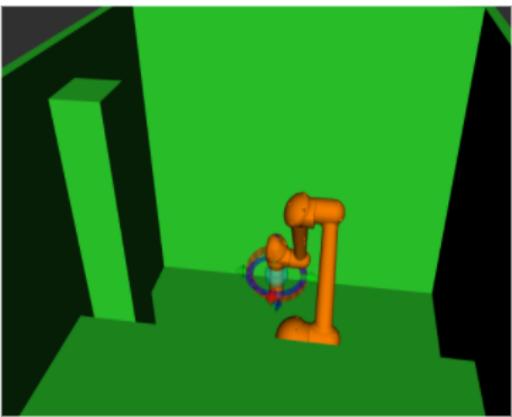
Figure 24. Planification vers une collision potentiel

Définition des différentes configurations

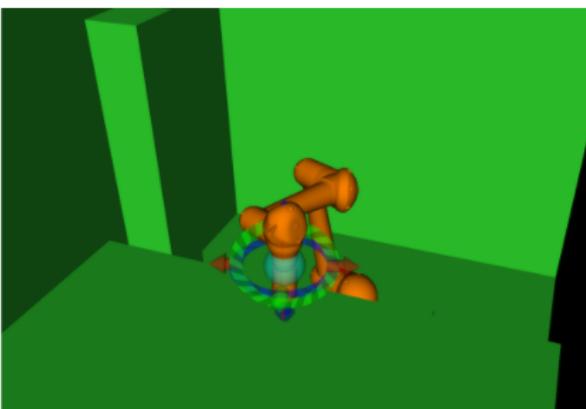
Nous remarquons que lors de l'exécution des tâches robotiques, le robot passe souvent par les mêmes points. Généralement les points sont enregistrés auparavant, c'est qui permet après d'indiquer leur ordre d'exécution dans une boucle while qui s'arrêtera uniquement lorsque l'opérateur en aura besoin.

Ainsi, en considérant la zone de travail, nous enregistrons différents points de passage, « to_discard », « Pick_n_Place et » et « to_Operator ». Nous illustrons ces points sur la figure 25.

Position: To_discard



Position: Pick_n_Place



Position: To_Operator

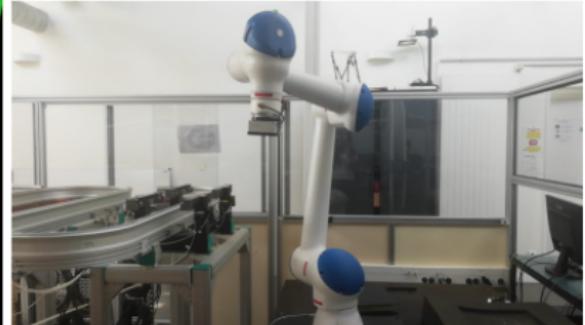


Figure 25. Points de passage

Détection du dépassement de seuil d'effort

Le planificateur Moveit se charge de calculer les trajectoires vers les différents points de passage. Néanmoins, le logiciel ne considère pas l'apparition des nouveaux objets dans la zone de travail tant qu'ils ne sont pas décrits en simulations.

Alors, par mesure de sécurité nous créons un chien de garde veillant à ce que l'effort de l'organe terminal ne dépasse pas un seuil donné. Si lors de manipulations le seuil est dépassé, cela pourrait indiquer qu'un objet ou une personne est rentré en contact avec l'organe terminal et dans ce cas là il faudrait arrêter l'exécution de la trajectoire.

Sur la figure 26, nous montrons les messages affichés par le chien de garde lorsqu'il dépasse un seuil de 60N.

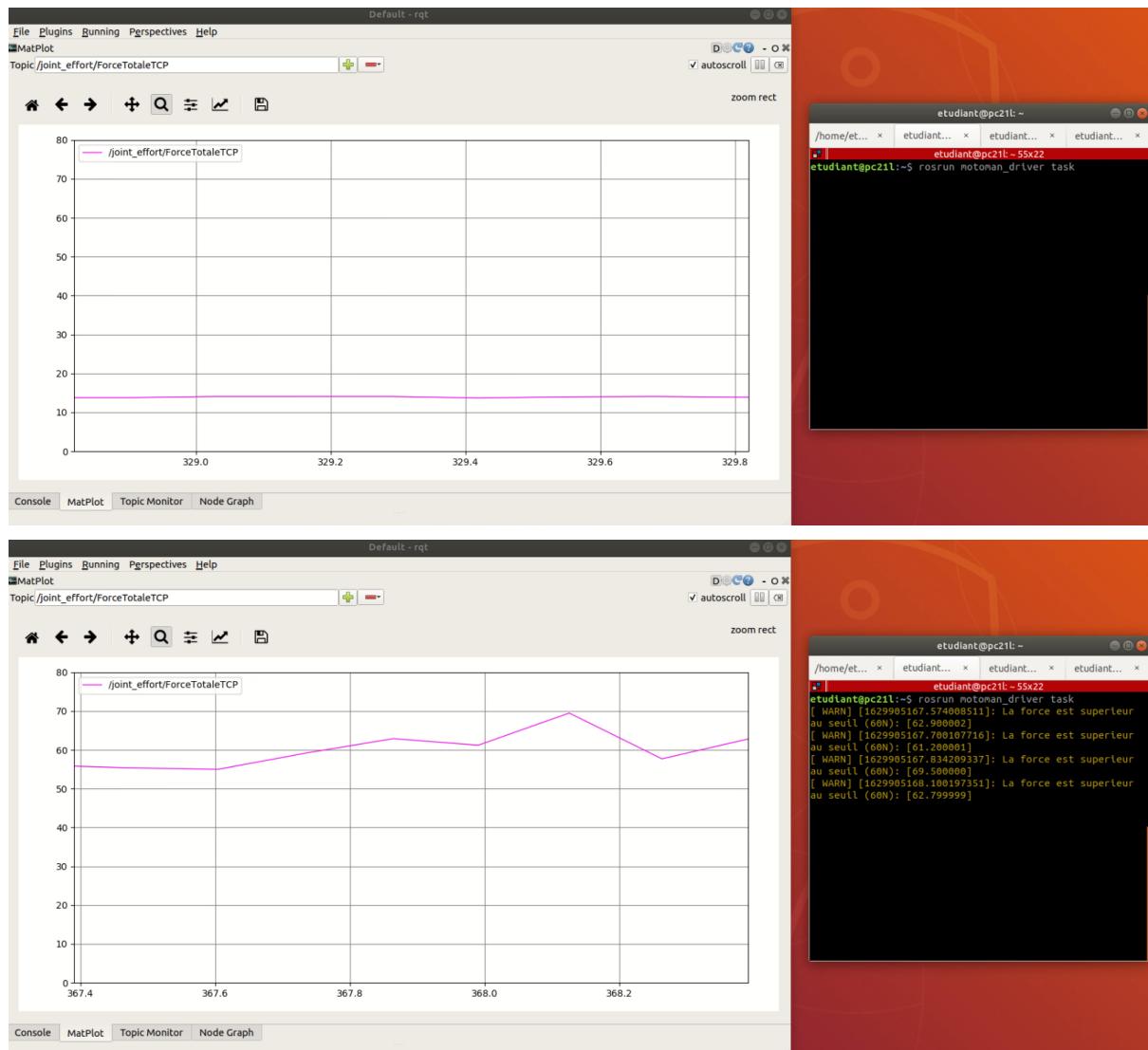


Figure 26. Exécution du chien de garde.

Conclusion

L'utilisation des logiciels Open Source représente un véritable avantage pour le développement des nouvelles technologies. La création d'une communauté qui vise à résoudre différentes problématiques autour d'un même sujet amène à la contribution des idées innovantes par des personnes avec des formations différentes. ROS fait preuve de cela.

Le travail que j'ai effectué au sein de l'AIP m'a permis de tirer profit de ce middleware et de mettre en évidence sa puissance face à des applications robotiques industrielles avec le robot Yaskawa HC10.

Malgré le fait de ne pas connaître l'environnement de développement et le langage natif du robot, j'étais capable de le simuler, le commander et récupérer les données de ses capteurs à travers ROS comme si c'était le cas. Grâce aux compétences techniques et théoriques que j'ai acquises tout au long de ma formation j'ai pu discerner et résoudre les problématiques auxquelles j'étais confronté lors de la réalisation des procédures mentionnées ci-dessus.

Basés à nouveau sur l'idéologie Open Source, le travail réalisé pendant ce stage a été réfléchi pour être réutilisé par l'atelier dans ses formations ROS. En sachant aussi que le robot n'était pas encore intégré sur l'environnement du middleware lors de mon arrivée en stage, ce travail vise aussi à donner un premier aperçu des étapes à suivre pour l'intégration des robots ayant un driver en accord avec la norme ROS-I.

Ce stage a été vraiment enrichissant non seulement d'un point de vue académique mais aussi professionnel que personnel. Avoir travaillé dans un domaine en plein essor comme le développement logiciel avec des systèmes Open Source a enrichi considérablement mon parcours. J'ai également pu approfondir mes connaissances dans le domaine de la robotique ainsi qu'adopter une méthodologie de travail rigoureuse me permettant d'atteindre mes objectifs. En outre, l'autonomie et le sens critique que j'ai pu développer lors de mon stage, sont des capacités transversales que j'apprécie énormément.

Bibliographie

- [1] The Linux Foundation. “Software-defined vertical industries: transformation through open source” Septembre 2020
<https://www.linuxfoundation.org/wp-content/uploads/software-defined-vertical-industries-092520.pdf> [Consulté le 17 juillet 2021]
- [2] ROS Industrial. “Our Brief History.”,<https://rosindustrial.org/briefhistory> [Consulté le 17 juillet 2021]
- [3] AIP-Primeca. “Une ROS Academy sur le pôle S-MART Occitanie.”
<https://www.aip-primeca-occitanie.fr/ros-academy/> [Consulté le 20 juillet 2021],
- [4] ZDNet. “How open-source software transformed the business world” September 24, 2020
<https://www.zdnet.com/article/how-open-source-software-transformed-the-business-world/> [Consulté le 20 juillet 2021]
- [5] YASKAWA. “ROS-Enabled and Ready to Go!” Septembre 30, 2019
<https://www.motoman.com/en-us/about/y-blog/ros-enabled-ready-to-go>
[Consulté le 1 août 2021]
- [6] ROS.org “Motoman driver” Octobre 15, 2020 http://wiki.ros.org/motoman_driver [Consulté le 25 juillet 2021]
- [7] ROS industrial “Message Structures of the ROS-Industrial Simple Message Protocol” 16 Janvier 2018 <https://github.com/ros-industrial/rep/blob/master/rep-0006.rst> [Consulté le 20 août 2021]
- [8] ROS.org “Create a MoveIt Package for an Industrial Robot” 6 Juillet 2018
http://wiki.ros.org/Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot
[Consulté le 4 avril 2021]
- [9] ROS.org “MoveIt Setup Assistant”
http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistantTutorial.html [Consulté le 5 avril 2021]
- [10] ROS industrial “Build a MoveIt! Package”
https://industrial-training-master.readthedocs.io/en/melodic/_source/session3/Build-a-MoveIt!-Package.html [Consulté le 5 avril 2021]
- [11]ROS.org. “simple_message” 23 mai 2021 http://wiki.ros.org/simple_message [Consulté le 12 mai 2021]

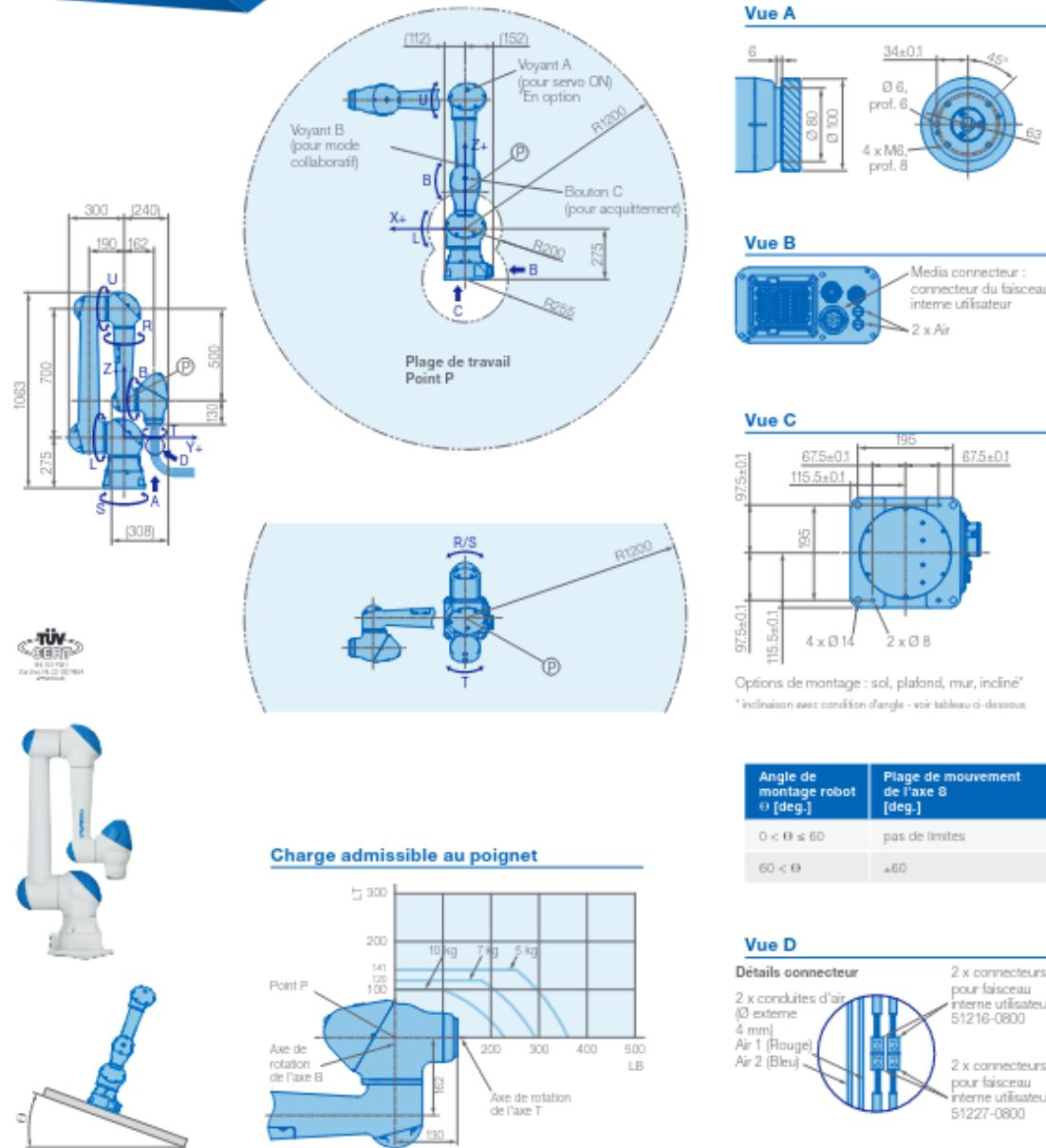
Références Images

- [i] Extrait du site: <https://s-mart.fr/collectif-academique-industrie-du-futur/>
- [ii] Extrait du site: <https://www.aip-primeca-occitanie.fr/ros-in-occitanie/>
- [iii] Extrait du site: <https://www.aip-primeca-occitanie.fr/plateforme-production/>
- [iv] Extrait du site:
https://www.yaskawa.fr/Global%20Assets/Downloads/Brochures_Catalogues/Robotics/MOTOMAN_Robots/HC10_HC10DT/Flyer_Robot_HC10_HC10DT_FR_12.2020.pdf
- [v] Extrait du site: https://fr.wikipedia.org/wiki/Robot_Operating_System
- [vi] Extrait du site: <https://rosindustrial.org/news/tag/Motoman+Robotics+Division>
- [vii] Extrait du site:
<https://blog.robotiq.com/bid/70845/What-is-ROS-and-ROS-Industrial-for-Robots>
- [viii] Extrait du site: http://wiki.ros.org/motoman_driver

Annexes

Annexe 1. Fiche technique du robot Yaskawa HC10 extrait du [iv]

MOTOMAN HC10



Spécifications HC10						
Axes	Amplitude maximale de mouvement [°]	Vitesse maximale [°/sec.]	Couple autorisé [Nm]	Moment d'inertie autorisé [kg · m²]	Axes contrôlés	6
S	±180	130	–	–	Max. charge admissible [kg]	10
L	±180	130	–	–	Répétabilité [mm]	±0,1
U	+355/-5	180	–	–	Max. plage de travail R [mm]	1200
R	±180	180	27,4	0,78	Température [°C]	0 à +40* 0 à +35**
B	±180	250	27,4	0,78	Humidité [%]	20 - 80
T	±180	250	9,8	0,1	Poids [kg]	47
					Alimentation élec., moyenne [kW]	1

* en fonctionnement ** lors de l'utilisation de la housse de protection (optionnelle)

Annexe 2. Identifiants de messages pour le protocole de message simple. Extrait du [7]

Pour les messages standard:

ID	Name	Comment
0	-	Reserved
1	PING	-
2	GET_VERSION	Retrieve version of the current driver
3-9	-	Reserved for future use
10	JOINT_POSITION	Deprecated, also 'JOINT'
11	JOINT_TRAJ_PT	-
12	JOINT_TRAJ	-
13	STATUS	-
14	JOINT_TRAJ_PT_FULL	-
15	JOINT_FEEDBACK	-
16-19	-	Reserved for future use
20	READ_INPUT	Deprecated
21	WRITE_OUTPUT	Deprecated
22-999	-	Reserved for future use
1000-2999	-	Vendor specific
3000-64999	-	Reserved for future use
65000-65535	-	Freely assignable
65536-2147483647	-	Reserved for future use

Pour les messages spécifiques aux robots Yaskawa:

Motoman		
ID	Name	Comment
2001	MOTOMAN_MOTION_CTRL	Control motion-server settings/options
2002	MOTOMAN_MOTION_REPLY	-
2003	ROS_MSG_MOTO_READ_IO_BIT	Read single I/O point (any address)
2004	ROS_MSG_MOTO_READ_IO_BIT_REPLY	-
2005	ROS_MSG_MOTO_WRITE_IO_BIT	Write single input point ('Network Inputs' only)
2006	ROS_MSG_MOTO_WRITE_IO_BIT_REPLY	-
2007	ROS_MSG_MOTO_READ_IO_GROUP	Ready whole group (byte) of I/O (any address)
2008	ROS_MSG_MOTO_READ_IO_GROUP_REPLY	-
2009	ROS_MSG_MOTO_WRITE_IO_GROUP	Write whole group (byte) of 'Network Inputs'
2010	ROS_MSG_MOTO_WRITE_IO_GROUP_REPLY	-
2011	ROS_MSG_MOTO_ICTRL_REPLY	-
2012	ROS_MSG_MOTO_READ_MREGISTER	Read M-Register (M000 through M999)
2013	ROS_MSG_MOTO_WRITE_MREGISTER	Write M-Register (M000 through M599)
2014-2015	-	Reserved for future use
2016	ROS_MSG_MOTO_JOINT_TRAJ_PT_FULL_EX	Command multiple control-groups with a single message
2017	ROS_MSG_MOTO_JOINT_FEEDBACK_EX	Get feedback position of multiple control-groups with a single message
2018	ROS_MSG_MOTO_SELECT_TOOL	Set the active tool file (for PFL function)
2019	-	Reserved for future use
2020	ROS_MSG_MOTO_GET_DH_PARAMETERS	Get DH parameters for all control-groups connected to the robot controller
2021-2099	-	Reserved for future use

Annexe 3. Tableau des registres d'effort et méthode de conversion. Extrait du manuel d'utilisation

HW1484764

Register List

10. Register List

If using the register, refer to "YRC1000 OPTIONS INSTRUCTIONS FOR Concurrent I/O (RE-CKI-A467)" or "YRC1000micro OPTIONS INSTRUCTIONS FOR Concurrent I/O (RE-CKI-A469)".

However, do not use the following registers because they are used for the collaborative operations.

M309	M308	M307	M306	M305	M304	M303	M302	M301	M300
Reserved									

M319	M318	M317	M316	M315	M314	M313	M312	M311	M310
Reserved	Reserved	Reserved	Reserved	Estimated external torque value T	Estimated external torque value B	Estimated external torque value R	Estimated external torque value U	Estimated external torque value L	Estimated external torque value S

M329	M328	M327	M326	M325	M324	M323	M322	M321	M320
Reserved	Reserved	Reserved	Estimated TCP external force resultant	Estimated TCP external force Mz	Estimated TCP external force My	Estimated TCP external force Mx	Estimated TCP external force Fz	Estimated TCP external force Fy	Estimated TCP external force Fx

M339	M338	M337	M336	M335	M334	M333	M332	M331	M330
Reserved	Reserved	Reserved	Reserved	Sensor data T (CH1)	Sensor data B (CH1)	Sensor data R (CH1)	Sensor data U (CH1)	Sensor data L (CH1)	Sensor data S (CH1)

M349	M348	M347	M346	M345	M344	M343	M342	M341	M340
Reserved	Reserved	Reserved	Reserved	Sensor data T (CH2)	Sensor data B (CH2)	Sensor data R (CH2)	Sensor data U (CH2)	Sensor data L (CH2)	Sensor data S (CH2)

M359	M358	M357	M356	M355	M354	M353	M352	M351	M350
Reserved	Reserved	Reserved	Reserved	External torque T: Maximum value	External torque B: Maximum value	External torque R: Maximum value	External torque U: Maximum value	External torque L: Maximum value	External torque S: Maximum value

M369	M368	M367	M366	M365	M364	M363	M362	M361	M360
Reserved	Reserved	Reserved	External force resultant: Maximum value	TCP external force Mz: Maximum value	TCP external force My: Maximum value	TCP external force Mx: Maximum value	TCP external force Fz: Maximum value	TCP external force Fy: Maximum value	TCP external force Fx: Maximum value

Each register value is expressed in units of 0.1 [N] (or [Nm]) and output by adding an offset of 10000.

For how to interpret the value, refer to the following examples.

Example:

Register	Value	External force (interpretation)	Calculation method
M310	9500	-50.0 [Nm]	(9500 - 10000)×0.1 = -50.0[Nm]
M321	11055	105.5[N]	(11055 - 10000)×0.1 = 105.5[N]

Annexe 4. Fonction pour rediriger les valeurs des positions de l'organe terminal vers d'autres adresses. Extrait du manuel d'utilisation

	6 Convenient Functions
	6.12 Present Manipulator Position Output Function

6.12 Present Manipulator Position Output Function

6.12.1 Function for Outputting Present Cartesian Position of Manipulator to Register

6.12.1.1 Outline

The present Cartesian position of the manipulator (values in the base coordinates) is output to the specified registers.

6.12.1.2 Parameters

The following parameters specify the details of the function and output register numbers.

S1CxG	Description
208	Enables/Disables the function for outputting the present Cartesian position (in the base coordinates) to registers. (command value) 0: disable 1: enable
209	Specifies the output size to the register. 0: output in 2 bytes 1: output in 4 bytes
210	Cartesian position (command value) X register number of output destination
211	Cartesian position (command value) Y register number of output destination
212	Cartesian position (command value) Z register number of output destination
213	Cartesian position (command value) Rx register number of output destination
214	Cartesian position (command value) Ry register number of output destination
215	Cartesian position (command value) Rz register number of output destination
216	Cartesian position (command value) Re register number of output destination
217	Enables/Disables the function for outputting the present Cartesian position (in the base coordinates) to registers. (FB value) 0: disable 1: enable
218	Specifies the output size to the register. 0: output in 2 bytes 1: output in 4 bytes
219	Cartesian position (FB value) X register number of output destination
220	Cartesian position (FB value) Y register number of output destination
221	Cartesian position (FB value) Z register number of output destination
222	Cartesian position (FB value) Rx register number of output destination
223	Cartesian position (FB value) Ry register number of output destination
224	Cartesian position (FB value) Rz register number of output destination
224	Cartesian position (FB value) Re register number of output destination

<Example 2>

S1C1G	Setting value
217	1
218	1
219	10
220	12
221	14
222	16
223	18
224	20
225	22

When the parameters are set as shown in the above table, the present position is output to the registers as follows:

- M010 = Lower 2 bytes of the manipulator's present Cartesian position (FB value) X [unit: μm]
- M011 = Upper 2 bytes of the manipulator's present Cartesian position (FB value) X [unit: μm]
- M012 = Lower 2 bytes of the manipulator's present Cartesian position (FB value) Y [unit: μm]
- M013 = Upper 2 bytes of the manipulator's present Cartesian position (FB value) Y [unit: μm]
- M014 = Lower 2 bytes of the manipulator's present Cartesian position (FB value) Z [unit: μm]
- M015 = Upper 2 bytes of the manipulator's present Cartesian position (FB value) Z [unit: μm]
- M016 = Lower 2 bytes of the manipulator's present Cartesian position (FB value) Rx [unit: 0.001 deg]
- M017 = Upper 2 bytes of the manipulator's present Cartesian position (FB value) Rx [unit: 0.001 deg]

Annexe 5. Mapping de la mémoire Yaskawa

Page 1 sur 7

Definition	Nom	Description	Plage d'adresses associés
0 xxx	JOB_I_	Instruction JOB d'entrée	00010 - 05127 (4096 signaux)
1 xxx	JOB_O_	Instruction JOB de Sortie	10010 - 15127 (4096 signaux)
2 xxx	EXT_TEACH_I_	Entrée externe du teach	20010 - 25127 (4096 signaux)
3 xxx	EXT_TEACH_O_	Sortie externe du teach	30010 - 35127 (4096 signaux)
4 xxx	CONF_I_	Configuration du robot en entrée	40010 - 42567 (2048 signaux)
5 xxx	CONF_O_	Configuration du robot en sortie	50010 - 55127 (4096 signaux)
6 xxx	PANEL_	à definir ---	60010 - 60647 (512 signaux)
7 xxx	RELAY_AUX_	Relais auxiliaire (Moteur,...)	70010 - 79997 (7992 signaux)
80 xxx à 85 xxx	STATUS_	Status du robot	80010 - 85127 (4096 signaux)
87 xxx	SYS_I_	Entrée système	87010 - 87207 (160 signaux)
27 xxx à 29 xxx	NET_I_	Entrées réseau	27010 - 29567 (2048 signaux)
37 xxx à 39 xxx	NET_O_	Sorties réseau	37010 - 39567 (2048 signaux)
M xxx	MREGISTER_	Registre Cobot du Robot	M000 - M999 (1000 signaux)

Annexe 5. Page 2 sur 7

MREGISTER_

Définition	Nom	Description
M 000 à M 559	MREGISTER_RESERVE_	Réservé par Yaskawa (Non défini)
M 560 à M 599	MREGISTER_ANA_O_	Signaux analogiques de sortie
M 600 à M 639	MREGISTER_ANA_I_	Signaux analogiques d'entrée. (Facteur de conversion à définir)
M 640 à M 999	MREGISTER_SYS_	Registre système du mode cobot

Annexe 5. Page 3 sur 7

MREGISTER_RESERVE_

Registres en rouge définis par Yaskawa. Ne pas modifier.

Définition	Nom	Description
M010 & M011	MREGISTER_RESERVE_POS_S	Position des axes (FB) [Unité: 0.0001 deg]
M012 & M013	MREGISTER_RESERVE_POS_L	Valeur codée sur 4 octets. Le premier registre correspond aux 2 octets inférieurs. Ex. M010
M014 & M015	MREGISTER_RESERVE_POS_U	
M016 & M017	MREGISTER_RESERVE_POS_R	
M018 & M019	MREGISTER_RESERVE_POS_B	Le dernier registre correspond aux 2 octets supérieurs. Ex. M011
M020 & M021	MREGISTER_RESERVE_POS_T	
M030 & M031	MREGISTER_RESERVE_POS_X	Position cartésienne du manipulateur (FB) X,Y,Z [Unité: µm] Rx,Ry,Rz [Unité: 0.001 deg]
M032 & M033	MREGISTER_RESERVE_POS_Y	
M034 & M035	MREGISTER_RESERVE_POS_Z	
M036 & M037	MREGISTER_RESERVE_POS_Rx	Valeur codée sur 4 octets. Le premier registre correspond aux 2 octets inférieurs. Ex. M030
M038 & M039	MREGISTER_RESERVE_POS_Ry	
M040 & M041	MREGISTER_RESERVE_POS_Rz	Le dernier registre correspond aux 2 octets supérieurs. Ex. M031
M042 & M043	MREGISTER_RESERVE_POS_Re	Valeur du 7eme axe robot (lorsqu'il y en a un). [Unité: 0.001 deg]
M050 & M051	MREGISTER_RESERVE_VIT_S	Vitesse des axes Adresses définies par défaut. [Unité: 0.0001 deg/sec]
M052 & M053	MREGISTER_RESERVE_VIT_L	
M054 & M055	MREGISTER_RESERVE_VIT_U	
M056 & M057	MREGISTER_RESERVE_VIT_R	Valeur codée sur 4 octets. Le premier registre correspond aux 2 octets inférieurs. Ex. M050
M058 & M059	MREGISTER_RESERVE_VIT_B	
M060 & M061	MREGISTER_RESERVE_VIT_T	Le dernier registre correspond aux 2 octets supérieurs. Ex. M051
M070 & M071	MREGISTER_RESERVE_VIT_TCP	Vitesse de la TCP [Unité: µm/sec]

MREGISTER_RESERVE_

Valeurs codées sur 2 octets (Un registre).

*e = Estimé *max = Maximale

Définition	Nom	Description
M 310 à M 315	MREGISTER_RESERVE_TRQe_S* .. _L .. _U .. _R .. _B .. _T	Valeurs de couple externe estimées pour chaque axe
M 320 à M 322	MREGISTER_RESERVE_Fe_X .. _Y .. _Z	Valeurs de force externe estimées pour le TCP
M 323 à M 325	MREGISTER_RESERVE_TRQe_X .. _Y .. _Z	Valeurs de couple externe estimées pour le TCP
M 326	MREGISTER_RESERVE_Fe_Totale	Valeur force résultante (totale) estimée du TCP
M 330 à M 335	MREGISTER_RESERVE_SENSOR_CH1_S .. _L .. _U .. _R .. _B .. _T	Données de sortie du capteur pour chaque axe - Canal 1 (Unité non définie)
M 340 à M 345	MREGISTER_RESERVE_SENSOR_CH2_S .. _L .. _U .. _R .. _B .. _T	Données de sortie du capteur pour chaque axe - Canal 2 (Unité non définie)
M 350 à M 355	MREGISTER_RESERVE_TRQmax_S .. _L .. _U .. _R .. _B .. _T	Maximum de la valeur absolue des valeurs couples accumulées pour chaque axe
M 360 à M 362	MREGISTER_RESERVE_Fmax_X .. _Y .. _Z	Maximum de la valeur absolue des valeurs forces accumulées pour le TCP

Annexe 5. Page 5 sur 7

M 363 à M 365	MREGISTER_RESERVE_TRQmax_X .. _Y .. _Z	Maximum de la valeur absolue des valeurs couples accumulées pour le TCP
M 366	MREGISTER_RESERVE_Fmax_Totale	Maximum de la valeur force résultante (totale) du TCP

Each register value is expressed in units of 0.1 [N] (or [Nm]) and output by adding an offset of 10000.

For how to interpret the value, refer to the following examples.

Example:

Register	Value	External force (interpretation)	Calculation method
M310	9500	-50.0 [Nm]	$(9500 - 10000) \times 0.1 = -50.0[\text{Nm}]$
M321	11055	105.5[N]	$(11055 - 10000) \times 0.1 = 105.5[\text{N}]$

Annexe 5. Page 6 sur 7

STATUS_

- **81320 to 81397:**
 These are individual monitoring signals for the machine safety output (MS-OUT) of the safety logic circuit.

- **81400 to 81477:**
 These are individual monitoring signals for the functional safety output (FS-OUT) of the safety logic circuit

Définition	Nom	Description
80011	STATUS_REMOTE	Mode du robot
80012	STATUS_PLAY	Mode du robot
80013	STATUS_TEACH	Mode du robot
80015	STATUS_HOLD	
80016	STATUS_START	
80017	STATUS_SRV_ON	Touche "servo on ready"
80023	STATUS_SAFF	
80025	STATUS_STOP_URG	Bouton d'émergence du Contrôleur
80026	STATUS_STOP_TEACH	Bouton d'émergence du Teach
80027	STATUS PORTE	
80040	STATUS_VITESSE_2	
80041	STATUS_VITESSE_1	
80053	STATUS_SRV_PRET	
80054	STATUS_SRV_STAT	
80060	STATUS_CHOC_HOLD	

Annexe 5. Page 7 sur 7

80065	STATUS_CHOC_URG	
80400 à 80437	STATUS_GENERAL_I _O	Entrée/Sortie définie par l'utilisateur. Ref: RE-CKI-A467.5
81320 à 81397	STATUS_SFTY_MS	Rediriger les signaux de sécurité interne (MS-OUTXX) Ref: RE-CKI-A467.5
81400 à 81477	STATUS_SFTY_FS	

STATUS_SFTY_MS_

Définition	Nom	Description
81385	STATUS_SFTY_MS_LIDAR_RED	Signal associé à la zone de fonctionnement du robot (Zone rouge) Signal MS-OUT 54