

ÉCOLE NATIONALE DES TECHNIQUES AVANCÉES

ROB314 : ARCHITECTURES LOGICIELLES POUR LA
ROBOTIQUE - PROJET

Conception d'un robot manipulateur

Auteure :
Caroline PASCAL

Responsable :
Emmanuel BATTESTI
Thibault TORALBA

13 Mars 2018



Table des matières

1	Introduction	2
2	Conception mécanique : Modélisation du robot manipulateur	2
2.1	Modélisation 3D du robot manipulateur - Matériel, inspiration et dimensionnement	2
2.2	Design et fabrication des pièces supplémentaires - Base du bras articulé et organe préhenseur	5
2.2.1	La base du robot manipulateur	5
2.2.2	L'organe préhenseur du robot manipulateur : la pince	6
2.3	Modélisation 3D du robot manipulateur sous ROS - Conversion en URDF . .	7
3	Conception logicielle : Contrôle en position du robot manipulateur	8
3.1	Génération de trajectoire - Intégration du framework <i>Moveit!</i>	8
3.2	Contrôle des servo-moteurs - Intégration du package <i>dynamixel_workbench</i>	9
3.3	Gestion de l'interface - Présentation du package <i>robot_arm_interface</i>	9
3.3.1	Planification et exécution des trajectoires	10
3.3.2	Gestion des commandes d'ouverture et de fermeture de la pince . . .	11
4	Mise en fonctionnement, essais et perspectives	12
4.1	Mise en fonctionnement et premiers essais	12
4.2	Perspectives d'amélioration	14
4.2.1	Côté conception mécanique	14
4.2.2	Côté conception logicielle	14
5	Conclusion	15

1 Introduction

Les **robots manipulateurs**, parfois aussi appelés **bras manipulateurs** à cause de leur ressemblance avec les bras humains, définissent les robots articulés permettant de réaliser des tâches de "manipulation", c'est-à-dire, capables de saisir, transporter et déposer des objets ou de la matière. Le plus souvent, ces actions sont réalisées à l'aide d'un outil spécifique, appelé effecteur terminal, qui peut prendre de nombreuses formes : pinces (motorisés, pneumatiques, hydrauliques), outils de découpe ou d'usinage (buse d'impression, scalpel, fil chaud), etc.

Depuis leur introduction sur les chaînes de montage de General Motors en 1961, les robots manipulateurs ont démontré leur habileté à dépasser les possibilités humaines, à la fois en vitesse et en force, et en particulier dans la réalisation de tâches dangereuses, complexes ou répétitives. C'est donc sans surprise que ces derniers se sont rapidement propagés dans d'autres secteurs d'activité : industrie (ABB, Kuka, Staübli,...), médecine (Da Vinci - Intuitive Surgical), nucléaire (Maestro - CEA), construction (ENPC Xtree), vente (Huawei Cobot), etc. Aujourd'hui encore, l'expansion des robots manipulateurs se poursuit, notamment à plus petite échelle, au sein des petites organisations et chez les particuliers. En effet, depuis quelques années, l'offre des robots manipulateurs compacts - un rayon d'action d'au plus 40cm pour une charge portée de quelques centaines de grammes, et un prix pouvant aller de 300 € à 2700 € - s'est particulièrement développée : le PhantomX de Interbotix, l'Ergo Jr de Poppy, le Niryo One de Niryo, etc.

Face à cette nouvelle tendance dans le milieu des robots manipulateurs, nous nous sommes donnés pour objectif de concevoir, à partir du matériel mis à notre disposition par le laboratoire U2IS, un de ces robots manipulateurs compacts, qui serait capable, par exemple, d'attraper les objets situés autour de lui. La conception de ce robot s'est déroulée en deux parties : une première partie dédiée à la **conception mécanique** du bras articulé et de son outil, et un seconde consacrée à la **conception logicielle** du contrôle en position du robot.

2 Conception mécanique : Modélisation du robot manipulateur

La première partie de ce projet a été consacrée à la conception mécanique du robot manipulateur, qui regroupe principalement le choix du matériel utilisé - moteurs, fixations, câblage,... -, la représentation sous la forme d'un modèle en 3D de l'assemblage du robot manipulateur, et le design ainsi que la fabrication des pièces supplémentaires.

2.1 Modélisation 3D du robot manipulateur - Matériel, inspiration et dimensionnement

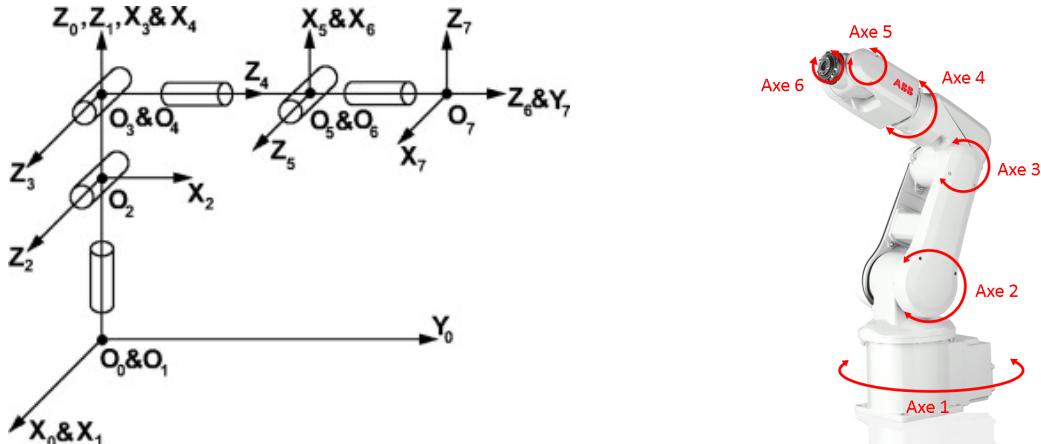
Matériel Lors du lancement de ce projet, le laboratoire U2IS a mis à notre disposition l'un des *Bioloid Beginners Kit* utilisés, entre autres, lors de la semaine "Ingénierie Système" de deuxième année.

Ce kit d'introduction à la robotique articulée est notamment composé de quatre servo-moteurs Dynamixel AX-12A, de pièces de liaison spécifiques en plastique rigide et de câbles TTL permettant d'assurer l'alimentation et la communication avec les moteurs. Afin de permettre la réalisation d'un robot manipulateur poly-articulé et piloté par l'intermédiaire de ROS, ce kit a été complété par trois servo-moteurs supplémentaires, et un adaptateur DYNAMIXEL2USB, permettant contrôler les moteurs depuis un ordinateur.

Dans le cadre de l'objectif que nous nous sommes fixés dans l'introduction, nous avons donc décidé d'utiliser au maximum les pièces de ce kit. Ainsi, nous avons pris la décision de n'utiliser que des servo-moteurs Dynamixel AX-12A, et d'essayer, si possible, de concevoir notre robot à partir des pièces de liaison fournies. Ce choix, en plus de nous permettre de réduire les coûts de fabrication et de conception liés à la création de nouvelles pièces, nous a également permis de tirer profit de la conception mécanique de ces pièces, tant par leur résistance que leur praticité d'adaptation aux moteurs et au câblage.

Inspirations Une fois le matériel à utiliser déterminé, nous avons décidé d'opter pour une architecture articulée à 6 axes, qui offre la possibilité d'atteindre n'importe quel point de l'espace d'action du robot avec n'importe quelle orientation - dans des termes plus mathématiques, cette architecture réalise une bijection entre l'espace articulaire et l'espace opérationnel du robot. Cette capacité est d'autant plus pertinente pour un robot manipulateur, qui sera amené à attraper des objets de formes et de dimensions variées disposés à différents endroits dans son espace de travail.

Pour définir l'agencement des 6 servo-moteurs dans la chaîne articulée de notre robot, nous nous sommes inspirés du plus petit robot manipulateur 6-axes commercialisé par le groupe ABB : le IRB120, dont le schéma cinématique est représenté sur la figure 1.



(a) Modèle cinématique 6-axes retenu

(b) Identification des 6 axes sur le robot IRB120

FIGURE 1 – Représentation schématique du modèle cinématique retenu et comparaison avec le robot IRB120

Cet agencement particulier est très répandu dans les robots manipulateurs industriels, car c'est certainement celui de rapprochant le plus de la morphologie des bras humains : les trois premiers axes (ou moteurs) représentent l'épaule et le coude, et les trois derniers axes

(ou moteurs), le poignet.

Modèle 3D La modélisation 3D de notre robot a été réalisée principalement sur le logiciel SolidWorks, à l'aide des fichiers de CAO en libre-téléchargement sur le site de la société Robotis.

La plupart des liaisons entre les 6 servo-moteurs du robot manipulateurs ont pu être réalisée en utilisant les pièces fournies dans le kit Bioloid, sans modification supplémentaires. Toutefois, nous avons rencontré quelques difficultés pour les liaisons entre les axes 3 et 4, et les axes 5 et 6. En effet, ce type de liaison (rotation de $\frac{\pi}{2}$ autour de l'axe X du premier axe) ne peut pas être réalisé simplement à partir des pièces fournies dans le kit Bioloid. Pour résoudre ce problème, nous avons décidé de "détourner" l'utilisation de certaines pièces et éléments de fixation du kit afin de réaliser nous-même ces liaisons manquantes. Ces éléments de liaison "hybrides" sont visibles sur la dernière version du modèle 3D de notre robot - figure 2.

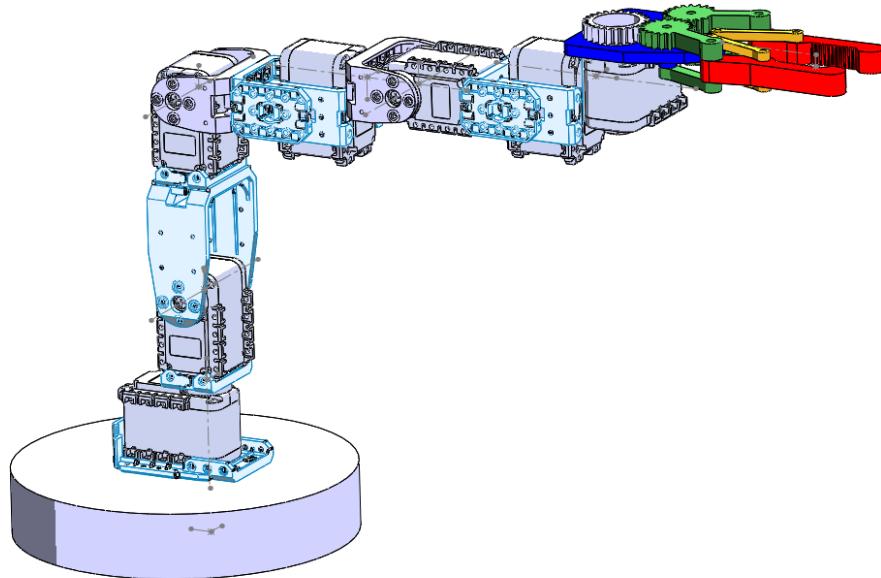


FIGURE 2 – Modèle 3D du robot manipulateur - Version finale

La configuration en "coude" dans laquelle est représenté notre robot correspond à sa configuration de référence, c'est-à-dire la configuration dans laquelle tous les servo-moteurs sont dans la position 0 rad.

Dimensionnement Avant de s'engager dans la partie logicielle de notre projet, nous avons décidé de réaliser une étape intermédiaire de dimensionnement statique, visant à déterminer la viabilité de l'assemblage finalement retenu. Pour ce faire, nous nous sommes intéressés de plus près aux caractéristiques mécaniques des servo-moteurs Dynamixel AX-12A, dont les plus pertinentes sont résumées dans le tableau 1¹.

1. Source : <http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/>

	Caractéristiques
Poids	54,6 g
Couple de décrochage	1,5 Nm
Vitesse à vide	6,18 rad/s
Échelle de positions	[-2,61 rad ; 2,61 rad]



TABLE 1 – Caractéristiques mécaniques et visuel d'un servo-moteur AX-12A

Nous noterons en particulier que la seule valeur de couple fournie par la documentation est celle du couple de décrochage, c'est-à-dire la valeur à partir de laquelle le moteur ne peut plus soutenir la charge qui lui est appliquée. En conséquence, le seul calcul de dimensionnement que nous avons réalisé est un dimensionnement de "décrochage", et non pas un dimensionnement de "fonctionnement nominal" - qui aurait pourtant été bien plus pertinent dans notre situation.

De plus, à ce stade de la phase de conception mécanique de notre robot, nous n'avions pas encore d'informations sur le poids de l'outil dont il allait être équipé, et sur la charge utile qu'il allait être amené à transporter. Nous avons donc décidé d'estimer, respectivement, à une centaines de grammes le poids de l'outil et de la charge utile transportée.

Dans le pire des cas, c'est-à-dire lorsque le bras est chargé et complètement étendu, nous aboutissons à un couple de charge statique d'environ 0,1 Nm sur le second axe, l'axe le plus critique pour notre robot. Cette valeur étant inférieure à la valeur du couple de décrochage de ce moteur, nous avons donc validé le dimensionnement de l'assemblage retenu pour le robot manipulateur.

2.2 Design et fabrication des pièces supplémentaires - Base du bras articulé et organe préhenseur

Comme l'illustre le modèle 3D complet de notre robot - figure 2 - les pièces fournies dans le kit Bioloid n'ont finalement pas été suffisantes pour concevoir l'intégralité de l'assemblage. En effet, une fois le corps du robot manipulateur assemblé, deux pièces supplémentaires étaient encore manquantes : une pièce à la base du bras articulé, qui reliera l'espace de travail et le premier servo-moteur, et l'outil, qui permettra au robot manipulateur de saisir des objets.

2.2.1 La base du robot manipulateur

L'objectif de la pièce située à la base de notre robot est double : elle doit pouvoir à la fois accueillir un servo-moteur et maintenir le robot sur un support horizontal et plan, comme une table par exemple.

Le design que nous avons finalement retenu est représenté sur la figure 3. Ce design a été pensé pour être usiné et percé grâce à une fraiseuse numérique 3-axes.

Concernant le premier objectif assuré par cette pièce, nous avons choisi d'utiliser comme interface avec le servo-moteur une des pièces de fixation planes fournies dans le kit Bioloid.

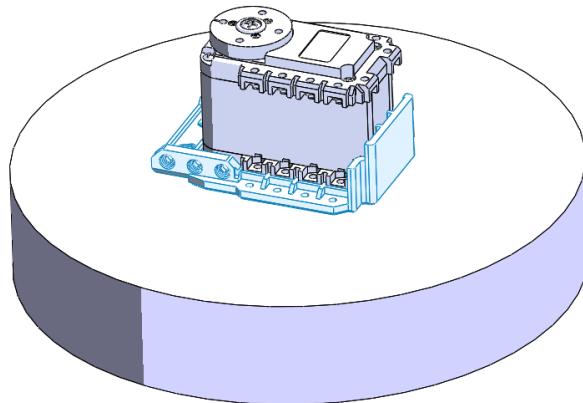


FIGURE 3 – Détail du modèle 3D retenu pour la pièce située à la base du robot manipulateur

Cette pièce est fixée au moyen d'écrous et de vis, dont les têtes seront "cachées" dans une poche usinée à cet effet.

Pour ce qui est du second objectif rempli par cette pièce, le choix du matériau d'usinage nous a été bénéfique. En effet, le laboratoire U2IS nous a proposé d'utiliser un matériau plastique très dense, mais pourtant propice au fraisage numérique. Afin de tirer parti de cette densité, nous avons émis l'hypothèse que le poids propre de la pièce permettrait à lui seul de compenser le poids du robot manipulateur et les efforts d'inertie à faible vitesse.

La pièce obtenue après usinage s'adapte parfaitement à la pièce d'interface provenant du kit Bioloid, et permet de fixer le moteur tout en laissant passer les câbles d'alimentation. Ses dimensions, 3 cm de hauteur pour un rayon de 15 cm, permettent également d'envisager l'utilisation d'un serre-joint dans le cas où le poids propre de la pièce ne suffirait pas à maintenir le robot manipulateur.

2.2.2 L'organe préhenseur du robot manipulateur : la pince

Le choix de l'outil dont le bras articulé allait être équipé s'est effectué très rapidement : le rôle du robot étant d'attraper des objets proche, il doit être équipé d'une pince (logique).

Nous décidâmes d'opter pour un système de pince motorisée à l'aide d'un servo-moteur AX-12A. Le modèle 3D que nous avons finalement retenu, inspiré de celui proposé par M. Ömer Yıldız sur le site GrabCAD, est représenté sur la figure 4.

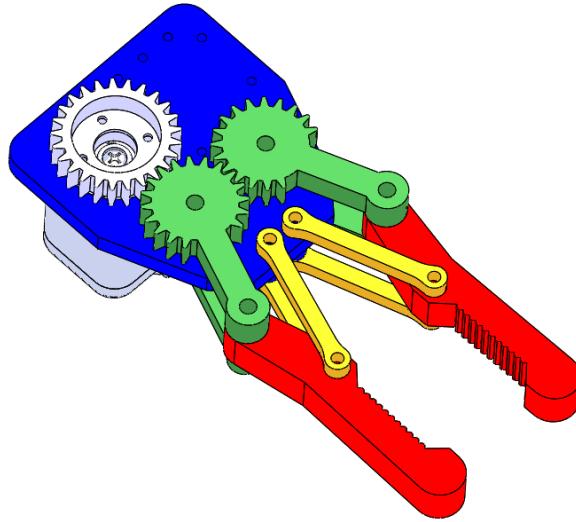


FIGURE 4 – Détail du modèle 3D retenu pour l'organe préhenseur du robot manipulateur

Ce modèle, très intéressant d'un point de vue mécanisme, repose sur un système de deux engrenages, dont le mouvement symétrique permet l'ouverture et la fermeture de la pince. Cet assemblage est entraîné par un troisième engrenage monté directement sur le servo-moteur, qui est lui-même fixé sur la base de l'outil. En pratique, l'ouverture et la fermeture de la pince peuvent être déclenchées en actionnant le servo-moteur jusqu'aux positions correspondantes.

Les différentes pièces de l'assemblage constituant la pince ont été conçues afin de permettre leur fabrication par impression 3D - Pièces planes, sans porte à faux. Cette méthode de fabrication numérique nous a permis d'obtenir le niveau de détail nécessaire à l'imbrication des engrenages et à l'emboîtement des pièces de l'assemblage, tout en assurant le bon fonctionnement du mécanisme (et un temps d'impression de moins de 6 heures!).

Malheureusement, quelques difficultés lors du décollage de la pièce constituant la base de la pince ont causé une légère déformation au niveau de la zone de fixation du servo-moteur, qui a entraîné un désaxage de l'axe du moteur par rapport au reste du mécanisme. En conséquence, l'engrenage d'entraînement n'est plus parfaitement imbriqué avec l'engrenage de manœuvre de la pince, causant, dans certains cas très rares, un "décrochage" du mécanisme de manœuvre.

De plus, une fois l'assemblage final de la pince réalisé, nous nous sommes rendus compte que les vis et écrous choisis pour fixer les pièces entre elles induisaient un jeu assez important, au point qu'une position du servo-moteur donnée pouvait mener à plusieurs configurations de la pince différentes. Le contexte de notre projet ne nécessitant pas une précision millimétrique dans les manœuvres de la pinces, nous avons finalement décidé de conserver la pince ainsi réalisée.

2.3 Modélisation 3D du robot manipulateur sous ROS - Conversion en URDF

Afin d'assurer la transition entre la phase de conception mécanique et la phase conception logicielle, il a tout d'abord été nécessaire de convertir la modélisation 3D de notre robot au

format URDF (Universal Robotic Description Format). Pour ce faire, nous avons eu recours au plugin SolidWorks *sw_urdf_exporter*², qui permet d'exporter un assemblage SolidWorks sous la forme d'un package ROS compatible avec les outils *RViz* et *Gazebo*.

Pour fonctionner correctement, cet outil de conversion nécessite que l'assemblage SolidWorks adopte une structure particulière : le robot manipulateur doit être décomposé sous la forme d'une successions de solides (ou "parts"), reliés entre eux par des liaisons pivot au niveau des axes des moteurs !

3 Conception logicielle : Contrôle en position du robot manipulateur

La seconde partie de ce projet a été dédiée à la conception logicielle du contrôle en position du robot manipulateur. En d'autres termes, cette partie regroupe la programmation des briques logicielles en charge de la génération de trajectoire, au contrôle des servo-moteurs, ainsi qu'à la gestion de leur interface.

3.1 Génération de trajectoire - Intégration du framework *Moveit!*

Qui dit "contrôle en position" sous-entend implicitement "génération de trajectoire", et dans cette perspective, nous avons décidé d'utiliser le package ROS *Moveit!*.

Le framework *Moveit!*³ est une plateforme qui permet de générer et de visualiser sous *RViz* des trajectoires 3D pour n'importe quel assemblage, sous réserve qu'il soit décrit au format URDF. Cet outil, très puissant à fortiori, est très complet, et intègre de nombreux paramètres dans la génération des trajectoires : "self-collisions", positions inatteignables par rapport aux positions limites des moteurs, évitement d'obstacles statiques et dynamiques, évitement des positions de singularité,... Nous noterons, toutefois, que ce package ne permet pas (encore) de modéliser les déformations mécaniques des corps en mouvements : les assemblages, et par extension, les robots, sont donc considérés parfaitement rigides.

En pratique, l'interface de *Moveit!* est très simple à prendre en main : il suffit de définir les poses⁴ de la position de départ et de la position d'arrivée pour pouvoir visualiser la simulation de la trajectoire générée - Une simulation très utile pour vérifier et valider une trajectoire avant exécution !

Moveit! propose une large base de donnée de robots pré-existants (ABB, Kuka, Interbotix,...), mais offre également la possibilité de manipuler des robots entièrement personnalisés grâce à un *Setup Assistant*, qui génère automatiquement un package ROS dédié à partir d'un fichier de description URDF. Cet outil d'intégration nous a donc permis de créer un package *Moveit!* dédié à notre robot, au détail près du fichier de configuration relatif aux limites en position, vitesse et couple des servo-moteurs qui a du être complété à la main.

2. Source : https://wiki.ros.org/sw_urdf_exporter

3. Source : <https://moveit.ros.org/>

4. Dans notre contexte, une **pose** correspond à la donnée de la position et de l'orientation du repère lié à l'effecteur terminal d'un robot manipulateur.

3.2 Contrôle des servo-moteurs - Intégration du package *dynamixel_workbench*

Une fois la génération de trajectoire simulée, il a ensuite été question de commander les sevo-moteurs de telle sorte que la position désirée soit également atteinte dans la réalité. Dans cette optique, nous avons opté pour le package ROS *dynamixel_workbench*, qui repose sur le kit de développement *DynamixelSDK*⁵.

Au sein du package ROS *dynamixel_workbench*, nous nous sommes intéressés en particulier au sous-package *dynamixel_workbench_controllers* qui, comme son nom l'indique, permet de piloter les servo-moteurs Dynamixel. Il est important de signaler, qu'à l'inverse du framework *Moveit!*, le package *dynamixel_workbench_controllers* repose sur de nombreux paramètres liés au hardware (baudrate et IDs des moteurs, port USB utilisé, positions, vitesses et couples maximaux,...) et qu'il est indispensable de bien les renseigner dans les fichiers de configuration !

De manière générale, les servo-moteurs Dynamixel possèdent deux modes de fonctionnement : le mode "wheel" et le mode "joint". Le premier mode permet de piloter le moteur comme une roue, c'est-à-dire en ne précisant qu'une consigne de vitesse, sans contrainte sur la position. A l'inverse, le second mode permet de piloter le moteur grâce à une consigne de position, sans pouvoir moduler la vitesse à laquelle le déplacement sera réalisé. En d'autres termes, les mouvements du moteurs seront tous réalisés à la même vitesse, qui correspond, en pratique, à la vitesse maximale précisée dans les fichiers de configuration du package, et qui peut être éventuellement modifiée. Malgré cette contrainte, c'est donc le mode de fonctionnement "joint" que nous avons choisi pour piloter les 6 servo-moteurs de notre robot.

Lorsque les sevo-moteurs sont dans le mode de pilotage "joint", deux types de consignes de positions sont prises en charge par le package *dynamixel_workbench_controllers* : une position unique, codée sur un entier compris entre 0 et 1023, qui sera ensuite traduit en une position comprise entre -2,61 rad et 2,61 rad, ou une succession de positions, renseignées directement en radians, sous la forme d'un *trajectory_msgs* : *:JointTrajectory*. Dans le cadre de notre projet, ces deux types de consignes se sont avérés complémentaires : le servo-moteur dédié aux manœuvres de la pince n'a besoin que d'une consigne de position unique (position ouverte ou fermée), tandis que les servo-moteurs du bras articulé sont amenés à suivre une succession de positions calculées à l'avance.

3.3 Gestion de l'interface - Présentation du package *robot_arm_interface*

A ce stade de notre projet, il ne nous manquait donc plus qu'à réaliser l'interface entre le framework de génération de trajectoire *Moveit!* et le package ROS de contrôle des servo-moteurs *dynamixel_workbench*. A cette fin, nous avons décidé de créer un nouveau package ROS dédié uniquement à cette tâche : le package *robot_arm_interface*.

Ce package, très simple, n'est composé que d'un fichier de lancement - permettant de lancer les noeuds à interfaçer - et d'un fichier source C++, contenant le corps du noeud ROS développé, et dont nous allons détailler le fonctionnement.

5. Source : https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_workbench/

3.3.1 Planification et exécution des trajectoires

Après quelques recherches préliminaires, nous nous sommes rapidement rendus compte qu'il serait complexe de réaliser directement une interface entre la simulation sur *RViz* de la trajectoire calculée par *Moveit!* et le pilotage de servo-moteurs. Aussi, nous avons décidé d'utiliser directement l'API C++ proposée par *Moveit!* qui permet réaliser la planification des trajectoires directement depuis le code source du noeud.

Cette API se présente sous la forme d'une classe C++ très simple d'utilisation, qui cache en réalité une interface ROS complexe - topics, services et actions - avec le noeud */move_group* du package ROS généré grâce à *Moveit!* dans la section 3.1.

Dans un premier temps, nous avons décidé de préciser la pose objectif (i.e. à atteindre à la fin de la trajectoire) de notre robot "en dur" dans le code source, de sorte que cette position soit bien située dans l'espace atteignable par le robot manipulateur. A terme, cette pose est censée représenter la position de l'objet que le robot doit attraper et l'orientation avec laquelle l'utilisateur souhaite attraper cet objet - par dessous, par dessus, de côté,... Nous noterons qu'il est également possible de renseigner une pose objectif sous la forme d'une configuration articulaire, c'est-à-dire par la donnée des positions de chaque servo-moteur.

En partant de cette pose objectif et de la configuration articulaire courante du robot, il est alors possible de planifier une trajectoire, qui sera stockée sous la forme d'un *trajectory_msgs* : *:JointTrajectory*. Ce message structuré décrit la trajectoire de manière très complète : positions, vitesses et accélérations des 6 servo-moteurs à intervalles de temps réguliers, et donnée du temps écoulé depuis le début de l'exécution de la trajectoire. Cependant, comme nous l'avons expliqué dans la section 3.2, le mode de fonctionnement retenu pour les servo-moteurs Dynamixel ne permet pas de contrôler la vitesse, et encore moins l'accélération des moteurs. En conséquence, les deux tiers des informations de ces messages, pourtant indispensables pour obtenir un mouvement harmonieux, seront complètement ignorées.

La communication instaurée par l'API *Moveit!* fonctionnant de manière bipartite, il est également possible de visualiser et de simuler les trajectoires planifiées depuis le code source sur *RViz*. Nous avons décidé de tirer profit de cette fonctionnalité afin de permettre une validation visuelle de la trajectoire avant son exécution - Il n'y jamais trop de prudence.

Une fois la trajectoire confirmée, son exécution est ensuite réalisée en deux temps :

1. Tout d'abord, le message *trajectory_msg* : *:JointTrajectory* de la trajectoire calculée est publié sur le topic *dynamixel_workbench/joint_trajectory* du noeud éponyme. Cette publication permet de transmettre les informations relatives à la trajectoire à effectuer sans pour autant déclencher son exécution ;
2. Ensuite, une requête *std_srvs* : *:Trigger* est envoyée sur le service *dynamixel_workbench/execution* du noeud éponyme afin de déclencher, cette fois, l'exécution de la trajectoire. La réponse de cette requête signale si la communication s'est bien réalisée, mais ne nous donne en revanche aucune information sur l'avancement de l'exécution de la trajectoire.
3. Finalement, l'exécution du code est bloquée durant une durée artificielle de 5 secondes pour attendre la fin du mouvement du robot manipulateur.

Les interfaces et échanges explicités dans cette sections sont résumés dans le diagramme

de la figure 5. Nous noterons que l'échange représenté en bleu correspond en réalité à l'interface encapsulée par l'API *Moveit!*.

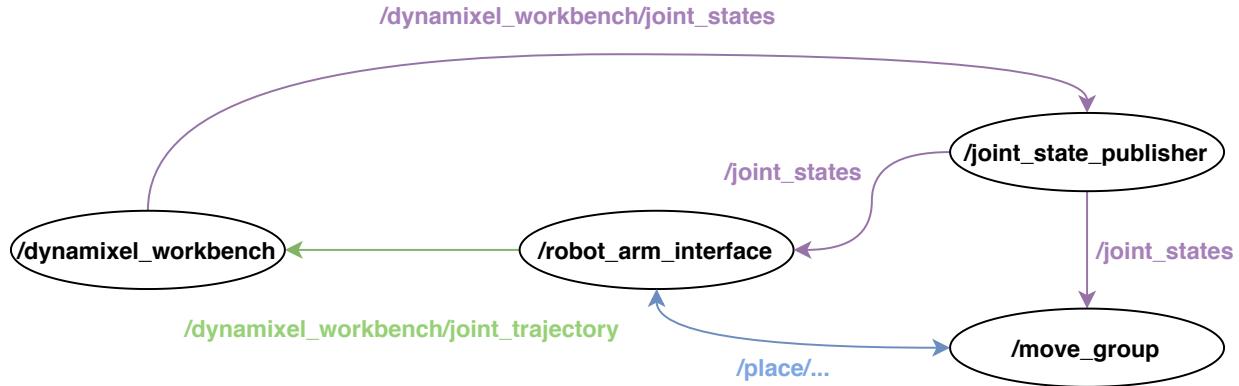


FIGURE 5 – Représentation simplifiée du rqt_graph associé aux nœuds impliqués dans le package *robot_arm_interface*

Nous remarquerons également que des liens de rétroaction partant du nœud */dynamixel_workbench* vers les nœuds */move_group* et */robot_arm_interface* y sont représentés. Ces liens, gérés par le nœud */joint_state_publisher*, permettent de mettre à jour en temps réel - sur la visualisation *RViz*, entre autres - la configuration articulaire courante du robot manipulateur en se basant sur les données fournies par les servo-moteurs.

3.3.2 Gestion des commandes d'ouverture et de fermeture de la pince

Une fois la gestion de la planification et de l'exécution des trajectoires mise en place, nous avons pu nous focaliser sur les manœuvres d'ouverture et de fermeture de la pince.

En première approche, nous avons décidé de déterminer de manière complètement empirique les positions du servo-moteur correspondant aux états "ouverte" et "fermée" de la pince. Pour permettre au robot manipulateur de saisir des objets sans forcer la fermeture complète de la pince, l'état "fermée" a été défini comme l'état dans lequel la pince est fermée sur l'objet à attraper, sans pour autant le couper en deux.

Cette détermination empirique a été réalisée grâce au logiciel *Dynamixel Wizard 2*⁶, qui permet de manipuler les servo-moteurs Dynamixel depuis une interface graphique, et les valeurs ainsi mesurées ont été directement reportées en "dur" dans le code source du nœuds⁷.

Finalement, l'actionnement de la pince est réalisé - très simplement - en envoyant une requête mentionnant l'ID du servo-moteur, la mention de l'adresse "Goal_Position" et la consigne correspondant à l'état souhaité, sur le service *dynamixel_workbench/dynamixel_command*. A l'instar de la requête envoyée pour déclencher l'exécution des trajectoires, une interruption artificielle de l'exécution du code d'une durée d'une seconde a également été mise en place afin d'attendre la fin de la manœuvre.

6. Source : https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/

7. Les valeurs actuellement inscrites dans le code source correspondent à une ouverture complète de la pince, et à la saisie d'un paquet de mouchoir sur sa tranche la plus fine.

4 Mise en fonctionnement, essais et perspectives

4.1 Mise en fonctionnement et premiers essais



FIGURE 6 – Photographie du robot manipulateur après assemblage



FIGURE 7 – Photographie de la pince après assemblage

Au terme des deux phases de conception - mécanique et logicielle - et de quelques tests "à vide" préliminaires, destinés avant tout à éviter tout contact trop violent entre notre robot et les tables du laboratoire U2IS, nous avons pu assembler notre robot sous sa forme la plus aboutie - figures 6 et 7 - et le confronter à quelques essais.

Lors de ces essais, le robot devait réaliser la suite de tâches suivantes : se déplacer dans sa configuration de référence, se déplacer jusqu'à sa configuration objectif, ouvrir la pince, fermer la pince sur l'objet à attraper, attendre un délai de 5 secondes, lâcher l'objet en ouvrant la pince, et finalement retourner dans sa configuration de référence.

Du point de vue purement logiciel, ces essais ont été une vraie réussite : les trajectoires ont été correctement planifiées et simulées par l'API *Moveit!*, et les consignes - position ou trajectoire - ont bien été transmises au nœud *dynamixel_workbench*, déclenchant l'exécution des trajectoires et les manoeuvres de la pince en respectant les délai d'attente programmés.

En revanche, du point de vue mécanique, le bilan est un peu plus mitigé. Malgré un assemblage réalisé sans encombre (en témoigne la photographie de la figure 6), et un outil parfaitement fonctionnel (et très satisfaisant visuellement, même sur la figure 7), les essais ont avant tout mis en exergue un réel problème de dimensionnement du robot manipulateur et de ses actionneurs, ainsi que la présence d'un jeu très important dans les pièces de liaison "hybrides".

En effet, dès la première mise en tension de notre robot, nous nous sommes rapidement rendus compte que les moteurs situés à la base du robot manipulateur, notamment celui du deuxième axe, n'avaient pas la puissance nécessaire pour supporter le reste du bras articulé. En conséquence, au lieu de se comporter comme un robot parfaitement rigide et de former un angle droit dans sa configuration de référence - cf. figure 2 - notre robot peine à soulever la pince, et oscille dangereusement.

De plus, après quelques mouvements difficilement réalisés, nous avons également constaté que les pièces de liaison "hybrides" que nous avions fabriquées n'étaient pas du tout rigides, et présentaient même du jeu. Ces mouvements internes, comme nous pouvons l'imaginer, ont accentué d'autant plus les phénomènes de flambement et d'oscillation que nous avions déjà observés.

En définitive, le flambement et les oscillations, dus principalement à un défaut de puissance et un manque de rigidité, rendent l'utilisation de notre robot quasiment impossible. Les positions demandées ne sont pas atteintes, les mouvements sont réalisés de manière très hasardeuse, voire même dangereuse lorsque la pince se rapproche de la table, et la plupart du temps, les servo-moteurs finissent par décrocher au bout de quelques cycles.

Ainsi, même si notre robot parvient à attraper un paquet de mouchoir et à le maintenir quelques secondes en l'air, il n'est malheureusement pas encore opérationnel.

Pour les personnes curieuses ou assez courageuses pour reproduire ces essais, un *manuel d'utilisation* sous forme de fichier README est disponible dans l'archive du méta-package ROS fournie avec ce compte-rendu. Cette archive contient également l'intégralité des fichiers de CAO, ainsi que l'ensemble des fichiers source C++ (.cpp), de configuration (.yaml) et de lancement (.launch) rédigés dans le cadre de ce projet.

4.2 Perspectives d'amélioration

4.2.1 Côté conception mécanique

Il s'agit ici des pistes d'amélioration "prioritaires" dans le sens où, la plupart du temps, il s'agit des facteurs qui ont causé l'échec des essais réalisés.

- Revoir le dimensionnement des moteurs du robot manipulateur, en particulier pour ceux situés au niveau de la base - trois premiers axes. Ce dimensionnement doit cette fois être réalisé en prenant en compte les couples nominaux des moteurs, et non pas leur couples de décrochages, qui ne sont pas du tout représentatifs d'une utilisation normale. Si l'utilisation de moteurs plus puissants n'est pas envisageable, il faudrait alors considérer une réduction du poids propre du robot manipulateur et de son outil : utilisation de pièces plus légères, réduction du nombre d'axes,...
- Utiliser un servo-moteur plus petit pour l'actionnement de la pince. Le AX-12A actuellement en place est clairement sur-dimensionné pour la tâche qui lui est confiée. Le remplacer par un autre servo-moteur moins puissant, donc moins lourd et moins encombrant permettrait de gagner quelques précieux grammes ;
- Revoir la conception des pièces de liaison "hybrides", et les remplacer par des pièces d'un seul tenant afin de minimiser la présence de jeu interne. Il serait par exemple intéressant d'investir dans des pièces de type FP04-F53, qui sont compatibles avec les AX-12A, mais ne sont pas fournies dans le kit Bioloid.
- Re-imprimer la pièce constituant la base de la pince, en faisant attention à ne pas la déformer lors du décollage, et choisir des couples vis-écrou plus adaptés pour assembler les pièces entre elles, et ainsi limiter le jeu lors des manœuvres.

4.2.2 Côté conception logicielle

A l'inverse, ces pistes d'améliorations sont à considérer à plus long terme, et correspondent finalement plus à l'ajout de nouvelles features, qui pourraient rendre notre robot plus modulaire.

- Implémenter une fonctionnalité permettant de détecter l'état "en mouvement" ou "à l'arrêt" du robot manipulateur, afin d'enlever les attentes artificielles et parfois mal dimensionnées après chaque action ;
- Trouver un moyen d'intégrer les consignes de vitesse et d'accélération calculées par *Moveit!* dans les consignes transmises aux servo-moteurs Dynamixel ;
- Remplacer le contrôle en position au niveau du servo-moteur associé à la pince par un contrôle en effort. Ce type de contrôle - aussi pris en charge par le package *dynamixel_workbench* - permettrait ainsi d'éviter la phase de calibration de la position du servo-moteur correspondant à l'état "fermée" ;
- Ajouter un contrôle dynamique du robot manipulateur au moyen d'une manette ;
- Intégrer des capteurs supplémentaires pour détecter automatique la position, l'orientation et les dimensions de l'objet à attraper - Caméras, OptiTrack,...

5 Conclusion

Malgré un (léger) défaut de dimensionnement, le robot manipulateur conçu lors de ce projet nous a permis d'atteindre l'objectif fixé dans l'introduction : le laboratoire U2IS est désormais en possession d'un robot manipulateur compact et (presque) opérationnel, et ce, sans avoir dépensé le moindre centime.

A travers ses résultats mitigés, ce projet nous a permis de mettre en exergue la difficulté de concevoir, à notre échelle, un robot manipulateur 6-axes à la fois *abordable* d'un point de vue financier, et *performant* d'un point de vue opérationnel. Dans notre cas, par exemple, il aurait été possible d'atteindre de meilleures performances en choisissant des moteurs plus puissants, et des éléments de fixation plus résistants, mais ces améliorations seraient venues à un coût non-négligeable.

Il s'agit là d'un réel compromis auquel même les professionnels sont confrontés, à l'image de Poppy, qui a préféré proposer un robot abordable mais avec des performances discutables, et d'Interbotix, qui, à l'inverse, propose des robots plutôt chers, mais vraiment performants. En définitive, il faudra encore attendre quelques années, ou alors devenir extrêmement riche très rapidement, avant de pouvoir demander à un robot manipulateur domestique de nous servir des céréales !

Cependant, au delà de cet objectif initial, ce projet nous a permis de découvrir et d'apprendre à utiliser des outils très puissants, comme *Moveit!*, *DynamixelSDK* ou encore le plugin de conversion *sw_urdf_exporter*, dont les possibilités laissent présager un très grand potentiel dans le domaine de la robotique articulée. Il nous a également permis de nous confronter à l'aspect "mécanique" de la robotique, à travers des étapes de conception, de dimensionnement (avec une réussite discutable) et de fabrication numérique - un aspect peut-être un peu trop passé sous silence cette année.

Pour conclure ce rapport sur une note constructive, je tiens à souligner que ce cours-projet était réellement une **réussite**, et mérite **sincèrement** de prendre une place plus importante dans le parcours Robotique dans les années à venir. Quel dommage de devoir attendre les dernières semaines de cours pour enfin mettre les mains dans le cambouis !