AIP PRIMECA

Rapport de stage de fin d'étude

Intégration de robots industriels sous ROS et validation sur une tâche d'inspection industrielle

2 Mars 2020 – 29 septembre 2020

Nathan MORET

Spécialité automatique et systèmes embarqués
Promotion 2019-2020

En vue de l'obtention du titre d'ingénieur ENSISA

Tuteur pédagogique
Raphael DUPUIS

Tuteurs professionnels
Michel TAÏX
Thierry GERMA

UNIVERSITÉ HAUTE-ALSACE

ensisa

Rapport professionnel

Titre du rapport :

Intégration de robots industriels sous ROS validation sur une tâche d'inspection industrielle

Mots-clés :

Robotique industrielle, Intégration, Robot Operating System, Contrôleur, Driver, Démonstrateur

Résumé :

ROS, Robot Operating System, est un middleware permettant de développer des applications pour la supervision et le contrôle robotique. Son utilisation depuis sa création en 2007 par l'incubateur Willow Garage s'est largement répandue dans le milieu de la recherche grâce à sa licence open source mais reste encore peu utilisé en industrie. Mon stage au sein de la plateforme AIP-PRIMECA, Atelier Inter-établissements de productique et Pôle de Ressources Informatiques pour la MECAnique, s'inscrit dans ce contexte et a pour but de promouvoir, via le projet RIO, l'utilisation de ROS dans l'industrie de la robotique industrielle. Ma mission durant les six mois de stage a donc été d'intégrer le contrôle et la supervision d'un robot industriel de la marque Stäubli sous ROS au sein d'un démonstrateur d'une chaine de production industrielle. Ce mémoire présente le travail que j'ai réalisé.

Summary :

ROS, Robot Operating System, is a middleware used to develop applications of robotic supervision and control. Since its creation in 2007 by the company Willow Garage, it's used in the research community thanks to an open source license but is still little used in industry. My internship within AIP-PRIMECA, "Atelier Inter-établissements de productique et Pôle de Ressources Informatiques pour la MECAnique ", is integrated in this context and his goal is to promote the use of ROS in the robotics industry. My mission during this six-month internship was to integrate the control and the supervision of an industrial robot of the Stäubli brand with ROS in an industrial production line demonstrator. This thesis presents the work that I have done.

Approbation de diffusion

Je soussigné(e) TAÏX Michel.....
Tuteur professionnel de MORET Nathan.....
Dans l'entreprise AIP-PRIMECA.....

En tant que tuteur professionnel, j'atteste par la présente avoir pris connaissance du rapport de stage de fin d'études et donne mon accord pour son envoi au tuteur pédagogique.

Je soussigné(e) GERMA Thierry.....
Tuteur professionnel de MORET Nathan.....
Dans l'entreprise AIP-PRIMECA.....

En tant que tuteur professionnel, j'atteste par la présente avoir pris connaissance du rapport de stage de fin d'études et donne mon accord pour son envoi au tuteur pédagogique.

Formulaire d'information sur le plagiat

Dans le règlement des examens validé par la CFVU du 2 octobre 2014, le plagiat est assimilé à une fraude.

Le plagiat consiste à reproduire un texte, une partie d'un texte, des données ou des images, toute production (texte ou image), ou à paraphraser un texte sans indiquer la provenance ou l'auteur. Le plagiat enfreint les règles de la déontologie universitaire et il constitue une fraude. Le plagiat constitue également une atteinte au droit d'auteur et à la propriété intellectuelle, susceptible d'être assimilé à un délit de contrefaçon.

En cas de plagiat dans un devoir, dossier, mémoire ou thèse, l'étudiant sera présenté à la section disciplinaire de l'université qui pourra prononcer des sanctions allant de l'avertissement à l'exclusion.

Dans le cas où le plagiat est aussi caractérisé comme étant une contrefaçon, d'éventuelles poursuites judiciaires pourront s'ajouter à la procédure disciplinaire.

Je soussigné(e) MORET Nathan .....
Etudiant(e) à l'Université de Haute Alsace en : Ecole d'ingénieurs ENSISA .....
Niveau d'études : Bac +5 .....
Formation ou parcours : Automatique et systèmes embarqués .....

Reconnaît avoir pris connaissance du formulaire d'information sur le plagiat et atteste que ce document est exempt de toutes formes de plagiat.

Remerciements

Je souhaite remercier mes tuteurs professionnels, Michel TAÏX, enseignant-chercheur au LAAS-CNRS et Thierry GERMA, chef de projet Robotique spatiale et Vision chez Magellium, tous deux intervenants auprès de l'AIP-PRIMECA pour m'avoir accompagné et dirigé tout au long de mon travail sur ce projet. Ils ont su m'accorder tout le temps nécessaire afin de parfaire ma formation d'ingénieur.

Je remercie aussi toute l'équipe de l'AIP-PRIMECA composée de Cyril BRIAND, directeur de la plateforme, de Francine FUGIER, secrétaire et de Michèle GRIMAL, ingénieure informatique, pour leur accueil chaleureux dans la structure. Je remercie en particulier Fabien MARCO, ingénieur contrôle système, pour avoir suivi de près le projet et m'avoir fait prendre du recul sur mon travail et Olivier STASSE, enseignant-chercheur au LAAS-CNRS, pour m'avoir accordé du temps lorsque j'en avais besoin. Cette équipe bienveillante m'a permis d'évoluer dans les meilleures conditions possibles.

Je tiens également à remercier l'ENSISA pour l'enseignement que j'ai suivi et pour les compétences que j'ai acquises durant cette formation. Je remercie en particulier mon tuteur pédagogique, Raphaël DUPUIS ainsi que le responsable de la spécialité ASE, Jean-Philippe LAUFFENBURGER pour m'avoir communiqué l'offre de stage et permis de le réaliser.

Pour finir, je remercie mes parents de m'avoir soutenu et offert les meilleures conditions possibles à la réalisation de mon projet.

Glossaire

Abréviation	Signification
AIP-PRIMECA	Atelier Inter-établissements de productique et Pôle de Ressources Informatiques pour la MECAnique
INP	Institut National Polytechnique
INSA	Institut National des Sciences Appliquées
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes
CNRS	Centre National de la Recherche Scientifique
ESPE	Ecole Supérieur du Professorat et de l'Education
GIS	Groupement d'Intérêt Scientifique
S.mart	Systems Manufacturing Academics Resources Technologies
ROS	Robot Operating System
SRS	Stäubli Robotics Suite
RIO	Ros In Occitanie
BDS	Berkeley Software Distribution
OS	Operating System
LAN	Local Aera Network
TCP	Protocole de Contrôle de Transmissions
IP	Internet Protocol

Table des matières

Introduction

D'après l'Organisme Internationale de Normalisation, un robot industriel est un robot « manipulateur, multi-application, reprogrammable, commandé automatiquement, programmable sur trois axes ou plus, qui peut être fixé sur place ou mobile, destiné à être utilisé dans des applications d'automatisation industrielle » [1]. Ces caractères polyvalents et autonomes sont à l'origine de leur grande présence dans le paysage industriel actuel. Malgré son encrage important dans l'industrie, le domaine de la robotique industrielle continue d'évoluer afin de s'adapter au mieux à l'industrie du futur à l'image des robots collaboratifs qui permettent une intégration physique plus simple au sein d'une chaine de production. La robotique industrielle est devenue omniprésente dans l'industrie et continue son expansion.

Le marché des robots industriel est occupé par des grands constructeurs tel que Yaskawa, Kuka ou encore Stäubli qui proposent leur propre solution d'intégration basée sur un langage de programmation dédié à la marque (le langage Val3 pour Stäubli par exemple). Cela implique donc la connaissance de ces différents langages ainsi que l'acquisition des licences des suites robotiques pour l'utilisation de ces robots. Ces contraintes peuvent donc devenir un obstacle à l'automatisation des chaines de production des entreprises les plus modestes.

ROS peut alors présenter une solution à cette limitation. Possédant une licence open source, le contrôle et la supervision d'un robot industriel avec les outils de ROS peut se faire de manière plus générique et plus accessible.

Mon stage intervient à ce niveau et a pour objectif d'intégrer un bras robot Stäubli RX60 et un bras robot Kuka KR6 R700 sous ROS dans le but de démontrer les capacités et les performances de cette solution. Ce projet commence par l'élaboration du cahier des charges pour déterminer la stratégie à adopter et choisir les tâches industrielles de démonstration. Il se poursuit par la réalisation de la solution retenue. Pour finir la solution est testée sur le robot réel afin de déterminer ses performances et ses limites.

Les compétences utilisées et acquises lors de ce stage de fin d'étude de six mois seront présentées à la fin de ce mémoire.

Présentation de la plateforme

1. La structure

L'AIP-PRIMECA, Atelier Inter-établissements de productique et Pôle de Ressources Informatiques pour la MECAnique, est un espace dédié à l'enseignement et à la recherche en conception et fabrication mécanique.

Figure 1 - Logo de la plateforme [2]

Créée en 1995 et inaugurée en avril 1996 par un regroupement d'établissement toulousains composé de l'Université Paul Sabatier, l'INP Toulouse, l'INSA Toulouse et le LAAS-CNRS, la plateforme AIP-PRIMECA Occitanie s'inscrit dans une politique de mutualisation des ressources et des compétences afin de renforcer la formation pratique dans certains domaines nécessitant des moyens lourds et couteux, en phase avec la réalité industrielle. Les locaux de la plateforme se situe dans le hall technologique ESPE (Ecole Supérieur du Professorat et de l'Education) de l'université toulousaine Paul Sabatier.

Figure 2 - Hall technologique ESPE [3]

L'AIP-PRIMECA Occitanie fait maintenant partie du réseau national S.mart (Systems Manufacturing Academics Resources Technologies) qui est un GIS (Groupement d'Intérêt Scientifique) créé en 2015. Ce réseau permet la mutualisation des ressources en formation et en recherche dont l'objectif est de fédérer un ensemble d'actions pédagogiques, scientifiques et technologique entre les régions et le niveau national. Cette initiative a pour but de préparer au mieux la mutation industrielle engagée autour de l'industrie du futur.

Figure 3 - Logo du réseau S.mart [4]

Le réseau S.mart est composé depuis 2018 de dix pôles au niveau national dont le pôle Occitanie où se trouve l'AIP-PRIMECA Occitanie.

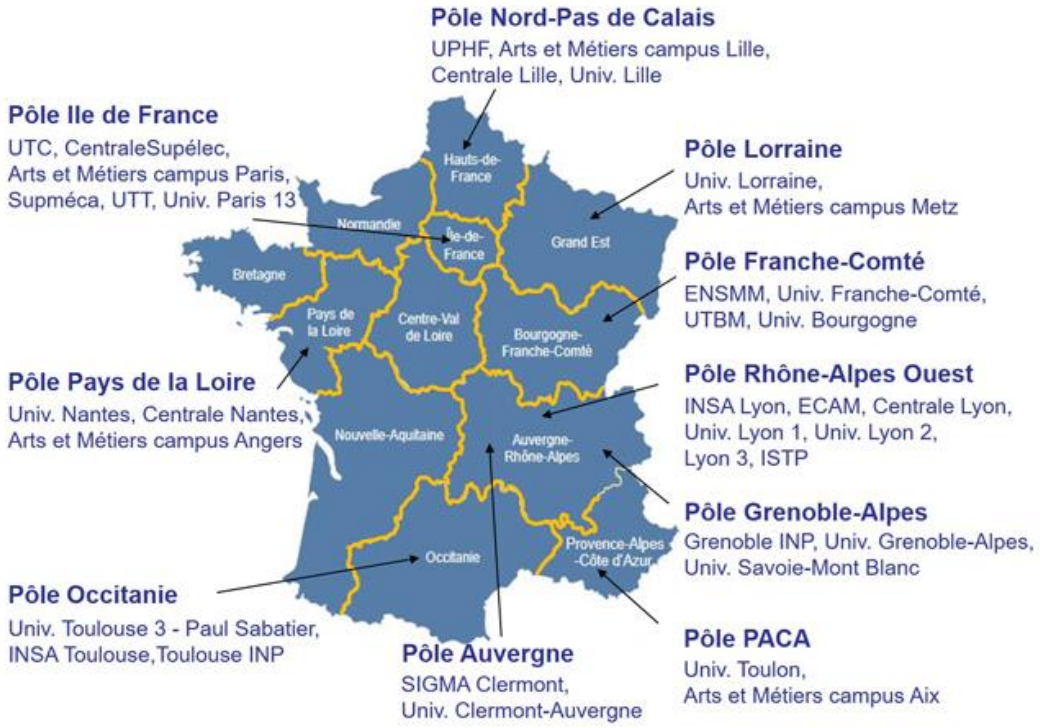


Figure 4 - Pôles du réseau S.mart [4]

Cette plateforme possède donc un parc informatique et un matériel industriel gérés par une équipe de quatre personnes, Cyril BRIAND à la direction, Francine FUGIER au secrétariat, Michèle GRIMAL et Fabien MARCO au pôle technique. Ces ressources sont mises à la disposition de projet pédagogique et de recherche.

2. Le matériel

Dans le cadre de la formation et de la recherche dans le domaine de l'industrie, la plateforme possède un prototype d'une chaîne de production industrielle.

2.1. La cellule flexible

La cellule flexible est un démonstrateur d'une chaîne de production industrielle composée de quatre modules de travail eux même composés d'un robot et d'un système de vision. Ces modules de travail sont connectés entre eux par un système de transport à navette.

Figure 5 - Photo de la cellule flexible

C'est système de transport Montrac commercialisé par la société Allemande Montratec. Il est composé d'un circuit de monorail (Annexe 1) sur lequel circule des navettes automotrices (Annexe 2). Ce système est commandé par des automates programmables industriels Modicon M340. Le but est de transporter des pièces manufacturées d'un poste de travail à l'autres pour leur faire subir différentes opérations industrielles.

Pour le moment seulement deux modules de travail ont été installés. L'un est composé d'un bras robot du constructeur Suisse Stäubli avec un système de vision, assuré par deux caméras de la marque Cognex, situé au-dessus des navettes automotrices afin de détecter et d'analyser leur cargaison (Annexe 3). L'autre est composé du même système de vision et d'un bras robot du constructeur Allemand Kuka.

2.2. Bras robot Stäubli RX60

Le bras robot Stäubli RX60 de la plateforme est un bras 6 axes d'une masse de 41 kg produit en 2005 (Annexe 4). Il n'est actuellement plus commercialisé par son constructeur. Son contrôleur est à l'origine un contrôleur CS7 qui a par la suite été mise à jour en contrôleur CS8. Le langage de programmation est le langage Val3 propre aux robots Stäubli.

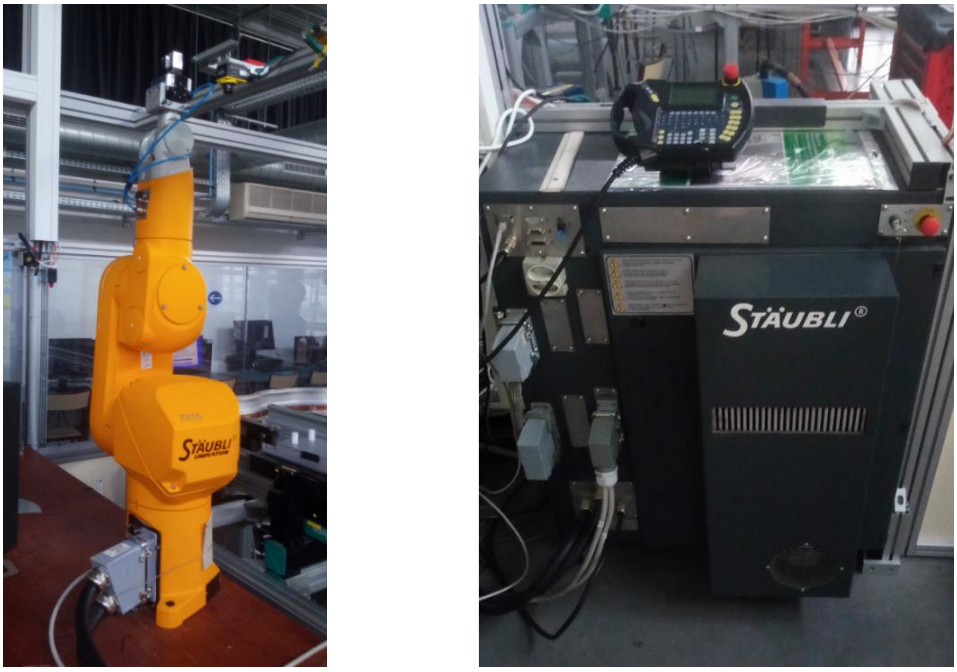


Figure 6 - Photos du bras robot RX60 et de son contrôleur CS8

Ces caractéristiques sont présentées dans les tableaux suivants :

Volume de travail Rayon de travail maxi entre axes 2 et 5 Rayon de travail mini entre axes 2 et 5 Rayon de travail entre axes 3 et 5	 600 mm 233 mm 310 mm
Vitesse maxi au centre de gravité de la charge Répétabilité à température constante	8 m/s +/- 0,02 mm

Tableau 1 - Caractéristiques générales du bras robot Stäubli RX60

Axe	1	2	3	4	5	6
Amplitude (°)	320	255	269	540	230	540
Vitesse nominale (° /s)	287	287	319	410	320	700
Résolution angulaire (°.10^{-3})	0,724	0,724	0,806	1,177	0,879	2,747

Tableau 2 - Amplitude, vitesse et résolution du bras robot Stäubli RX60

Charge transportable	Bras standard
A vitesse nominale	2,5 kg
A vitesse réduite	4,5 kg

Tableau 3 - Charge transportable du bras robot Stäubli RX60

2.3. Bras robot Kuka KR6 R700

Le bras robot Kuka KR6 R700 de la plateforme est un bras 6 axes d'une masse de 53 kg produit en 2018 (Annexe 5). Sont contrôleur est un contrôleur KR C4 compact. Le langage de programmation est le langage KRL propre aux robots Kuka.
Ces caractéristiques générales sont présentées ci-dessous :

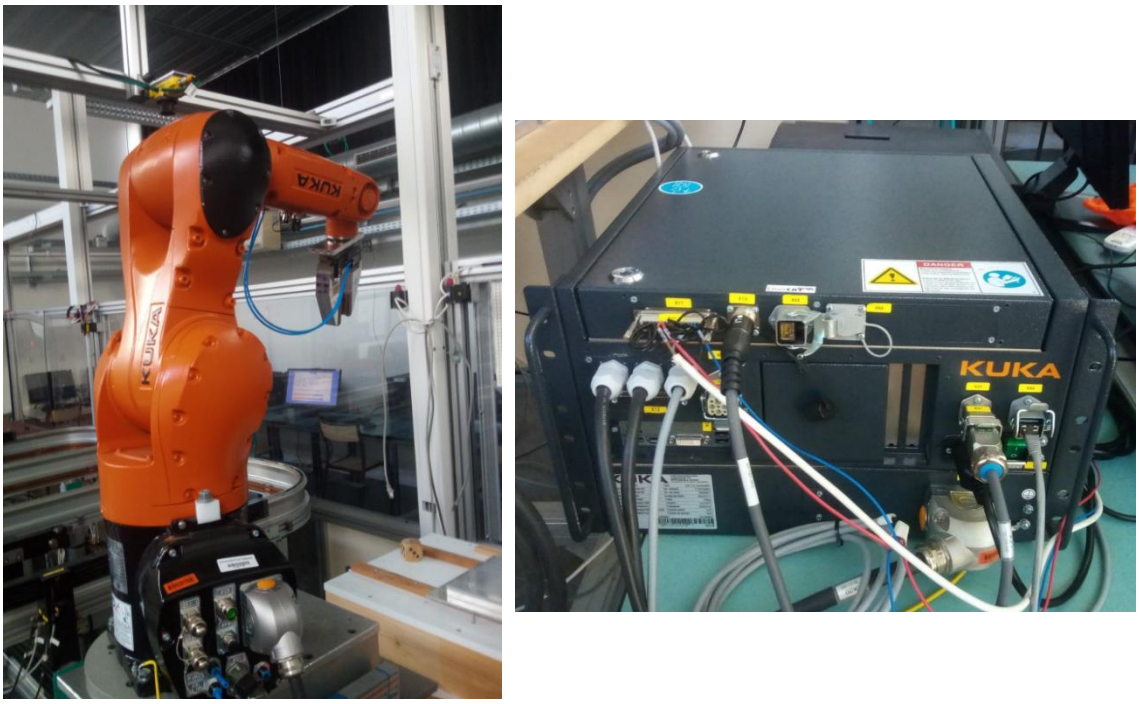


Figure 7 - Photos du bras robot Kuka KR6 R700 et de son contrôleur KR C4 compact

Portée maximum	726 mm
Charge maximum	6,8 kg
Répétabilité	+ /- 0,02 mm

Tableau 4 - Caractéristiques techniques du bras robot Kuka KR6 R700

Axe	1	2	3	4	5	6
Plage de mouvement	+/- 170°	-190° / 45°	-120° / 156°	+/- 185°	+/- 120°	+/- 350°

Tableau 5 - Caractéristiques des axes du bras robot Kuka KR6 R700

Contexte du stage

L'utilisation des équipements présentés précédemment nécessite la connaissance de différents langages de programmation dont la plupart sont exclusifs au constructeur. De plus le développement d'applications sur ce type de matériel passe par des logiciels propriétaire qui nécessite l'achat d'une licence pour être utilisés. C'est par exemple le cas du bras robot Stäubli RX60 qui nécessite la connaissance du langage Val3 propre au constructeur Stäubli et l'utilisation de la suite robotique Stäubli (SRS) Et du Kuka qui nécessite la connaissance du langage KRL et du logiciel Kuka Sim Pro.

1. Le projet RIO

Le projet RIO (ROS In. Occitanie) [5] est un projet qui a débuté en janvier 2020 au sein du projet européen ROS'in [6] et qui est hébergé par la plateforme AIP-PRIMECA Occitanie.

Figure 8 - Logo du projet RIO [7]

Toujours dans une vision centrée autour de l'industrie du futur, son but est de promouvoir l'utilisation de ROS dans le milieu de l'industrie en démontrant les performances d'une telle solution auprès des industriels et d'anticiper son développement par l'intermédiaire de la formation. Cette volonté est construite autour de trois axes.

Figure 9 - Les objectifs du projet RIO [7]

Son premier objectif est la création de programmes de formation de premier cycle, diplômés et postuniversitaires bien ciblés qui correspondent aux exigences de l'industrie et favoriseront l'utilisation de ROS dans l'industrie. Pour y parvenir, un autre objectif est de former de nouveaux instructeurs capables de mener les différentes sessions de formation. Le dernier objectif est d'intégrer et de contrôler les différents éléments de la cellule flexible (automates, convoyeurs, robots industriels, systèmes de vision, etc.) de la plateforme AIP-PRIMECA à l'aide de ROS.

Ce dernier objectif cherche donc à résoudre la problématique de la maîtrise des différents langages de programmation et de la dépendance aux solutions propriétaires des constructeurs pour le développement d'applications utilisant toutes les capacités de la cellule flexible. L'intérêt d'un tel objectif est donc de mettre en avant les capacités de supervision et de contrôle des systèmes robotiques de ROS à travers son caractère générique. Le but est d'utilisé la cellule flexible comme un démonstrateur de chaîne de production industrielle entièrement intégrée et contrôlée par ROS.

Ce projet est mené par une équipe venant de tous les structures impliquées. Elle est dirigée par Cyril BRIAND, directeur de la plateforme AIP-PRIMECA, et composée de :

- Michel TAÏX, enseignant-chercheur au LAAS-CNRS
- Thierry GERMA, chef de projet Robotique spatiale et Vision chez Magellium
- Fabien MARCO, ingénieur contrôle système à l'AIP-PRIMECA
- Olivier STASSE, enseignant-chercheur au LAAS-CNRS
- Bertrand VANDEPORTAELE, enseignant-chercheur au LAAS-CNRS

J'ai ainsi pris place dans cette équipe au cours de mon stage pour réaliser ma mission qui s'inscrit dans la réalisation du démonstrateur.

2. Robot Operating System

Dans l'optique de simplifier et de standardiser l'intégration de système robotique, ROS est développé depuis 2007 et est maintenant sous licence BDS (Berkeley Software Distribution License) qui est une licence open source. Ce type de licence permet la lecture, l'utilisation et la modification des codes sources ce qui permet à la communauté de ROS de développer et d'améliorer l'environnement ROS librement. Cette accessibilité est l'un des nombreux intérêts de ROS ce qui explique son utilisation croissante dans le domaine pédagogique et de la recherche.

Figure 10 - Logo de ROS [8]

ROS est un méta système d'exploitation, il se situe entre le système d'exploitation et le middleware. En effet, à l'image d'un système d'exploitation, il fournit des fonctionnalités de bas niveau telle que l'abstraction du matériel ou encore la gestion de la concurrence et des processus et à l'image d'un middleware, il crée un réseau d'échange d'informations entre différentes applications sans tenir compte des caractéristiques matérielles et logicielles des réseaux informatiques et des systèmes d'exploitation ordinateurs impliqués. Pour ROS, cet échange d'information se fait de la manière suivante :

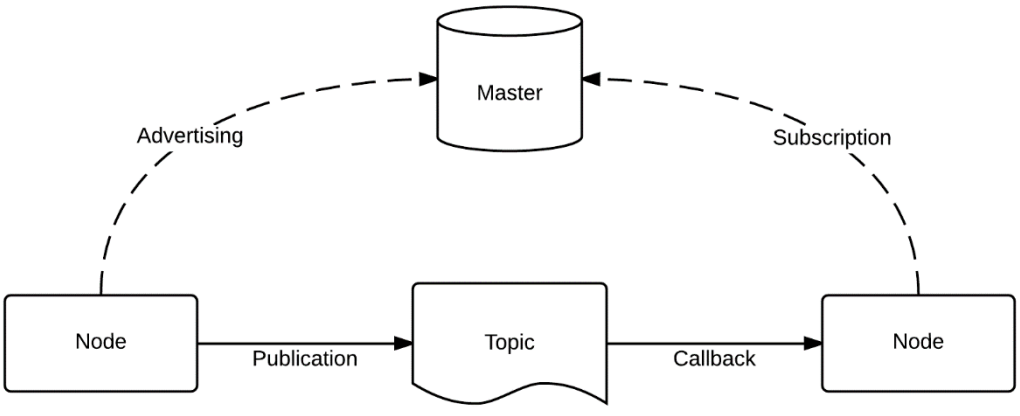


Figure 11 - Schéma du protocole de communication de ROS [9]

Les processus de ROS sont appelés « nœud ». Les nœuds communiquent entre eux avec des messages via des « topics ». Quand un nœud veut publier un

message sur un topic, il avertit dans une premier temps le master puis dans un second temps le publie, il est alors appelé « publisher ». Lorsque qu'un nœud veut consulter un message, il doit s'abonner au topic correspondant. Pour cela il avertit le master sa volonté de souscrire à un topic et le master crée la connexion. Ce nœud est alors appelé « suscriber ». Le suscriber a accès au message du topic via une fonction callback, c'est-à-dire une fonction qui n'est pas appelé par l'initiative de l'opérateur mais chaque fois qu'un message arrive, ROS appelle alors le gestionnaire de messages et lui transmettra le nouveau message. Un nœud peut être à la fois suscriber et publisher. ROS présente donc une structure de communication entre les processus de type publisher / suscriber permettant d'échanger des messages de type différent. Cette architecture lui confère une grande modularité et un caractère d'interopérabilité lui permettant d'interagir avec des systèmes robotiques différents.

Un des intérêts de ROS est qu'il possède un certain nombre d'outils comme par exemple :

- OpenCV pour le traitement du signal
- OctoMap pour la représentation de l'environnement
- SMACH pour la planification de tâches
- MoveIt pour la planification de mouvement
- ROS control pour le contrôle

Pour finir ROS est disponible sous différentes distributions qui sont créées en fonction de ses mises à jour et de son évolution. A mon arrivée sur le site, le parc informatique étant sous Ubuntu 16.04LTS, mon travail a été réalisé sous la distribution de ROS Kinetic. Durant la fin de mon stage, le parc informatique ayant été passé sur Ubuntu 18.04LTS, j'ai réalisé un portage de mon travail avec succès sur la distribution ROS Melodic.

3. Organisation de la mission

La mission de mon stage consiste à intégrer les deux robots industriels de la plateforme sous ROS pour les superviser et les contrôler dans le but de réaliser des démonstrations de tâche industrielle. Cette partie du projet a débuté avec le début de mon stage et m'a été entièrement confié. Partant de zéro, l'une des premières étapes de mon travail a été d'établir le cahier des charges de cette aspect du projet avec le reste de l'équipe à partir des attentes du démonstrateur pour choisir les tâches industrielles visées et d'une recherche bibliographique pour déterminer l'architecture de la solution. C'est donc à ce moment que le planning de la mission a été défini.

3.1. Planning prévisionnel

Ma mission est composée de trois phases : l'initialisation du projet, le développement de la solution et la mise en pratique de la solution réalisée.

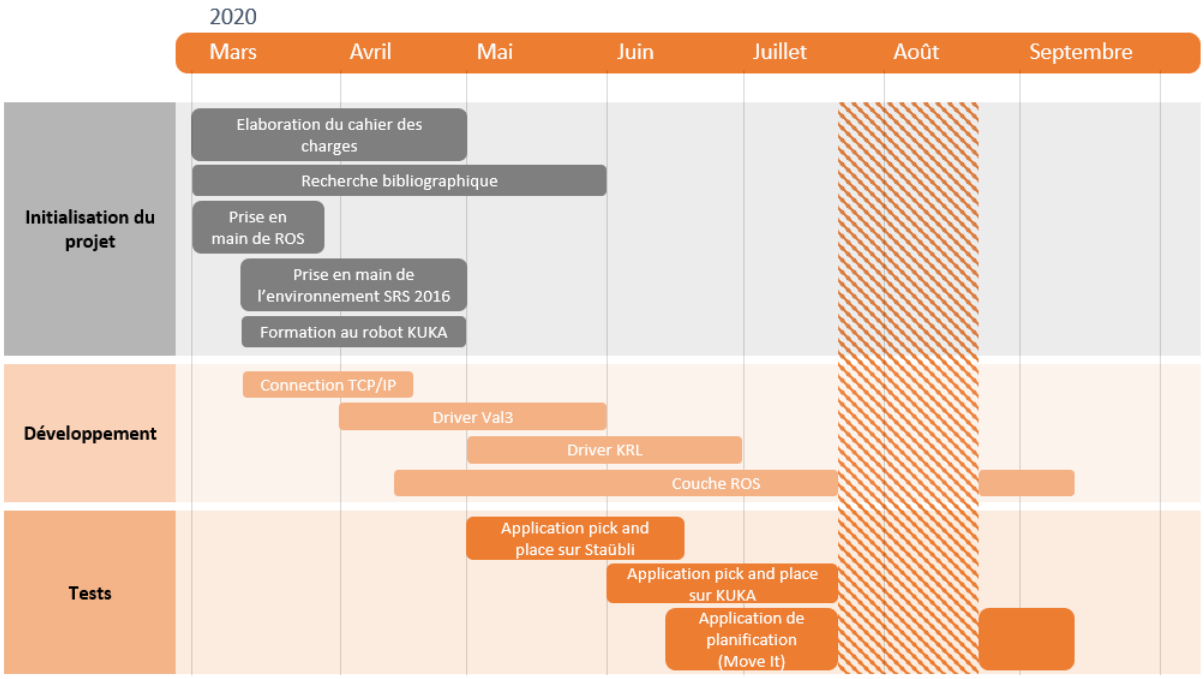


Figure 12 - Diagramme de Gantt prévisionnel

- **Initialisation du projet :**
  Cette phase consiste à établir une base solide de travail afin d'assurer au mieux le développement de la solution. Les attentes sont définies via la réalisation du cahier des charges. Les outils et le matériel sont pris en main par la révision des connaissances déjà acquise sur ROS et l'utilisation du robot Stäubli et de son environnement de développement (ici SRS 2016) lors de ma formation et par une formation à l'utilisation du robot Kuka et de son environnement dont je n'ai aucune expérience.

- **Développement de la solution :**
  La plateforme fermant ces portes du 25 juillet 2020 au 24 août 2020, le planning a été fixé de façon à pouvoir présenter une démonstration d'une tâche industrielle simple sur les deux bras robots avant la fermeture. Cette phase consiste donc à programmer la solution à partir du travail réalisé dans la phase précédente étape par étape. Une fois la communication établie entre ROS et le contrôleur, le choix est fait de concentrer mon travail dans un premier temps sur le robot Stäubli avec lequel je possède déjà une expérience pour ensuite intégrer la robot Kuka à la solution.

- **Le test de la solution :**
  Cette dernière phase a pour but de valider sur les robots réels la solution réalisée durant la phase précédente sur les différentes tâches industrielles définies dans le cahier des charges. C'est à ce moment que, en fonction des résultats obtenus, la solution est modifiée et adaptée pour fixer les disfonctionnements éventuels.

L'organisation de travail s'est organisée autour d'un travail en autonomie échelonné par une réunion hebdomadaire avec mes tuteurs professionnels pour faire le point sur l'avancement du travail réalisé et d'une réunion mensuelle avec toute l'équipe du projet RIO pour synthétiser l'avancement du projet et les résultats obtenus sur l'ensemble du projet RIO et ainsi pouvoir adapter les objectifs pour la suite du projet.

3.2. Planning effectif

Comme dans de nombreux projets, le planning prévisionnel n'est pas toujours respecté dû à des problèmes imprévus ou à des circonstances particulières. Dans le cas de ce projet, le contexte sanitaire sans précédent de la première moitié de l'année 2020 nous a forcé à adapter le planning.

Mon stage a été effectué en télétravail sur la période du 16 mars 2020 au 25 juillet 2020 suite aux mesures sanitaires fixées au niveau national. J'ai donc dû emporter dans la précipitation une partie du matériel afin de poursuivre mon travail dans ces conditions. Seul le matériel consacré au robot Stäubli RX60 (ordinateur du simulateur du robot et jeton de la licence SRS 2016) a pu être déplacé sur le nouveau site de travail. La structure générale du planning de mon travail n'a pas été modifié cependant toutes les tâches d'intégration du robot Kuka KR6 R700 ont dû être supprimées. De plus la phase de test a aussi dû être adaptée par l'impossibilité d'avoir accès aux robots réels. Elle s'est donc déroulée sous simulation durant la période de télétravail et a été réalisée sur le robot Stäubli à partir du 24 août 2020, date de mon retour sur site.

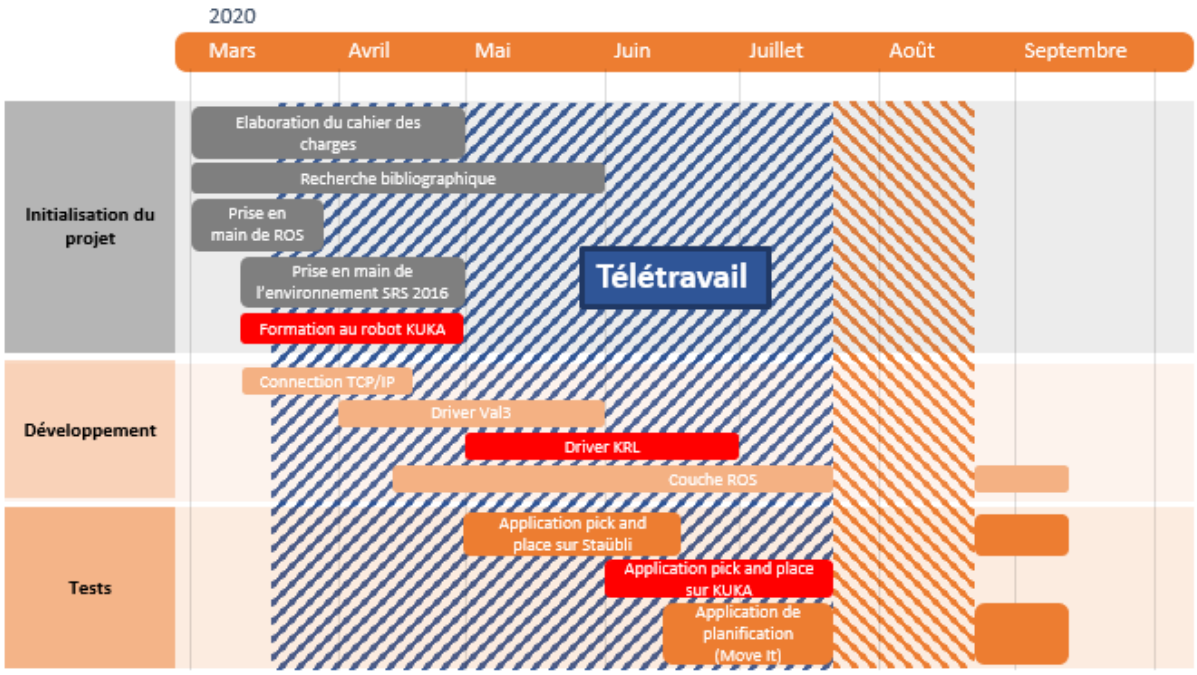


Figure 13 - Planning effectif de la mission

Les réunions se sont donc déroulées en visioconférence et leur fréquence n'a pas été modifiée.

En résumé, pour s'adapter au contexte, l'objectif de mon stage a été réduit à l'intégration de la solution uniquement sur le bras robot Stäubli RX60 en reportant l'intégration du bras robot Kuka KR6 R700 sur une autre phase du projet RIO.

Elaboration du cahier des charges

Cette partie du projet débutant de zéro, l'élaboration du cahier des charges est une étape importante qui permet de fixer les objectifs et la stratégie de la solution. Se basant sur une recherche bibliographique approfondie, le cahier des charges nous permet de suivre un fil conducteur cohérant dans le travail à fournir.

1. Tâches industrielles visées

L'objectif de mon travail est de réaliser un démonstrateur de ROS à partir des robots industriels présent sur le site pour démontrer l'intérêt d'une telle solution auprès des acteurs de l'industrie. Les tâches de démonstrations doivent donc être choisies de manière à concorder avec la réalité du monde de l'industrie. Ces tâches doivent aussi pouvoir s'intégrer dans l'environnement du robot, c'est-à-dire la cellule flexible. Il est donc nécessaire de faire interagir le robot avec les autres systèmes de la cellule comme le système de transport ou encore le système de vision composant les postes de travail. Leur complexité est aussi un critère important dans leur détermination. Les tâches choisies sont les suivantes :

- **Une tâche de pick and place :**
  Le pick and place est une tâche simple et très utilisée dans l'industrie qui consiste à prendre un objet à un endroit pour le déplacer à un autre endroit ou à changer son orientation. Cette tâche est facile à mettre en œuvre et permet d'utiliser différent type de mouvement du robot comme un mouvement linéaire, circulaire ou encore d'approche. De plus cette tâche permet d'interagir avec les navettes de la cellule flexible pour opérer sur les objets qu'elles transportent.

- **Une tâche de suivi de contour de frome :**
  Cette tâche consiste à faire suivre les contours d'un objet par l'outil du robot. Dans l'industrie cette application peut être utilisée pour faire un dépôt de colle ou encore une soudure sur une pièce. L'intérêt de cette tâche et de faire interagir le système de vison du poste de travail avec le bras robot. L'idée est de faire passer l'outil du robot à des coordonnées correspondants au contour d'un objet obtenue suite à un traitement d'image par le système de vision. La partie vision du projet RIO n'étant pas encore commencée, le but est de montrer la capacité de la solution de suivre une trajectoire donnée.

- **Une tâche d'évitement d'obstacle :**
  L'intérêt de cette tâche est d'inclure dans la solution l'outil MoveIt de ROS qui est un planificateur de mouvement. Dans une chaîne de production industrielle, l'espace de travail d'un robot peut contenir des obstacles dont le robot ne tiens pas compte. Le but est donc d'intégrer l'environnement au fonctionnement du robot.

2. Architecture de la solution

Lorsqu'on programme un robot industriel avec la solution du constructeur, on code un programme dans un langage propre au constructeur sur une suite robotique pour le transférer sur le contrôleur. Une fois le programme téléchargé sur le contrôle, il communique à l'OS (système d'exploitation) du contrôleur les instructions qui sont alors effectuées par le robot.

Pour contrôler le robot avec ROS, la première piste explorée est donc de remplacer le programme en langage propre au constructeur par une couche ROS contenant le programme à réaliser par le robot. Cette solution impose donc de faire communiquer la couche ROS directement avec l'OS du contrôleur du robot. C'est une solution très bas niveau qui nécessite de connaître l'architecture hardware du contrôleur. Devant la charge de travail conséquente à l'élaboration d'une telle solution en comparaison avec la durée de mon stage, cette piste a vite été abandonnée ce qui nous a poussé à considérer le contrôleur du robot comme une boîte noire et atténuant l'intérêt d'utiliser certains outils qu'offre ROS comme ROS control.

La baie du contrôleur disposant de ports Ethernet et USB, l'architecture finalement retenue se base sur un programme en langage propre au robot permettant de communiquer entre l'OS du contrôleur et la couche ROS, c'est-à-dire un driver. Ce driver a pour rôle de réceptionner, d'interpréter et de transférer les informations entre l'OS du contrôleur et la couche ROS.

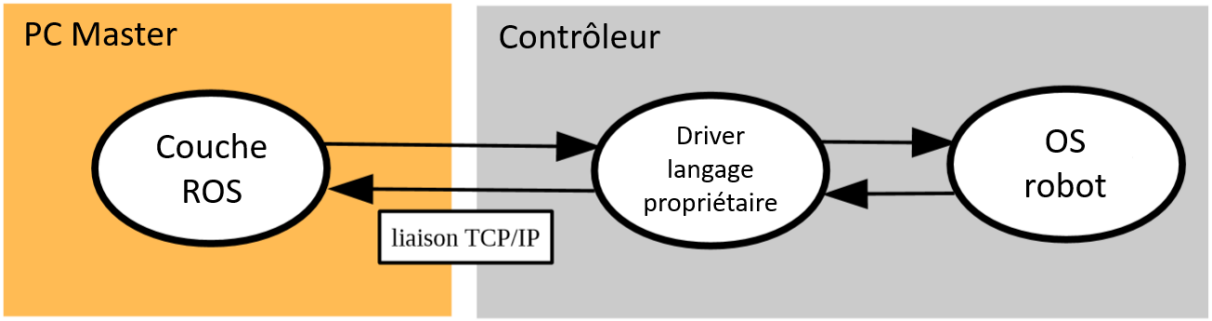


Figure 14 - Schéma de l'architecture de la solution

Cette architecture fait intervenir la couche ROS comme une surcouche de contrôle par l'intermédiaire d'un programme de contrôle en langage du robot classique détourné en driver. Cette solution impose donc le développement d'un driver différent pour chaque contrôleur pouvant être transfert par le biais du port USB ou Ethernet. Une fois la driver crée, cette architecture permet de ne plus dépendre des solutions d'intégration propriétaire.

3. Communication entre ROS et le contrôleur

Trois couches de programme doivent communiquer ensemble, la couche ROS, la couche driver et la couche OS du contrôleur. Le driver étant codé comme un programme de contrôle classique avec la solution du constructeur, la communication entre le driver et l'OS du contrôleur n'a pas besoin d'être traitée et est inclus dans la boîte noire du contrôleur. Il reste donc à définir la communication entre le driver et la couche ROS.

Les éléments de la cellule flexible de la plateforme AIP-PRIMECA Occitanie sont connectés à un parc d'ordinateur par un réseau LAN (réseau local) filaire Ethernet comme le montre la figure ci-dessous :

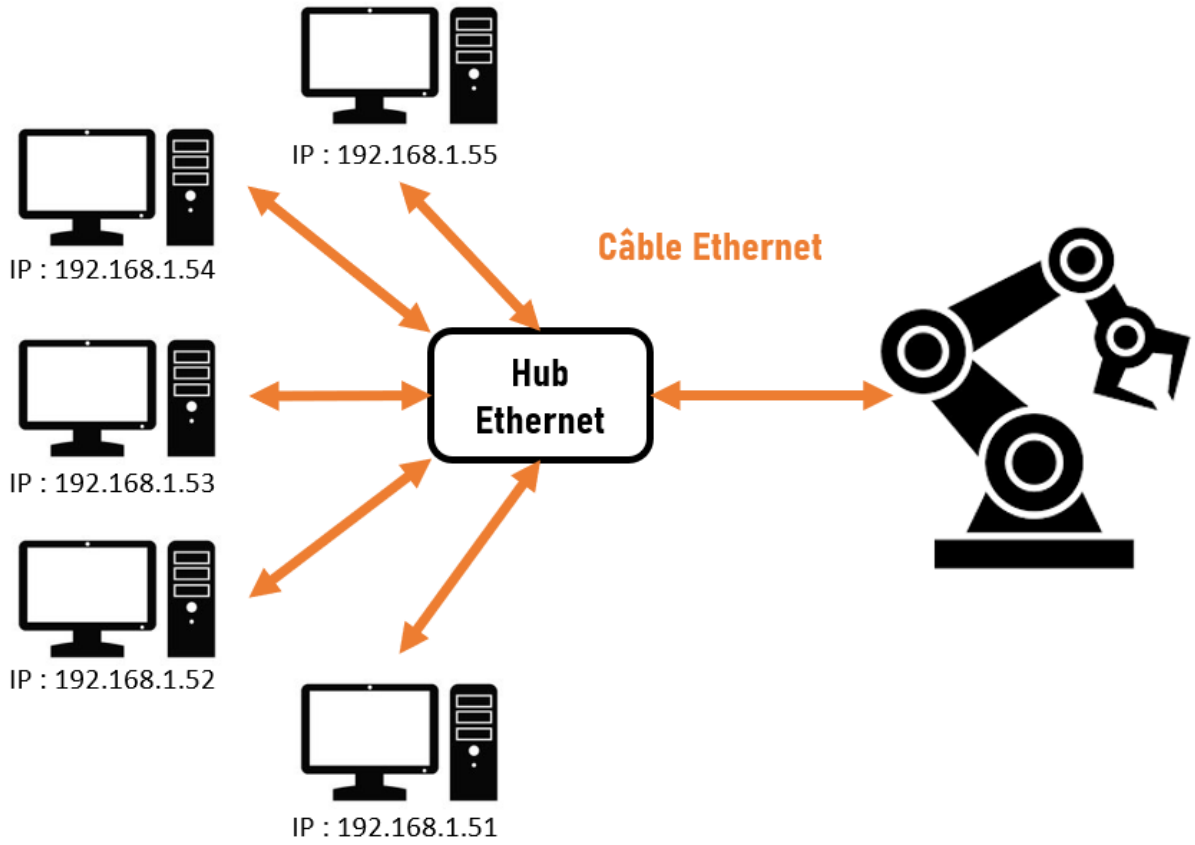


Figure 15 - Réseau de communication

Il est nécessaire de commencer par choisir un modèle de communication. Par soucis de standardisation et de simplicité, le choix s'est vite tourné vers le modèle TCP/IP. Le modèle TCP/IP est une norme de communication basée sur une architecture en quatre couches :

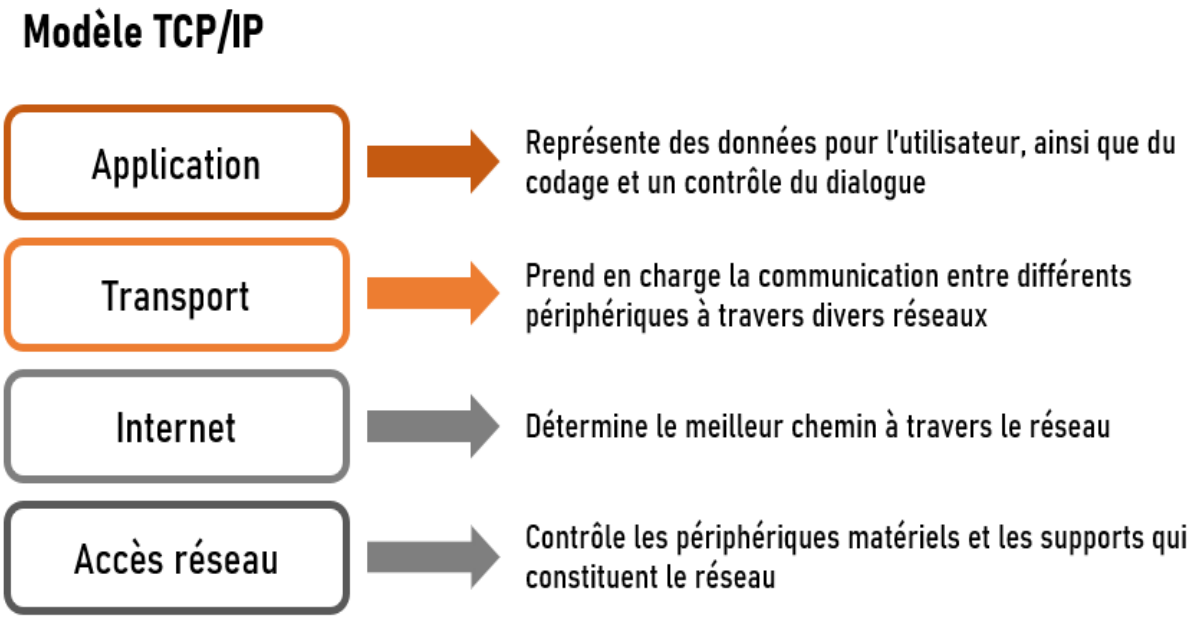


Figure 16 - Couches du modèle TCP/IP

Le nom de ce modèle provient du nom ses deux principaux protocoles, le protocole TCP (Transmission Control Protocol) pour la couche de transport et le protocole IP (Internet Protocol) pour la couche internet. Leur fonctionnement est le suivant :

- **Protocole TCP :**
  TCP à un premier rôle d'initialisation et de fin de communication entre deux machines par le biais de message de synchronisation et d'acquittement. Ensuite le rôle de ce protocole et de fragmenter en segment les messages de la couche application pour les transmettre à la couche internet et inversement de inversement de reconstruire en message les segments issues de la couche internet pour les transmettre à la couche application. Pour finir TCP gère le flux de données par le biais de buffer (tampon) de taille finie.

- **Protocole IP :**
  Ce protocole encapsule les données que l'on appelle alors datagramme. Le principe de cet encapsulage est d'ajouter un en-tête aux données permettant leur transport notamment à partir de l'adresse IP des machines.

Ce modèle est basé sur la communication d'un serveur et de clients. La place du serveur doit donc être déterminer. Ici le critère de simplifier l'utilisation de la solution est déterminant. Le serveur doit ainsi être intégré dans la couche ROS pour ne pas que le fonctionnement des processus de cette dernière ne soit affecté lorsque le robot n'a plus la possibilité de communiquer. L'architecture de communication utilisée entre la couche ROS et le contrôleur du robot est alors sous la forme suivante :

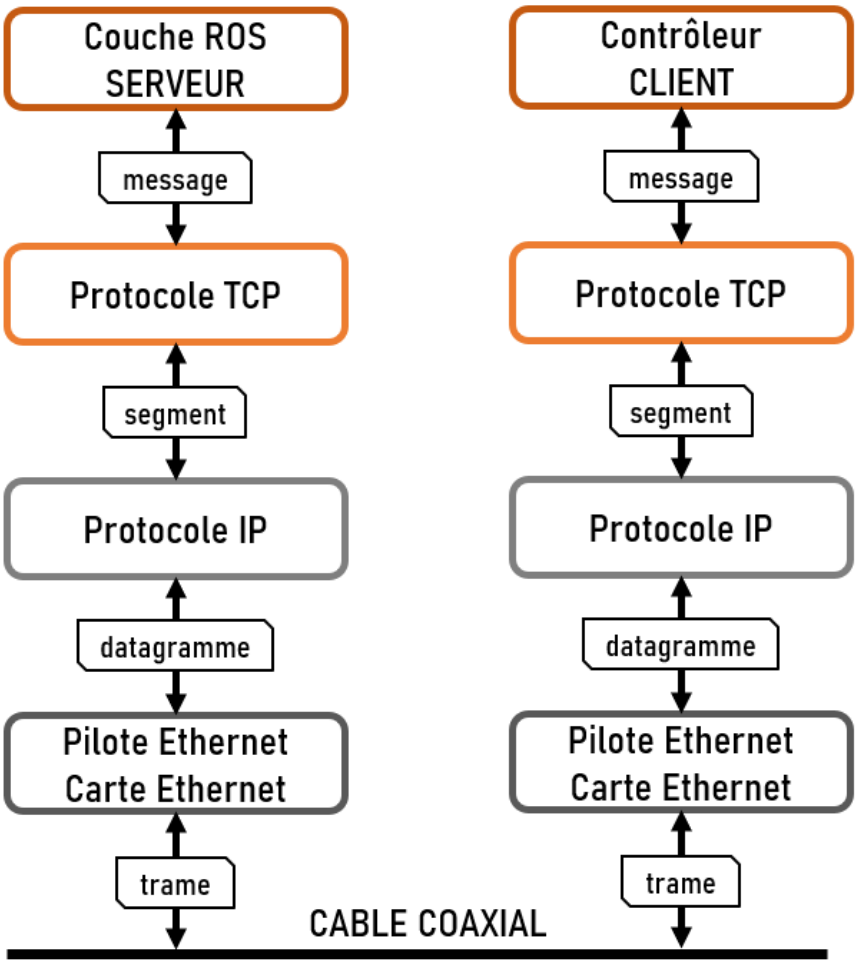


Figure 17 - Architecture de communication de la solution

Le protocole de communication des couches applications devra alors être réalisé.

Intégration de la solution

En se basant sur le cahier des charges préalablement déterminé, la solution doit être développée selon trois axes. Le premier et de créer un protocole de communication entre les deux couches applications du modèle TCP/IP respectivement défini par la couche ROS et le driver du contrôleur. Le deuxième axe est l'élaboration du driver pour qu'il remplisse sa fonction d'interpréteur entre la couche ROS et l'OS du robot. Pour finir le dernier axe est la programmation de la couche de contrôle et de supervision du robot sous ROS.

1. Protocole de communication

Les deux couches applicatives communiquent entre elles via des messages de différents types contenant plusieurs données. Il est donc nécessaire d'identifier les différents types de message, de marquer la fin d'un message et de marquer la fin des différentes données qui composent un message. La structure des messages utilisés pour la communication est la suivante :

Id X donnée X donnée X … donnée X F

Avec :

- Id : l'identifiant du message décrivant son contenu
- X : le marqueur de fin de valeur
- F : le marqueur de fin de message

Les messages utilisés ont été déterminé par l'étude des informations que peut recevoir et communiquer le contrôleur du robot. Ils peuvent ainsi être regroupés en six types différent :

- **Message de mouvement :** contient la consigne pour les différents types de mouvement (point à point, linéaire et circulaire)
- **Message de paramètres de mouvement :** contient les paramètres de vitesse et de lissage
- **Message dédié à l'outil :** contient la consigne de contrôle de l'outil ainsi que ces paramètres
- **Message de configuration du robot :** contient la consigne de configuration (Annexe 6) et de la vitesse moniteur
- **Message d'état du robot :** contient les informations de l'état du robot réel
- **Message stop :** averti la fermeture d'une application

Les informations de ces différents messages sont détaillées dans les tableaux suivants :

	ID	Description
Mouvement	11	Coordonnée articulaire pour mouvement point à point
	121	Coordonnée cartésienne pour mouvement point à point
	122	Coordonnée cartésienne pour mouvement linéaire
	1231	Coordonnée cartésienne du point d'arrivée d'un mouvement circulaire
	1232	Coordonnée cartésienne du point intermédiaire d'un mouvement circulaire
Paramètres mouvement	2	Consigne de vitesse et de lissage du mouvement
Outil	31	Consigne des paramètres de l'outil
	32	Commande de l'action de l'outil
Configuration robot	41	Consigne de la configuration des articulations du robot
	42	Consigne de la vitesse moniteur du robot
Etat	5	Retour de l'état du robot
Stop	99	Consigne pour vider la pile de consigne du contrôleur

Tableau 6 - Identifiants des différents messages

ID	Message
11	11 X j1 X j2 X j3 X j4 X j5 X j6 X F
121	121 X x X y X z X rx X ry X rz X F
122	122 X x X y X z X rx X ry X rz X F
1231	1231 X x X y X z X rx X ry X rz X F
1232	1232 X x X y X z X rx X ry X rz X F
2	2 X accel X vel X decel X tvel X rvel X blend X leave X reach X F
31	31 X x X y X z X rx X ry X rz X otime X ctime X F
32	32 X action X F
41	41 X shoulder X elbow X wrist X F
42	42 X speed X F
5	5 X isPowered X isCalibrated X workingMode X esStatus X speed X shoulder X elbow X wrist X isSettled X F
99	99 X F

Tableau 7 - Architectures des différents messages

Donnée	Définition	Unité ou valeurs
j1 / j2 / j3 / j4 / j5 / j6	Position Joint axes	degré
x / y / z	Translation sur axes	millimètre
rx / ry / rz	Rotation autour axes	degré
accel	Accélération articulaire maximale autorisée	% de l'accélération nominale du robot
vel	Vitesse articulaire maximale autorisée	% de la vitesse nominale du robot
decel	Décélération articulaire maximale autorisée	% de la décélération nominale du robot
tvel	Vitesse maximale autorisée de translation du centre outil	millimètre par seconde
rvel	Vitesse maximale autorisée de rotation de l'outil	degré par seconde
blend	Mode de lissage	0 → pas de lissage 1 → lissage articulaire 2 → lissage cartésien
leave	Distance où commence le lissage	millimètre
reach	Distance où finit le lissage en	millimètre
otime	Délai d'ouverture de l'outil	seconde
ctime	Délai de fermeture de l'outil	seconde
action	Contrôle de l'outil	0 → ouverture de l'outil 1 → fermeture de l'outil 2 → attente de fin mouvement de l'outil
shoulder	Configuration de l'épaule	0 → configuration de l'épaule droite imposée 1 → configuration de l'épaule gauche imposée 2 → changement configuration épaule interdit 3 → configuration de l'épaule libre
elbow	Configuration du coude	0 → configuration du coude positive imposée 1 → configuration du coude négative imposée 2 → changement configuration coude interdit 3 → configuration du coude libre

wrist	Configuration du poignet	0 →configuration du poignet positive imposée 1 →configuration du poignet négative imposée 2 →changement configuration poignet interdit 3 →configuration du poignet libre
speed	Vitesse moniteur	% de la vitesse nominale du robot
isPowered	État d'alimentation du bras	0 → hors tension 1 → sous puissance
isCalibrated	État du calibrage du robot	0 → au moins un axe du robot n'est pas calibré 1 → tous les axes du robot sont calibré
workingMode	Mode de fonctionnement courant du robot	0 → invalide ou en transition 1 → manuel 2 → test 3 → local 4 → déporté
esStatus	État du circuit d'arrêt d'urgence	0 → pas d'arrêt d'urgence 1 → fin arrêt urgence, attente de validation 2 → arrêt d'urgence ouverture
isSettled	État du mouvement du robot	0 → position du robot non stabilisée 1 → le robot est arrêté

Tableau 8 - Données contenues dans les différents messages

Le protocole de message étant défini, il fixe les bases du développement du driver du contrôleur et de la couche ROS.

2. Driver Val3

Le driver Val3 est le driver dédié à l'utilisation du robot Stäubli RX60 piloté par le contrôleur CS8 dont le langage de programmation est le Val3 qui est un langage de haut niveau. Ce langage est propre à la solution propriétaire proposée par le constructeur Stäubli. Il repose sur des applications composées de fonctions de bases propre à ce langage, de programmes qui sont une séquence d'instructions VAL3 à exécuter, de variables et de tâches. Une tâche est un programme en cours d'exécution qui peut fonctionner en parallèle d'autres tâches de manière synchrone ou asynchrone [10]. Dans le cas du projet, l'application Val3 crée (le driver Val3) est une application dont les tâches sont dédiées à la communication avec la couche ROS en utilisant le protocole de message précédemment présenté. L'architecture du driver est donc la suivante :

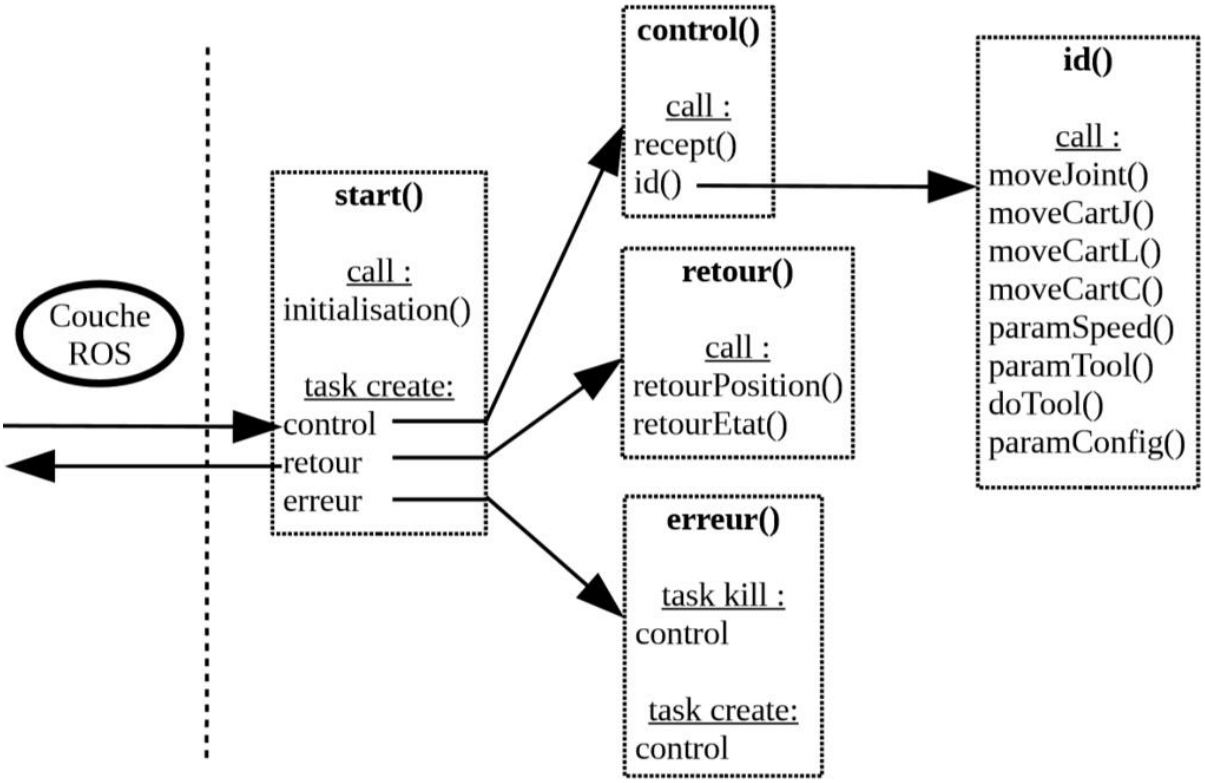


Figure 18 - Architecture du driver Val3

Cette application est composée de trois tâches, control, retour et erreur respectivement composées d'un programme du même nom. Ces tâches sont asynchrones et n'ont pas la même priorité. La priorité est argument d'une tâche correspondant au nombre de ligne exécutée de la tâche avant de passer à une autre tâche dans le cas d'instructions non bloquantes.

- **Tâche control :** Réceptionne les messages envoyés par la couche ROS et les traite
  → Priorité = 20

- **Tâche retour :** Lis les informations du robot réel et les envois à la couche ROS
  → Priorité = 50
- **Tâche erreur :** Traite les erreurs bloquantes mineures pour éviter une intervention de l'opérateur
  → Priorité = 1

La tâche retour possède la priorité la plus grande car le retour de l'état du robot est primordial à la couche ROS pour pouvoir adapter son contrôle sur le robot. Inversement la tâche erreur possède une priorité très faible car elle n'intervient que lorsque qu'une instruction est bloquante, il est donc inutile de l'appeler volontairement.

Ces tâches s'appuient sur différents programmes et différentes variables présentées ci-dessous :

Type	Définition du type	Variables	Définition de la variable
sio	Entrées-sorties sur liaison série et socket Ethernet	siCo	Entrée-sortie socket Ethernet pour la communication avec la couche ROS
dio	Entrées-sorties numériques	diOutil	Entrée-sortie pour le contrôle de l'outil
config	Configurations du robot	cConfig	
tool	Outils montés sur un robot	tTool	
mdesc	Paramètres de déplacement du robot	mSpeed	
Joint	Positions des axes du robot	jJoint	Consigne position articulaire
		jRetour	Mesure position articulaire réel
point	Positions cartésiennes d'un outil	pCart	Consigne position cartésienne
		pInterm	Positon cortésienne du point intermédiaire pour mouvement circulaire
		pRetour	Mesure position cartésienne réel

Tableau 9 - Variables du driver Val3

Le but est de rentre l'application de le modulaire possible afin qu'elle puisse traiter la majorité des cas de contrôle. L'intérêt de se limiter au maximum dans le nombre de variables crées et de ne pas chercher à créer une variable pour chaque cas de figure (qui serait beaucoup trop long) mais de mettre à jour le plus souvent possible une même variable traitant la majorité des cas.

Nom	Définition	Variable(s) utilisée(s)	Id message(s) utilisé(s)
start	Appelle dans un premier temps le programme initialisation et dans un second temps lance les 3 tâches parallèles control, retour et erreur		
initialisation	Test l'écriture et la lecture de message sur la liaison de communication avec la couche ROS puis créé cette liaison	siCo	
control	Appelle successivement les programmes recept et id dans une boucle infinie		
recept	Traduit les données reçues en format ASCII en message de format string	siCo	
id	Répartit les messages transmis par le programme recept() dans les différents programmes de contrôle en fonction de l'identifiant		42 99
moveJoint	Commande un mouvement articulaire à partir d'une position articulaire reçues	jJoint tTool mSpeed	11
moveCartJ	Commande un mouvement articulaire à partir d'une position cartésienne reçues	pCart tTool mSpeed	121
moveCartL	Commande un mouvement linéaire à partir d'une position cartésienne reçues	pCart tTool mSpeed	122
moveCartC	Commande un mouvement circulaire à partir d'une position cartésienne reçues et d'une position cartésienne intermédiaire	pCart pInterm tTool mSpeed	1231 1232
paramSpeed	Actualise les paramètres de mouvement	mSpeed	2
paramTool	Actualise les paramètres de l'outil	tTool	31
doTool	Commande l'outil à partir des données reçues ou ordonne l'attente de fin de mouvement de l'outil	diOutil	32
paramConfig	Actualise les paramètres de configuration de l'épaule, du coude et du poignet	cConfig	41

retour	Appelle successivement les programmes retourPosition et retourEtat dans une boucle infinie		
retourPosition	Mesure la position cartésienne de l'outil et articulaire du robot, créé en format string et envoie le message contenant ces informations à la couche ROS	pRetour jRetour siCO	11 121
retourEtat	Lis l'état du robot, crée en format string et envoie le message à la couche ROS	siCO	5
erreur	Tue et recréé la tâche control quand l'erreur de position inaccessible pour le robot apparaît		

Tableau 10 - Données contenues dans les différents messages

Dans la tâche control le driver ne traite qu'un message à la fois, c'est-à-dire que lors de la réception d'un nouveau message, il le traduit, l'interprète et réceptionne le message suivant que lorsque l'instruction a été envoyé à l'OS du contrôleur. Le robot mettant un temps non négligeable pour réaliser une action en comparaison avec le temps de traitement d'un message, les instructions d'actions s'accumule dans une pile de consigne de l'OS permettant au contrôleur de faire exécuter les consignes au robot tout en tenant compte des consignes futures.

3. Couche ROS

C'est au niveau de la couche ROS que le contrôle et la supervision du robot industriel sont réalisés. Cette couche doit donc avoir un caractère générique pour pouvoir être utilisable sans modification pour différents robots. Son rôle est de communiquer avec le driver préalablement programmé du robot visé par l'intermédiaire d'un serveur qu'elle héberge, récupérer les informations du robot et le contrôler en fonction de ces informations. L'architecture de cette couche s'appuis ainsi sur trois nœuds, un premier nœud dédié au serveur, un autre nœud servant à décrire l'état du robot réel et un dernier nœud utilisé pour réaliser des applications de contrôle du robot. Cette architecture est schématisée ci-dessous :

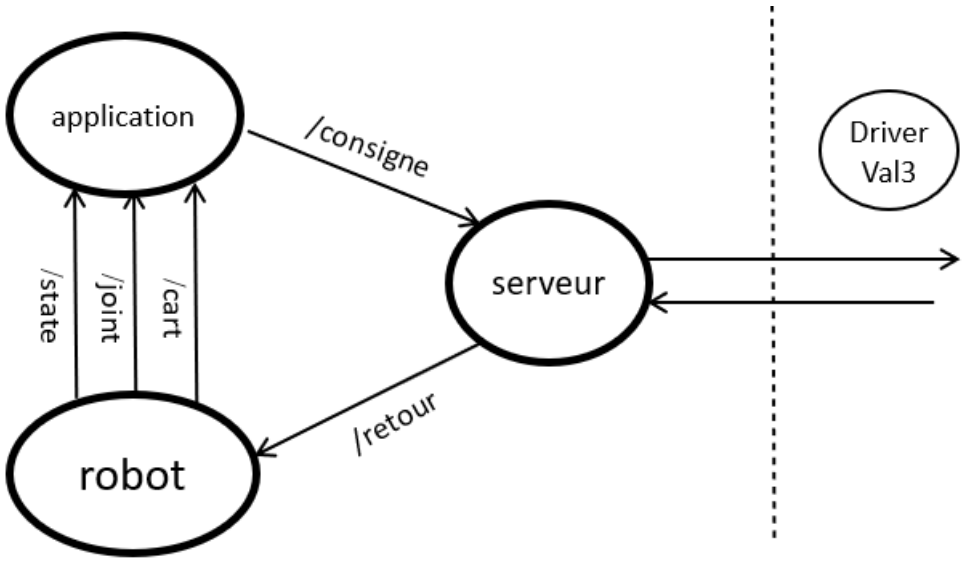


Figure 19 - Architecture de la couche ROS

Ces nœuds sont contenus dans la package *ros_arm* crée pour le développement de la couche ROS de la solution et communiquent entre eux via les topics suivants :

Topic	Type	Message
/retour	std_msgs/String	string data
/cart	ros_arm/Cart	double x double y double z double rx double ry double rz
/joint	ros_arm/Joint	double j1 double j2 double j3 double j4

		double j5 double j6
/state	ros_arm/State	int32 isPowered int32 isCalibrated int32 workingMode int32 esStatus int32 moniteurSpeed int32 shoulder int32 elbow int32 wrist int32 isSettled
/consigne	std_msgs/String	string data

Tableau 11 - Topics de la couche ROS

Les messages des topics /retour et /consigne possédant une structure simple, le type utilisé provient de la libraire standard des messages de ROS. Les autres messages possédant une structure plus complexe et étant adaptée à la solution développée, il a été nécessaire de les déclarer dans le package *ros_arm.*

3.1. Nœud serveur

Le nœud serveur est un nœud programmé en langage python (par soucis de simplicité de codage) qui n'est dédié qu'à l'initialisation et la création du serveur de communication et au transfert de message entre les autres nœuds et le driver.

Le serveur doit donc gérer deux types de communication différent, une communication publisher/suscriber avec les autres nœuds de la couche ROS et une communication TCP/IP avec le driver du contrôleur du robot. Pour se faire il fonctionne par l'intermédiaire de deux threads qui sont des processus s'exécutant en parallèle. Ces deux threads sont lancés dans la fonction main() et fonctionne de la façon suivante :

- Le premier thread exécute la fonction reception() qui souscrit au topic /consigne et réceptionne les messages de ce topic dans une fonction callback()
- Le deuxième thread exécute la fonction serveur() qui attends la connexion d'un client (ici le driver du contrôleur) avant de communiquer avec lui par l'envoie des messages réceptionnés dans le premier thread et par la réception des messages envoyés par le client. Les messages du client sont ensuite publiés sur le topic /retour.

Le nœud serveur est donc un nœud publisher et suscriber.

Le serveur est mono client, c'est-à-dire qu'une fois que la connexion est établie un client, d'autres client ne peuvent pas s'y connecter ce qui empêche la réception de donnée parasite. Le serveur gère aussi une perte de connexion proprement en fermant et en relançant la connexion pour attendre la reconnexion du client.

3.2. Nœud robot

Le nœud robot a pour but de d'extraire et de traduire les informations sur le robot des messages envoyés par le driver et de mettre ces informations à disposition du nœud application de façon structuré et exploitable. Ce nœud est programmé avec une vision objet lui permettant de structurer l'exécution de ses deux rôles. Ayant appris la programmation objet avec le langage C++, j'ai utilisé ce dernier pour programmer ce nœud.

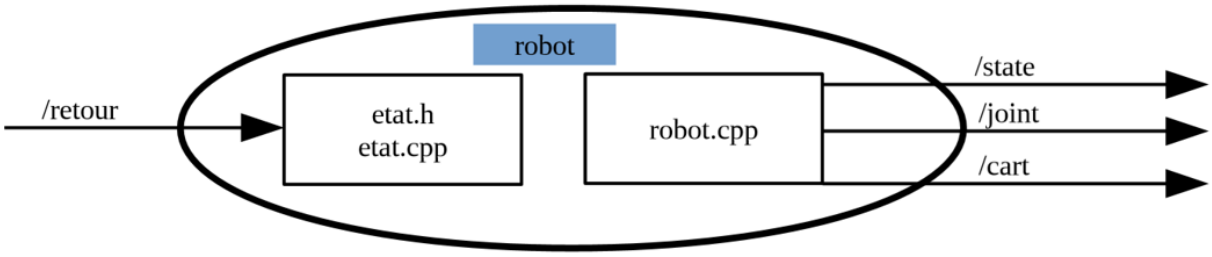


Figure 20 - Architecture du nœud robot

Ce nœud est composé d'une classe Etat et d'un programme principal robot dont leur rôle est le suivant :

- **Classe Etat :** souscrit au topic /retour puis réceptionne, traduit et met à jour les informations du robot dans des variables publiques via son unique fonction callback()
- **Programme robot :** initialise et crée le nœud robot puis publie sur le topic correspondant les valeurs des variables de l'objet *etat* de la classe Etat.

Ainsi lors de la création de l'objet *etat* par le programme robot, les informations du robot sont mises à jour et mis à la disposition des autres nœuds de la couche ROS.

3.3. Nœud application

Pour les mêmes raisons que le nœud robot, le nœud application est programmé en langage C++. Ce nœud est le siège de la programmation d'une application pour un utilisateur de la solution. Il est composé d'une classe Recept, d'une classe Control et d'un programme principale application.

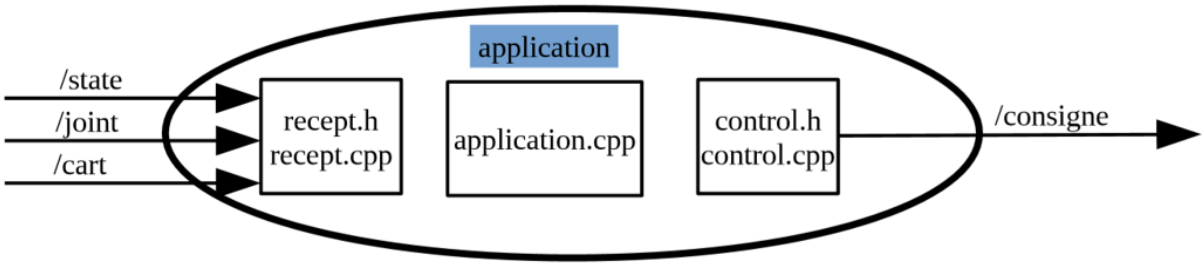


Figure 21 - Architecture du nœud application

Le fonctionnement de ces différents éléments est le suivant :

- **Classe Recept :** permet de souscrire à l'un des trois topics publiés par le nœud robot et met à jour ses variables publiques (qui sont les même que la classe Etat) avec les informations reçues.
- **Classe Control :** contient toutes les fonctions de contrôle du robot utilisables dans le programme application. Ces fonctions prennent en argument les valeurs d'une consigne qu'elles organisent en message de type string pour ensuite le publier sur le topic /consigne. Chaque fonction correspond à un message différent :

Fonction	Message publié
moveJoint	11 X j1 X j2 X j3 X j4 X j5 X j6 X F
moveCartJ	121 X x X y X z X rx X ry X rz X F
moveCartL	122 X x X y X z X rx X ry X rz X F
moveCartC	1231 X x X y X z X rx X ry X rz X F 1232 X x X y X z X rx X ry X rz X F
paramSpeed	2 X accel X vel X decel X tvel X rvel X blend X leave X reach X F
paramTool	31 X x X y X z X rx X ry X rz X otime X ctime X F
doTool	32 X action X F
paramConfig	41 X shoulder X elbow X wrist X F
setMoniteurSpeed	42 X speed X F
stop	99 X F

Tableau 12 - Fonctions de la classe Control

- **Programme application :** initialise et crée le nœud robot puis crée les objets nécessaires à la programmation de l'application pour le robot (objet *state* de la classe Recept, objet *joint* de la classe Recept, objet *cart* de la classe Recept, objet *control* de la classe Control). Un espace de ce programme est dédié à l'utilisateur de la solution pour déclarer les paramètres de l'application et programmer l'application par appel des fonctions des objets.

3.4. Planificateur de mouvement MoveIt

Afin d'intégrer à la solution l'outil de planification de mouvement MoveIt que propose ROS, j'ai utilisé le package **rx60_moveit_config** développé par l'université de

Kaiserslautern pour la configuration de MoveIt au robot Stäubli RX60. Le planificateur a été intégré à la solution de la manière suivante :

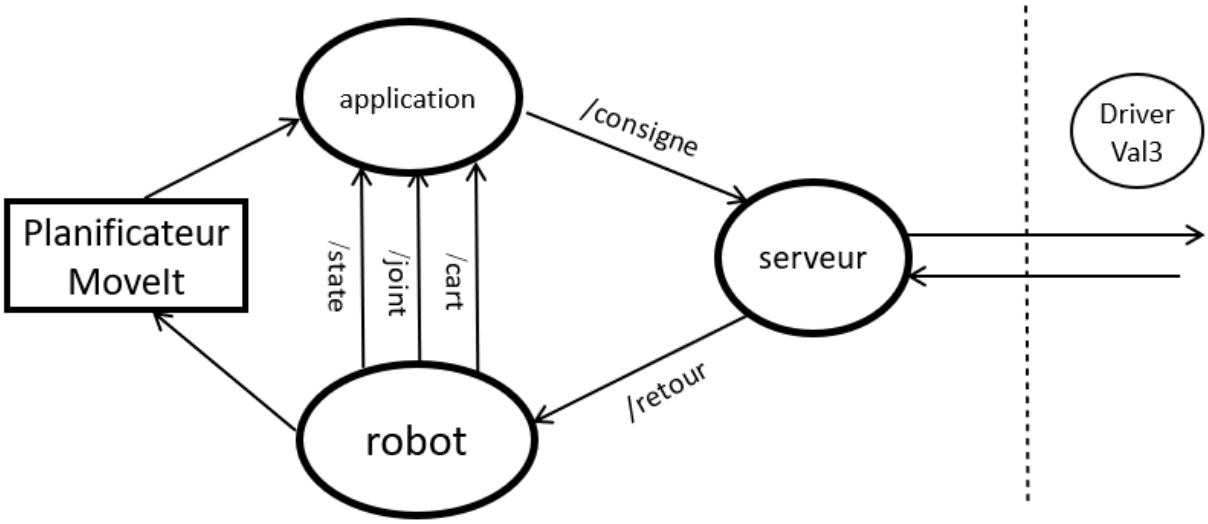


Figure 22 - Place du planificateur dans la solution

Aux vues de la complexité du fonctionnement du planificateur de mouvement, ce dernier a été considéré comme une boîte noire. L'objectif de son intégration dans la solution a donc été de trouver les bons topics d'entrée et de sortie afin de les connecter aux bons nœuds de la solution.

Le planificateur MoveIt fonctionne avec les coordonnées articulaires du robot. Quand il planifie un mouvement, il discrétise la trajectoire entre la position articulaire de départ du robot et la position articulaire de consigne en un nombre de position articulaire de passage variant selon la complexité du mouvement.

Suite à l'étude des topics utilisés par le planificateur grâce aux outils rqt_topic et rqt_graph de ROS qui permettent respectivement d'analyser les topics actifs en temps réel et l'architecture graphique des nœuds et des topics actifs, il est apparue que le planificateur récupère la position articulaire du robot réel via le topic */joint_states* du type sensor_msgs /JointState et que la trajectoire composée des positions articulaire de passage est publiée lors de l'exécution sur le topic */execute_trajectory/goal* du type moveit_msgs/JointTrajectory.

Les nœuds robot et application doivent donc être adaptés pour communiquer avec le planificateur. Le nœud robot publie alors les coordonnées articulaires du robot à la fois sur le topic */joint* et */joint_states* via le programme robot. Pour le nœud application, la classe Recept peut également souscrire au topic */execute_trajectory/goal* et une application dédiée à l'exécution du mouvement planifié est programmé. Cette application appelle successivement pour chaque position de passage la fonction de mouvement point à point en consigne articulaire *moveJoint* en paramétrant un lissage actif.

4. Synchronisation des données

Lors des premiers essais de la solution un problème de synchronisation des données a été mis en évidence. Lorsqu'une application était lancée, le contrôle accumulait de plus en plus de latence jusqu'à la saturation du buffer de la liaison TCP/IP faisant interrompre la connexion. Après la mesure du temps de traitement d'un message par le driver et la fréquence d'envoi de message par la couche ROS, il s'est avéré que la couche ROS envoyait les messages beaucoup plus rapidement que le temps de traitement du driver, faisant accumuler les consignes dans le buffer de la liaison TCP/IP jusqu'à sa saturation. Il a donc été nécessaire de synchroniser les échanges de messages.

Les mesures réalisées ont révélé que le temps de traitement des messages par le driver Val3 été compris en 30ms et 100ms selon les différents messages. La première idée a été de fixer en dur la fréquence d'envois des messages de la couches ROS à 33Hz correspondant donc à la fréquence maximale de traitement des messages par le driver. Cependant cette procédure a pour effet de saccader le mouvement de robot car l'accumulation de consignes dans la pile du contrôleur ne se fait plus. De plus lorsque plusieurs consignes d'actualisation de variable sont insérées entre deux consignes d'action, le robot se retrouve en attente de consigne d'action en plein mouvement.

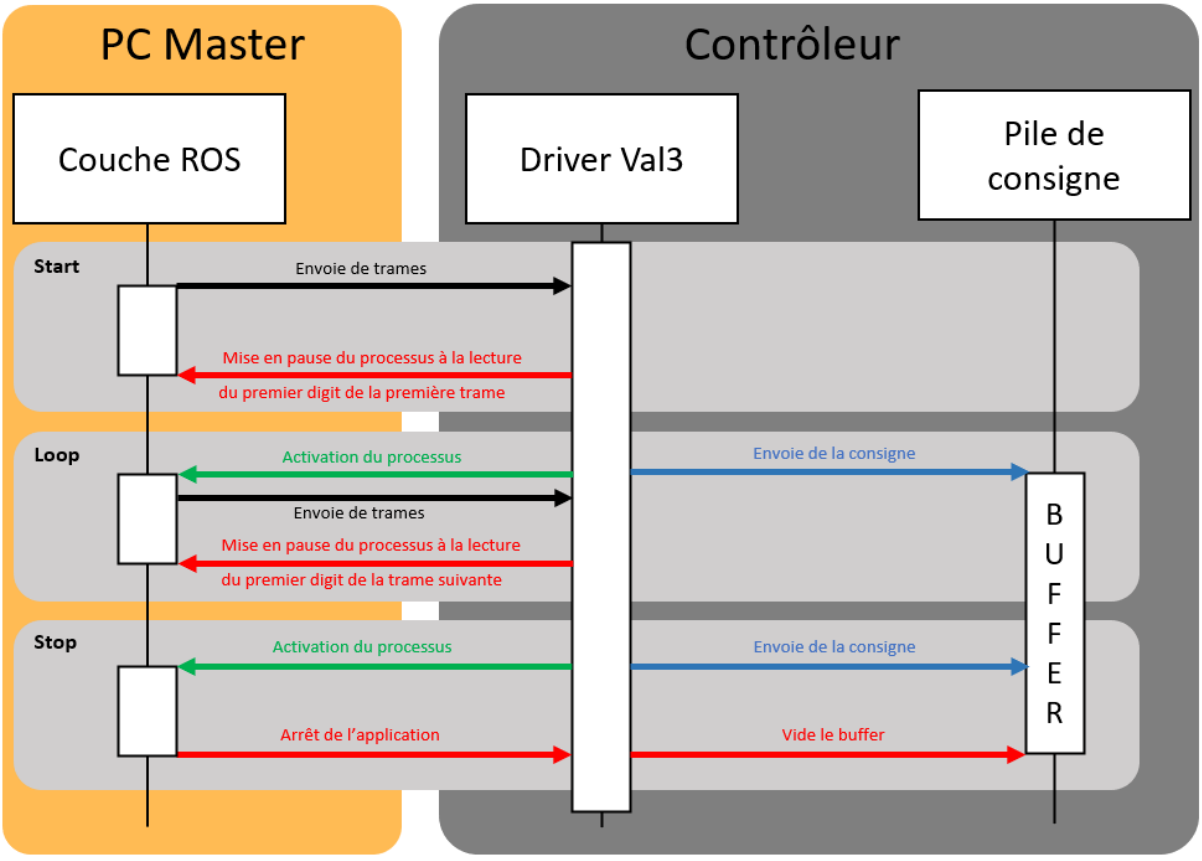


Figure 23 - Diagramme de séquence du contrôle du robot

La méthode retenue impose de fixer en dur une fréquence d'envoi des messages de la couche ROS supérieure à la fréquence de traitement des messages du driver (choix de 50hz) afin de permettre l'accumulations des consignes d'action dans la pile du contrôleur tout en retardant l'effet de saturation du buffer de la connexion. A cela s'ajoute l'implémentation d'un « flag », c'est-à-dire d'un signal d'autorisation d'envoi par le driver à la couche ROS, permettant d'exclure tout risque de saturation. L'effet de cette méthode est l'envoie rapide des messages par la couche ROS par paquet dans des intervalles de temps définit par le driver permettant à ce dernier de traiter des messages correspondant à l'état réel du robot. Cette synchronisation entre la couche ROS et le driver est détaillée dans la figure 23.

Validation de la solution

D'après ce qu'il a été convenu lors de l'élaboration du cahier des charge, la solution présentée dans la partie précédente a été mise en œuvre sur différentes applications de démonstration de tâches industrielles. Dû au contexte du stage imposant une longue période de télétravail, la validation des tâches industrielle n'a pu être mise en œuvre qu'à la toute fin du stage. Par le manque de temps pour mettre en place un protocole de mesure des données, la validation des tâches s'est faite de façon empirique.

1. Pick and place

La démonstration d'une tâche de pick and place permet de mettre en avant l'exécution des différents types de mouvement que propose le robot ainsi que la robustesse de la solution dans la durée.

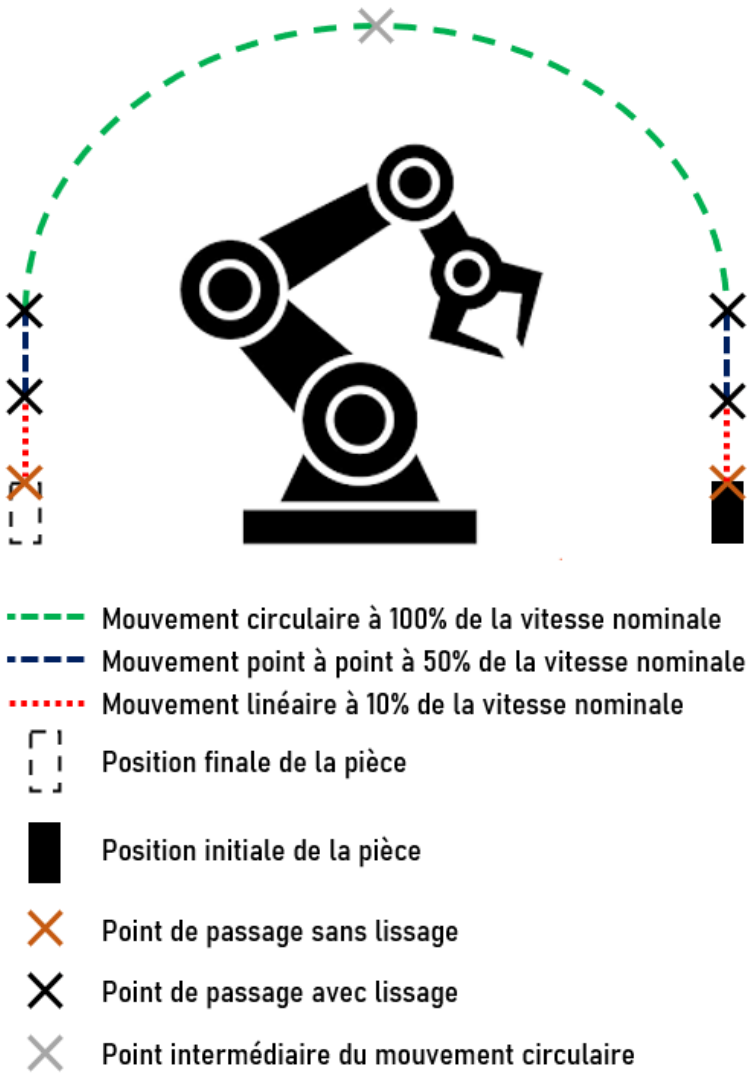


Figure 24 - Application de pick and place

Cette démonstration fait intervenir un mouvement circulaire, des mouvements linéaires et des mouvements point à point qui peuvent suivre des consigne articulaire ou cartésienne. La capacité de lissage et de mouvement d'approche sont également démontrés. Pour finir l'utilisation d'un outil (ici une pince pneumatique) est inclus dans cette manipulation.

Le test de cette démonstration s'est déroulé en fessant tourner cette application en boucle à des vitesses moniteur de plus en plus rapide sur de longue durée. Le test s'est révélé concluant par la bonne exécution des mouvements programmés, la bonne exécution de l'outil et par la robustesse de la solution qui à permit l'exécution de la tâche de pick and place sans aucun disfonctionnement sur toute la durée du test, soit deux heures, à la vitesse maximale du robot.

2. Suivi de contour de forme

Le démonstrateur de suivi de contour de forme a pour but de mettre en évidence la constance de la précision des mouvements contrôlés par la solution. Cette démonstration fait intervenir un outil composé d'une pince tenant un stylo afin de pourvoir dessiner la forme générale d'un objet quelconque.

Figure 25 - Dispositif du démonstrateur de suivi de contour de forme

Cette démonstration doit par la suite faire intervenir le système de vision du poste de travail du robot pour déterminer les points caractéristiques du contour d'une forme et les transmettre à la solution. Le développement de partie vision du projet RIO n'ayant pas encore commencé, j'ai réalisé un relevé des points caractéristique de la forme de manière manuelle :

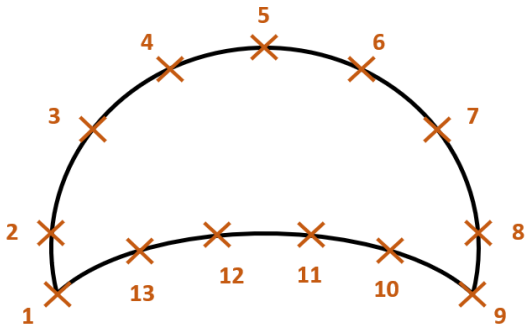


Figure 26 - Points caractéristique du contour de la forme

La prise du stylo par la pince n'étant pas très stable, j'ai réalisé le relevé de ces treize points en introduisant une distance d'un centimètre entre le passage du stylo et le contour de l'objet pour éviter un frottement pouvant faire bouger le stylo. Le résultat obtenu est le suivant :

Figure 27 - Résultat du suivi de contour de forme

On remarque tout d'abord que le contour dessiné par le robot respecte bien le contour de la forme. Le tracé ayant été réalisé par 100 itérations du suivi du contour de l'objet, on observe ensuite que le trait dessiné par le robot est resté fin, ce qui démontre la précision constante du suivi du contour de la forme.

3. Evitement d'obstacle

La tâche d'évitement d'obstacle a pour but de montrer les capacités que peut offrir les outils de ROS en les intégrants dans la solution. Ce démonstrateur montre donc la prise en compte de l'environnement du robot dans son contrôle par la solution via le planificateur de mouvement MoveIt. Un obstacle en mousse est placé dans l'environnement du robot et on donne pour consigne à ce dernier de se déplacer de part et d'autre de cette obstacle. La vitesse de fonctionnement du robot est réglée au maximum pour rendre la démonstration le plus visuel possible.

Dans un premier temps l'obstacle n'est pas déclaré dans la simulation du planificateur de mouvement :

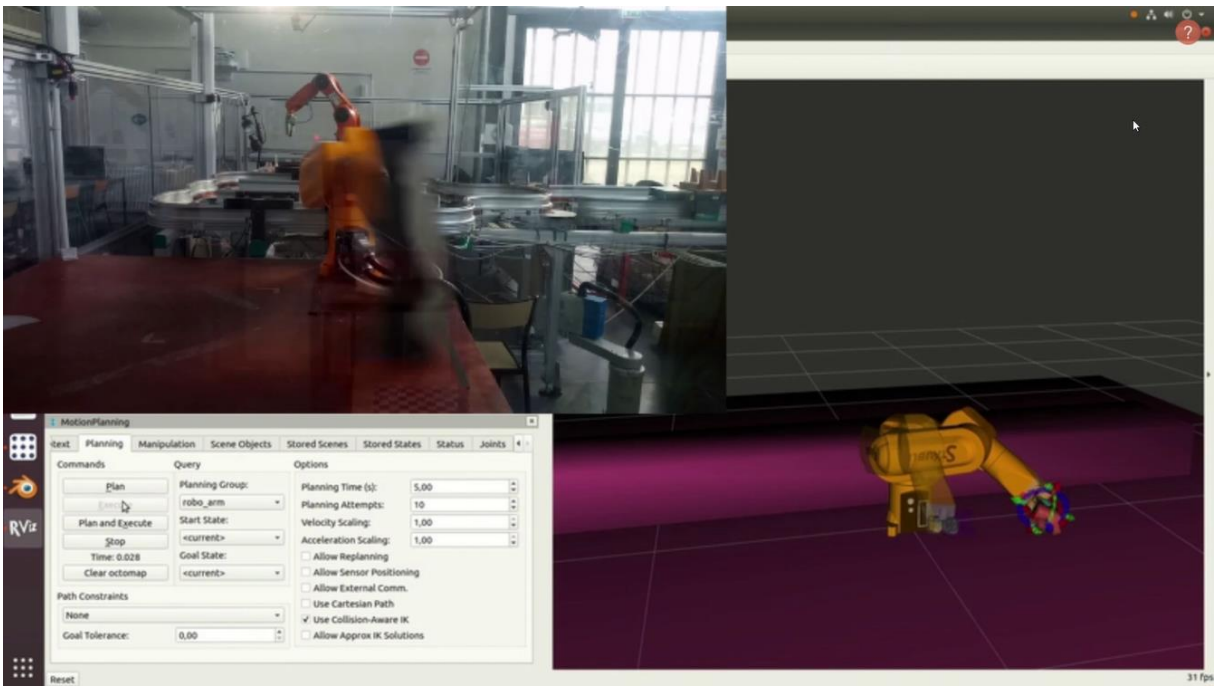


Figure 28 - Choc entre le robot et l'obstacle

Le planificateur de mouvement ne détecte pas d'obstacle entre le point de départ du robot et son point de consigne. Il planifie alors le chemin le plus directe. A l'exécution du mouvement planifié, on observe alors que le robot percute l'obstacle à pleine vitesse et l'envoi quelques mètres plus loin.

Dans un second temps, l'obstacle est déclaré dans la simulation du planificateur de mouvement. L'intérêt de cette partie de la démonstration est de comparer le changement de comportement du robot en fonction de la déclaration ou non d'un obstacle dans sa zone d'action :

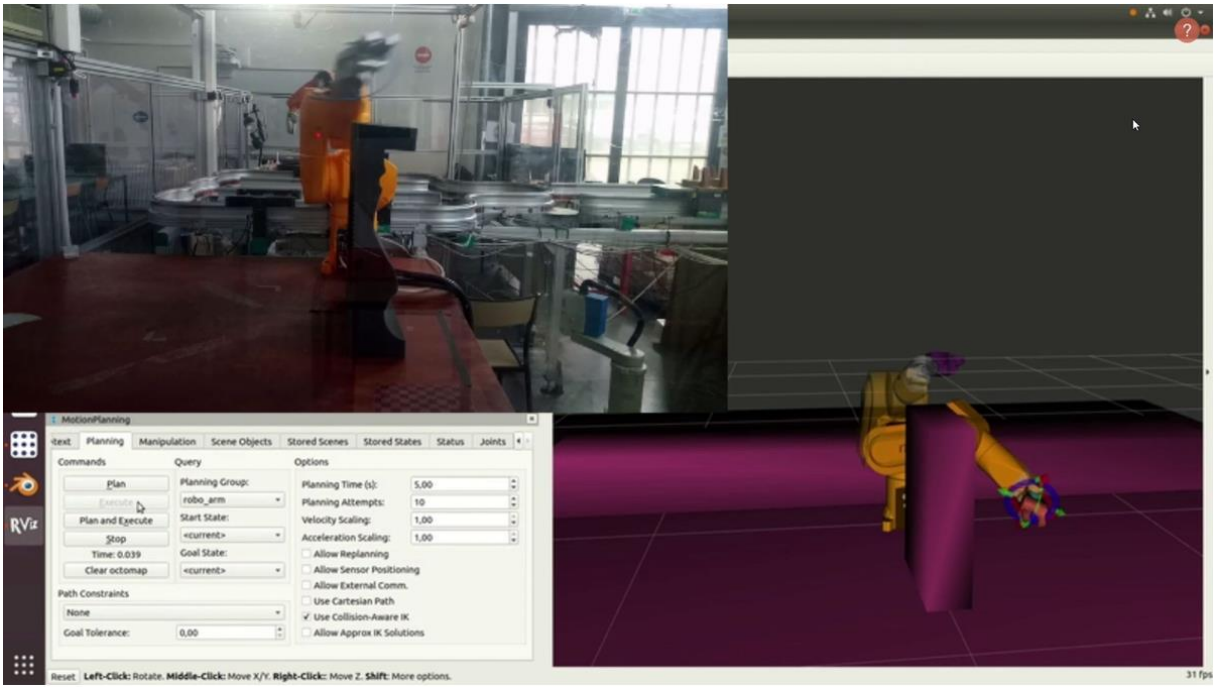


Figure 29 - Evitement de l'obstacle

Dans cette configuration, le planificateur de mouvement prend en compte la présence de l'obstacle et propose une trajectoire en cloche contournant l'obstacle par le haut. La trajectoire proposée n'est donc plus directe est fait intervenir des points de passages déterminés par la forme de l'obstacle. Lors de l'exécution de cette trajectoire, le robot réel réalise donc un mouvement de son point de départ jusqu'au point de consigne sans entrer en contact avec l'obstacle. Le mouvement du robot est donc dépendant de la déclaration de son environnement sur l'outil MoveIt de ROS.

Conclusion

Mon stage de six mois à la plateforme AIP-PRIMCA Occitanie est la dernière étape de ma formation d'ingénieur automatique et systèmes embarqués. La mission d'intégration de robot industriel sous ROS que j'ai effectué au sein de cette structure m'a permis de compléter cette formation d'un point de vue technique et personnel.

Cette partie du projet démarrant avec mon arrivée, j'ai participé, en se basant sur mes connaissances techniques acquises lors de ma formation et mes recherches bibliographique, à déterminer les objectifs et à développer un scénario de démonstration permettant de promouvoir au mieux l'utilisation de ROS dans un contexte se rapprochant au maximum du contexte industriel.

Ce travail a été la base du développement d'une solution de contrôle et de supervision de robot industriel présentant des caractéristiques de généricité et d'interopérabilité lui permettant d'être intégré dans un contexte plus large de chaîne de production industrielle.

La solution que j'ai développée a permis la réalisation de trois démonstrateurs opérationnels de tâche industrielle qui exploitent au maximum les capacités d'un robot industriel permettant ainsi de prouver la capacité de ROS à être utilisé dans le domaine de l'industrie.

D'un point de vue personnel, mon intégration au projet RIO m'a donné l'occasion d'approfondir mes connaissances dans le domaine de la robotique industriel et dans le domaine de la programmation informatique, notamment du développement sous ROS, qui sont deux domaines constituant ma spécialisation lors de ma formation à l'ENSISA. Ce stage s'inscrit ainsi dans une logique cohérente de ma formation d'ingénieur.

Pour finir, ce stage m'a permis de développer mes capacités d'organisation et de structuration de travail par le biais de l'élaboration d'un cahier des charges soumis à un calendrier précis. De plus j'ai aussi développé mes capacités d'autonomie et d'adaptation, notamment par l'adaptation du projet à un contexte particulier. Ces capacités me seront essentielles dans ma future vie professionnelle.

Références bibliographiques

[1] ISO 8373:2012(fr) Robots et composants robotiques – Vocabulaire
https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:fr

[2] Image extraite du site de l'AIP-PRIMECA : https://www.aip-primeca-occitanie.fr/

[3] Image extraite du site : http://aip-primeca.ups-tlse.fr/accueil.html

[4] Image extraite du site du réseau S.mart : https://s-mart.fr/

[5] Page du projet RIO : https://www.rosin-project.eu/ftp/ros-in-occitanie-rio

[6] Site du projet européen ROS'in : https://www.rosin-project.eu/

[7] Image extraite du site de ROS : https://www.ros.org/

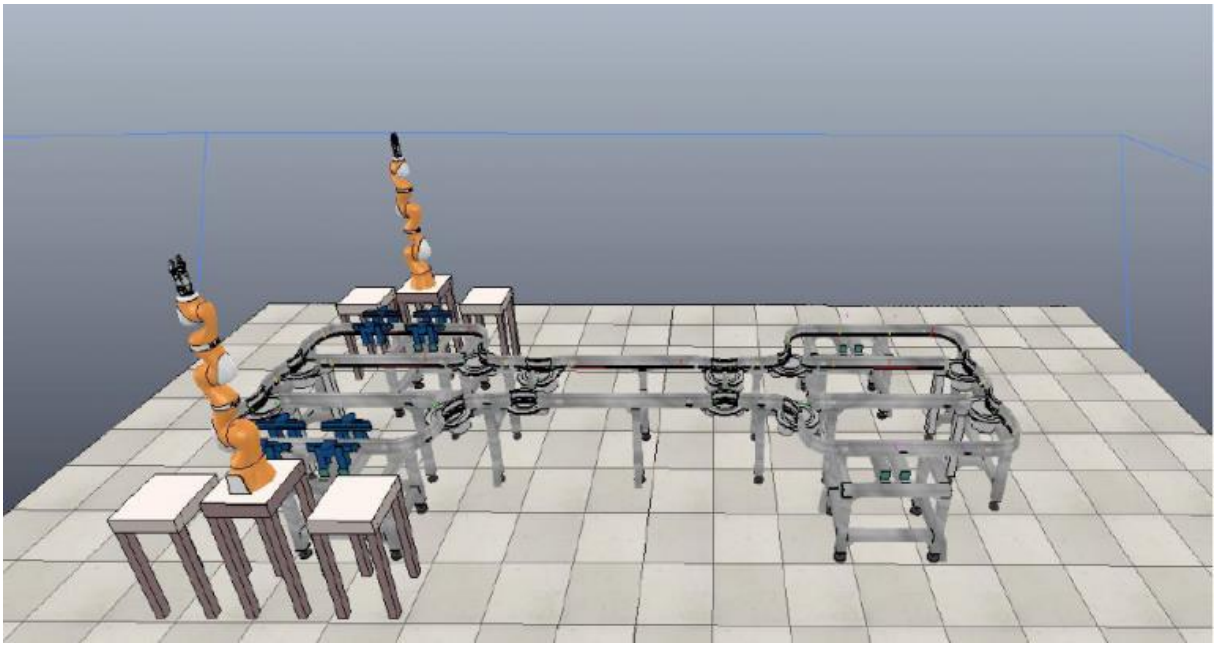
[8] Image extraite de la page Wikipédia de ROS :
https://fr.wikipedia.org/wiki/Robot_Operating_System

[9] *Robot Operating System*, Olivier STASSE, 2016

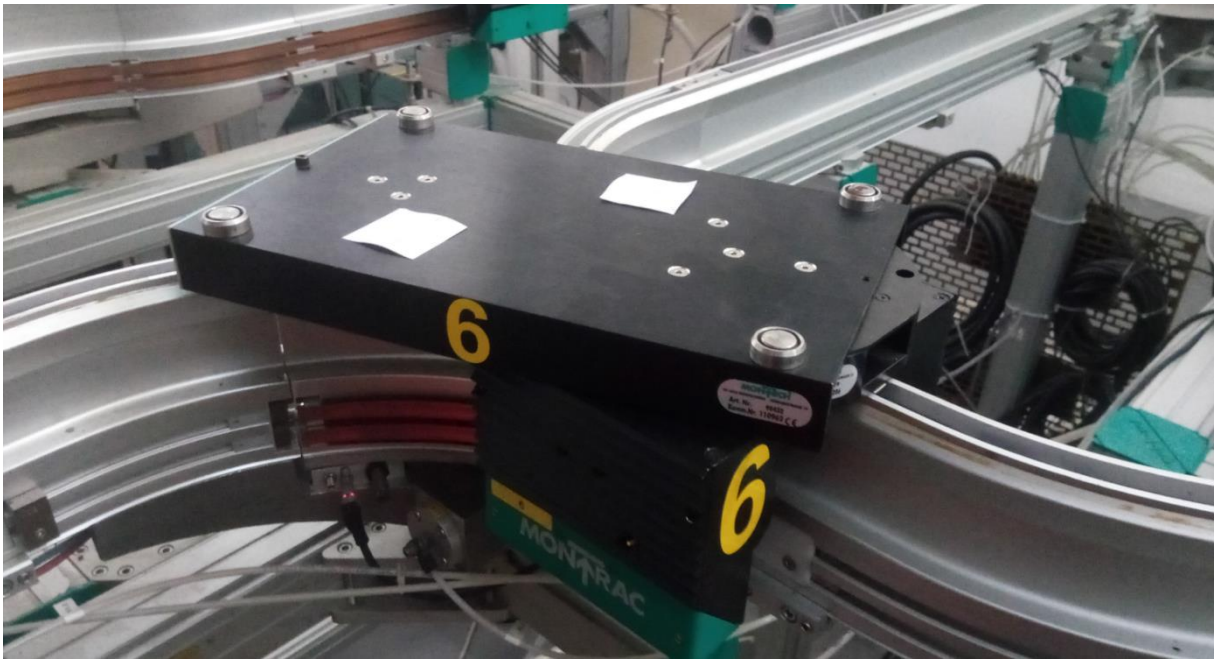
[10] *Manuel de Référence VAL3*, Stäubli, ver. 7, 2010

Annexes

Annexe 1 : Simulation sous VREP de la cellule flexible

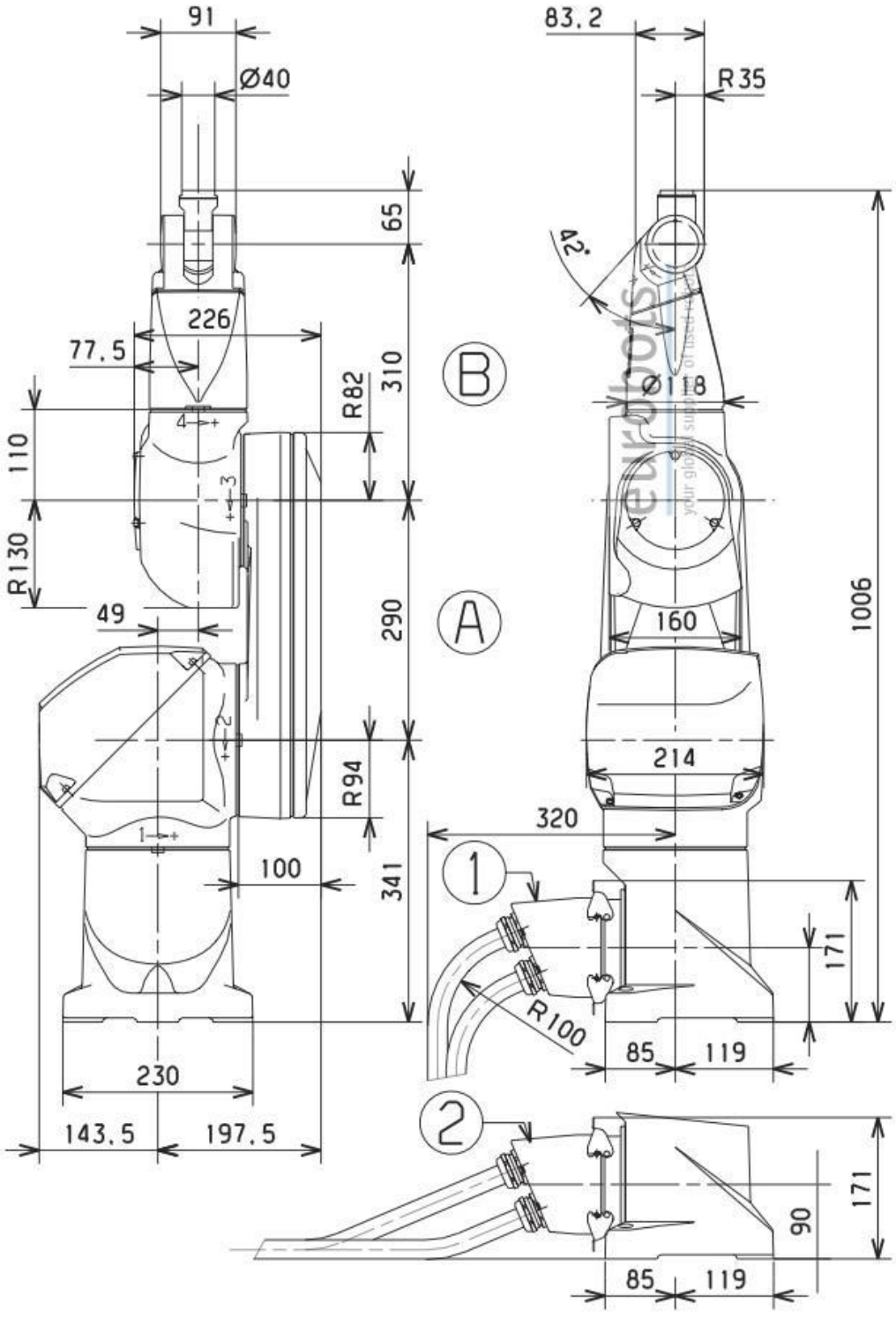


Annexe 2 : Navette automotrice de la cellule flexible



Annexe 3 : Module de travail de la cellule flexible

Annexe 4 : Dimensions du bras robot Stäubli RX60



Annexe 5 : Fiche technique du bras robot Kuka KR6 R700

KUKA

KR 6 R700-2

Caractéristiques techniques

Portée maximum	726 mm
Charge maximum	6,8 kg
Répétabilité de position (ISO 9283)	± 0,02 mm
Nombre d'axes	6
Position de montage	Sol; Plafond; Mur; Angle quelconque
Surface au sol	208 mm x 208 mm
Poids	env. 53 kg

Caractéristiques des axes

Plage de mouvements	
A1	±170 °
A2	-190 ° / 45 °
A3	-120 ° / 156 °
A4	±185 °
A5	±120 °
A6	±350 °

Conditions de service

Température ambiante lors du service	0 °C à 45 °C (273 K à 318 K)

Mode de protection

Degré de protection (IEC 60529)	IP65 / IP67
Degré de protection du poignet en ligne (IEC 60529)	IP65 / IP67

Contrôleur

Contrôleur	KR C4 compact

Certificats

Exigences ESD	IEC61340-5-1; ANSI/ESD S20.20

Boîtier de commande portatif

Boîtier de commande portatif	KUKA smartPAD

Temps de cycle

138 Cycles par minute (25 mm / 305 mm / 25 mm, 1 kg)

Graphique d'enveloppe d'évolution

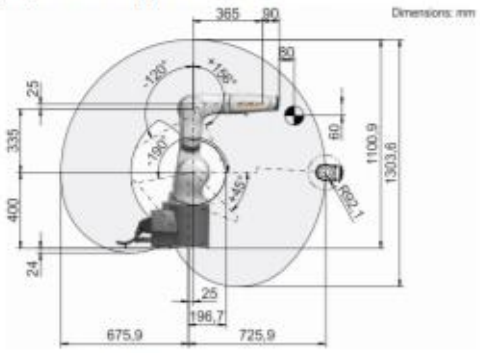
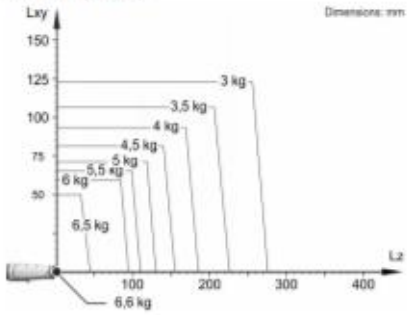
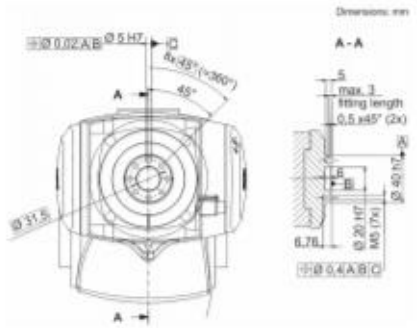


Diagramme des charges



KR 6 R700-2 est conçu pour une charge nominale de 3 kg pour une utilisation optimale de la performance et de la dynamique du robot. En réduisant les distances de charges, il est également possible d'utiliser des charges plus élevées jusqu'à la charge maximum. Les cas de charge spécifiques doivent être contrôlés avec KUKA.Load. Notre assistance technique KUKA Support est à votre disposition pour vous conseiller.

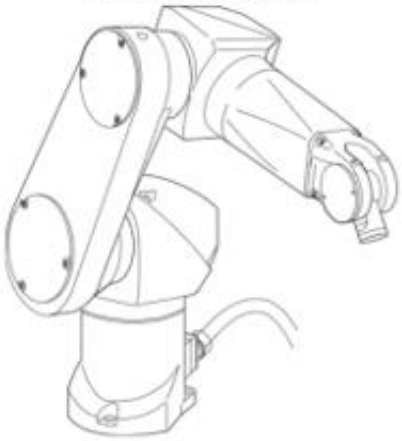
Bride de fixation



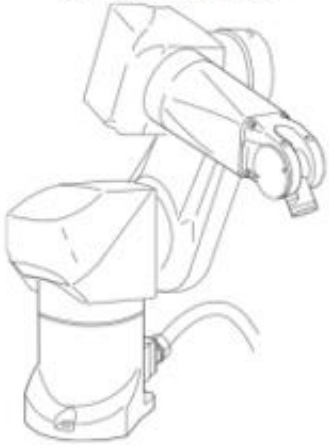
Annexe 6 : Configuration robot 6 axes

SHOULDER CONFIGURATION :

Configuration: righty



Configuration: lefty



ELBOW CONFIGURATION :

Configuration: enegative

Configuration: epositive

WRIST CONFIGURATION :

Configuration: wnegative

Configuration: wpositive

Annexe 7 : rqt_graph de du planificateur de mouvement