



CentraleSupélec

Projet Long

RAPPORT D'ACTIVITÉ DE LA SÉQUENCE 8

Karoline CARVALHO BÜRGER
Tiago DE JESUS RODRIGUES
Rafael ELLER CRUZ
Rafael Accácio NOGUEIRA

Orienté par
Mme. MAKAROV

14 juin 2017

Table des matières

1	Introduction	3
2	Objectif	3
3	Division du Travail	4
3.1	Commande Angulaire et Cartésienne	5
3.2	Trajectoires	5
3.3	Communication avec le robot	6
3.4	Interface Homme ↔ Machine	6
3.5	Désignation des tâches	7
3.6	Calendrier du projet	7
3.7	Suivi du calendrier du projet	8
4	Méthodologie	9
4.1	Commande Angulaire et Cartésienne	9
4.1.1	Commande Angulaire	9
4.1.2	Commande Opérationnelle	12
4.2	Trajectoires	17
4.2.1	Choix de conception et Mise en Œuvre	18
4.2.2	Trajectoire Linéaire	18
4.2.3	Trajectoire Circulaire	20
4.3	Communication avec le robot	22
4.3.1	Études initiaux	22
4.3.2	Premiers Tests	22
4.3.3	Tests du you_Bot	22
4.4	Interface Homme ↔ Machine	23
5	Résultats et Discussions	24
5.1	Commande Angulaire et Cartésienne	24
5.1.1	Commande Angulaire	24
5.1.2	Commande Cartésienne	29
5.2	Trajectoires	40
5.3	Communication avec le robot	42
5.3.1	Premiers Tests	42
5.3.2	Tests du you_Bot	42
5.4	Interface Homme ↔ Machine	43
5.4.1	Première fenêtre : Paramètres du tableau	43
5.4.2	Deuxième fenêtre : Utilisateurs	43
5.4.3	Troisième fenêtre : Jeu de Morpion	44
6	Conclusion	46
	Appendices	47
A	Tutoriels you_Bot / Matlab	47
A.1	Installation you_Bot	47
A.2	Configuration Matlab/ROS/you_Bot	48
A.3	Configuration Simulink/ROS/you_Bot	48

1 Introduction

Vu de l'extérieur, un manipulateur robotique polyvalent pourrait sembler être une technologie plutôt inoffensive, pas très différent d'un tapis roulant ou d'une perceuse, mais l'aspect innovant de ce type de machines est précisément sa polyvalence. Étant reprogrammable, le manipulateur n'est pas limité à une utilisation particulière, ce qui veut dire que le fabricant n'a pas besoin de traiter des considérations particulières d'une industrie donnée, mais seulement avec des aspects généraux (vitesse maximale, couple maximal, déplacement maximum, etc). Le but de ce travail est précisément d'explorer cet aspect polyvalent d'un manipulateur en proposant un système qui permet à l'utilisateur de commander la position de l'extrémité du manipulateur et ses trajectoires. On proposera une application simple pour ce système, dont le but est de démontrer son fonctionnement et évaluer ses performances. Le système complet est schématisé sous la forme simplifiée ci-dessous :

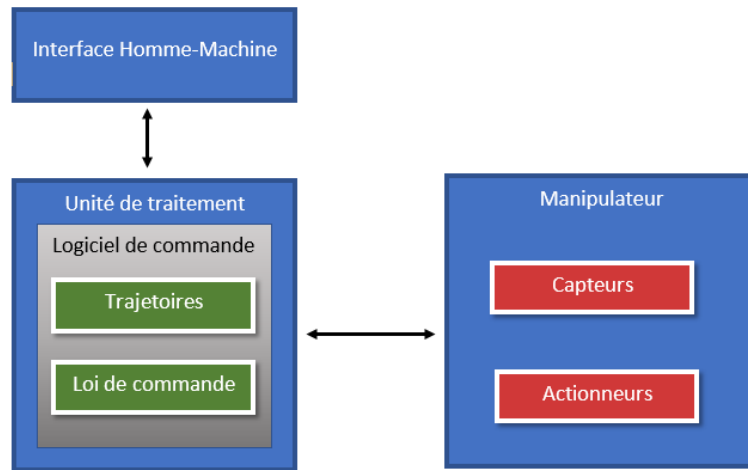


Figure 1 – Schéma simplifié du système.

2 Objectif

L'objectif de notre projet est de concevoir un système de commande pour le robot KUKA youbot qui permet de placer son outil dans n'importe quelle position fixée en avance, accessible pour le robot avec une orientation faisable, en suivant une trajectoire, soit linéaire, soit circulaire. Les coordonnées peuvent être fournies en position angulaire des articulations du robot (coordonnées angulaires) ou en position et orientation de l'outil du robot par rapport à sa base (coordonnées opérationnelles).

On développera aussi une application qui fait usage de ce système de commande pour permettre aux joueurs de jouer le Jeu du Morpion avec une interface graphique intégré à la commande le robot.

Ce rapport a pour but de montrer la planification de nos activités pour les séquences 7 et 8 et la façon dont on a réalisé chaque tâche. Il contient aussi une analyse des résultats obtenus.

3 Division du Travail

On souhaite que le produit final de ce travail soit un jeu du Morpion déployé dans le robot KUKA youBot et commandé par une interface graphique, comme on a déjà décrit ci-dessus.

Sur la base de ces objectifs, on peut séparer le travail en deux phases différentes que seront développées de manière séquentielle :

- Phase 1 : à propos du système robotique dans un environnement de simulation, lequel on doit traiter et résoudre les problèmes de commande du système ;
- Phase 2 : à propos de la mise en œuvre du jeu en environnement physique, pour lequel on doit construire les différentes interfaces du système : robot \leftrightarrow PC et PC \leftrightarrow utilisateur.

Les actions et les sous-produits de la première phase du projet seront séparés en deux structures et développés en parallèle : la première structure concernant l'étude et la génération de la trajectoire exécutée par l'outil du robot (structure A) et la deuxième structure concernant l'étude et la mise en œuvre de l'asservissement du système (structure B). Après la conclusion de chaque structure, il faut faire l'intégration des parties qui ont été développées. De cette façon, on prévoit à la fin de la phase 1 d'observer, dans l'environnement de simulation Matlab, l'outil réaliser une trajectoire stipulée.

De la même manière, la deuxième phase sera séparée en deux structures : la première structure concernant la programmation de l'environnement du robot en ROS et l'intégration avec la commande (structure A) et la deuxième structure concernant la création de l'interface graphique de communication de l'ordinateur avec l'utilisateur (structure B).

En plus de cette répartition du projet en deux phases bien définies, on a pu séparer le projet en deux différents domaines : automatique et informatique. Le diagramme en arbre suivant décrit les différentes phases et structures dont on a réparti le projet :

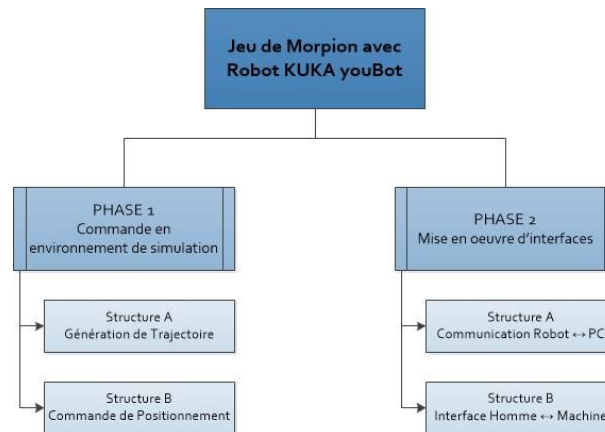


Figure 2 – Diagramme en arbre de la structure de découpage du projet

Ci-dessous, on a défini les tâches à entreprendre en chaque structure de découpage du projet. Dans la section 4 on a décrit les tâches de façon plus détaillée.

3.1 Commande Angulaire et Cartésienne

Cette partie du projet a été la première où on a travaillé, car pour commencer un projet on doit premièrement étudier le sujet. Cela correspond aussi à la première étape du développement d'une loi de commande : étudier et modéliser le système. Par contre, le modèle du robot nous a été fourni dans un fichier Matlab et donc on a employé du temps pour le comprendre.

L'étape initiale d'étude des systèmes robotiques a été la première tâche de tous les membres de l'équipe, une fois que le but du projet est d'apprendre comment les systèmes robotiques fonctionnent et comment les contrôler. De plus, on avait besoin de connaître les spécificités de ce type de système et tous les outils qu'on avait disponibles pour définir une application final qui était faisable dans le temps disponible.

On a réservé initialement 28 créneaux pour les tâches relationnée à la commande du robot. Par contre, des problèmes de performance des lois de commande nous ont emmenés à consacrer plus de temps dans cette partie du projet. Dans la section 3.7 on a le calendrier du déroulement réel du projet.

Dans la suite, la division de cette partie du travail en tâches est décrite, ainsi que la quantité de créneaux qu'on avait l'intention de consacrer à chaque tâche.

- 1.B.1 : Étude à propos de la modélisation des systèmes robotiques : 6 créneaux ;
- 1.B.2 : Compréhension du modèle Matlab du robot fourni : 2 créneaux
- 1.B.3 : Étude et développement des techniques de commande du robot dans l'espace des articulations et choix d'une méthode de commande : 4 créneaux ;
- 1.B.4 : Mis en oeuvre de la loi de commande du robot dans l'espace des articulations dans le Simulink et tests : 2 créneaux ;
- 1.B.5 : Étude et développement des techniques de commande du robot dans l'espace opérationnel et choix d'une méthode de commande : 2 créneaux ;
- 1.B.6 : Mis en œuvre de la loi de commande du robot dans l'espace opérationnel dans le Simulink et tests : 4 créneaux ;
- 1.B.7 : Optimisation des lois de commande : 2 créneaux ;
- 1.B.8 : Intégration avec le module de génération des trajectoires : 3 créneaux ;
- 1.B.9 : Intégration avec le robot : 3 créneaux ;

3.2 Trajectoires

Afin d'organiser et simplifier le travail de la création des trajectoires, on a défini deux types de trajectoire : linéaire et arc de cercle. Comme entrée de la fonction, on prend le type de trajectoire, le point initial, un point intermédiaire (pour des arcs de cercle) et le point final. La création de chacun de ces types de trajectoire peut être divisée en plusieurs étapes.

Pour les trajectoires linéaires :

- 1.A.1 : Création d'une trajectoire linéaire entre deux points : 2 créneaux ;
- 1.A.2 : Calcul du polynôme interpolateur : 2 créneaux ;

Pour les trajectoires Circulaires :

- 1.A.3 : Calcul du cercle : 2 créneaux ;
- 1.A.4 : Paramétrisation du cercle : 2 créneaux ;
- 1.A.5 : Calcul du angle du arc : 2 créneaux ;
- 1.A.6 : Calcul du polynôme interpolateur : 2 créneaux ;

3.3 Communication avec le robot

Cette partie concerne l'intégration entre le programme de commande et l'interface du robot Kuka youbot. Donc la première tâche à être accomplie c'est l'étude de cette interface de sorte qu'on puisse définir des tâches et planifier les étapes de cette phase.

- 2.A.0 Étude de l'interface et planification

Après l'étude général du problème nous avons divisé en diverses tâches :

- 2.A.1 Étude du ROS : 3 Créneaux ;
- 2.A.2 Tests turtle_bot utilisant bash/ROS : 3 Créneaux ;
- 2.A.3 Tests turtle_bot utilisant bash/ROS/Matlab : 3 Créneaux ;
- 2.A.4 Tests turtle_bot utilisant bash/Simulink : 3 Créneaux ;
- 2.A.5 Tests you_Bot utilisant bash/ROS : 3 Créneaux ;
- 2.A.6 Tests you_Bot utilisant bash/ROS/Matlab : 3 Créneaux ;
- 2.A.7 Tests you_Bot utilisant bash/ROS/Simulink : 3 Créneaux ;

3.4 Interface Homme ↔ Machine

Comme point de départ du développement de l'IHM, on a l'obtention et l'utilisation d'une fonction concise obtenue en tant que produit final de l'intégration effectuée à la fin de la phase 1 (tâche 1.B.8).

Dans un premier temps, on doit définir les caractéristiques, les contraintes et les dimensions de l'environnement de travail de l'outil (tableau du jeu). Alors, on doit créer les fonctions pour exécuter les deux mouvements que le robot réalisera : "X" et "O". Après, on doit définir les paramètres d'entrée et les données nécessaires pour le développement de l'interface et aussi la création et l'administration des objets graphiques de l'interface. Pour finir, on a besoin de faire des tests, des corrections nécessaires et des améliorations dans l'interface et l'intégration de ses fonctionnalités avec d'autres produits finaux du projet.

Dans la suite, on a ordonné ces tâches et le temps estimé pour l'exécution de chaque activité :

- 2.B.0 : Fonction concise qui détermine le paramètre d'entrée du robot (couple) à partir de la position finale et le type de trajectoire désirée (obtenue dans la tâche 1.B.8) : 6 créneaux ;
- 2.B.1 : Définition de les caractéristiques et dimensions d'environnement de travail (tableau du jeu) : 2 créneaux ;
- 2.B.2 : Définition et création de la structure de données et paramètres nécessaires pour l'IHM : 2 créneaux ;
- 2.B.3 : Création des fonctions de dessin "X" et "O" : 2 créneaux ;
- 2.B.4 : Définition de la disposition graphique et des objets de l'interface : 2 créneaux ;
- 2.B.5 : L'insertion des éléments dans l'interface : 2 créneaux ;
- 2.B.6 : Création et gestion des événements associés à les objets : 2 créneaux ;
- 2.B.7 : Tests, dépannage et intégration : 3 créneaux

On prévoit 6 créneaux pour la réalisation de la tâche 2.B.0 qu'on doit obtenir comme résultat de l'intégration des activités dans la première phase du projet et plus 15 créneaux pour l'exécution des autres tâches de l'interface.

3.5 Désignation des tâches

À partir du découpage et de la définition des différentes structures du projet, on a défini que les membres Rafael Accacio et Rafael Eller seront responsables de l'exécution de la structure A du projet dans les deux phases, tandis que les membres Karoline et Tiago seront responsables de l'exécution la structure B. Pour faire cette désignation des tâches, on a considéré les jours disponibles de projet (série) de chaque membre, afin que dans les jours lesquels seulement deux membres seront au laboratoire, chacun travaille dans des structures différentes. Ainsi, on prévoit obtenir une progression plus rapide et égale pour chaque structure.

Cette organisation n'empêche pas les membres d'aider, discuter et exécuter les différentes activités prévues, en prenant en compte la conclusion des tâches respectives. De plus, à la fin de chaque phase on prévoit l'intégration des structures, ainsi la relation et le partage d'informations entre l'équipe sera considérée pour l'obtention de bons résultats.

3.6 Calendrier du projet

Afin d'analyser et gérer le progrès de l'équipe et l'exécution des différentes tâches, on a fait l'estimation de la durée d'exécution des activités nécessaires pour la conclusion du projet. La table 3 décrit le calendrier de réalisation du projet.

Identifica- teur d'activité	Description de l'activité	Unités (1h30)	Calendrier du projet								
			Séquence 7	Séquence 8							
				S1	S2	S3	S4	S5	S6	S7	S8
1.B.1	Étude : modélisation systèmes robotiques	6									
1.B.2	Compréhension modèle Matlab du robot	2									
1.B.3	Étude: commande espace des articulations	4									
1.B.4	Exécution: commande espace des articulations	2									
1.B.5	Étude: commande espace opérationnel	2									
1.B.6	Exécution: commande espace opérationnel	4									
1.B.7	Optimisation des lois de commande	2									
1.B.8	Intégration avec générateur des trajectoires	3									
1.B.9	Intégration avec le robot	3									
2.B.0	Fonction concise des trajectoires et commande	6									
2.B.1	Définition d'environnement de travail (tableau)	2									
2.B.2	Création des fonctions "X" et "O"	2									
2.B.3	Définition de la disposition graphique d'IHM	2									
2.B.4	Définition de données d'entrée nécessaires IHM	1									
2.B.5	L'insertion des objets graphiques	3									
2.B.6	Création et gestion d'événements	2									
2.B.7	Tests, dépannage et intégration	3									
1.A.1	Trajectoire linéaire entre deux point	2									
1.A.2	Calcul du polynôme interpolateur (cas linéaire)	2									
1.A.3	Calcul du cercle	2									
1.A.4	Paramétrisation le cer de	2									
1.A.5	Calcul du angle du arc	2									
1.A.6	Calcul du polynôme interpolateur (cas cerde)	2									
2.A.0	Étude de l'interface et planification	3									
2.A.1	Étude du ROS	3									
2.A.2	Tests turtle_bot utilisant bash/ROS	3									
2.A.3	Tests turtle_bot utilisant bash/ROS/Matlab	3									
2.A.4	Tests turtle_bot utilisant bash/Simulink	3									
2.A.5	Tests you_Bot utilisant bash/ROS	3									
2.A.6	Tests you_Bot utilisant bash/ROS/Matlab	3									
2.A.7	Tests you_Bot utilisant bash/ROS/Simulink	3									
3.1	Préparation du rapport	7									
3.2	Préparation de la présentation	4									

Figure 3 – Calendrier de réalisation du projet.

Les unités de temps sont considérées en créneaux, où chaque créneau correspond à une durée de 1h30. Pour les jours dans lesquels seulement la moitié de l'équipe est présente dans le laboratoire, on a considéré la moitié du temps de chaque créneaux. On 54 créneaux au total pour la conclusion du projet.

3.7 Suivi du calendrier du projet

En raison des problèmes dans le déroulement de nos activités soit de performance du système, soit de manque de temps, on n'a pas suivi parfaitement le calendrier initiale du projet. On a essayé de l'adapter pour accomplir le plus grand nombre possible de tâches. Dans la table suivante, on a la vraie distribution temporelle de nos activités.

Identificateur d'activité	Description de l'activité	Unités (1h30)	Calendrier du projet								
			Séquence 7	Séquence 8							
				S1	S2	S3	S4	S5	S6	S7	S8
1.B.1	Étude: modélisation systèmes robotiques	6									
1.B.2	Compréhension modèle Matlab du robot	2									
1.B.3	Étude: commande espace des articulations	4									
1.B.4	Exécution: commande espace des articulations	2									
1.B.5	Étude: commande espace opérationnel	23									
1.B.6	Exécution: commande espace opérationnel	27									
1.B.7	Optimisation des lois de commande	21									
1.B.8	Intégration avec générateur des trajectoires	2									
2.B.0	Fonction concise des trajectoires et commande	6									
2.B.1	Définition d'environnement de travail (tableau)	2									
2.B.2	Création des fonctions "X", "O" et déplacement	2									
2.B.3	Définition de la disposition graphique d'IHM	2									
2.B.4	Définition de données d'entrée nécessaires IHM	1									
2.B.5	L'insertion des objets graphiques	3									
2.B.6	Création et gestion d'événements	2									
2.B.7	Tests, dépannage et intégration	3									
1.A.1	Trajectoire linéaire entre deux point	2									
1.A.2	Calcul du polynôme interpolateur (cas linéaire)	2									
1.A.3	Calcul du cercle	2									
1.A.4	Paramétrisation le cercle	2									
1.A.5	Calcul du angle du arc	3									
1.A.6	Calcul du polynôme interpolateur (cas cercle)	3									
2.A.0	Étude de l'interface et planification	6									
2.A.1	Étude du ROS	4									
2.A.2	Tests turtle_bot utilisant bash/ROS	3									
2.A.3	Tests turtle_bot utilisant bash/ROS/Matlab	3									
2.A.4	Tests turtle_bot utilisant bash/Simulink	3									
2.A.5	Tests you_Bot utilisant bash/ROS	3									
2.A.6	Tests you_Bot utilisant bash/ROS/Matlab	3									
2.A.7	Tests you_Bot utilisant bash/ROS/Simulink	3									
3.1	Préparation du rapport	7									
3.2	Préparation de la présentation	4									

Figure 4 – Calendrier de suivi du projet.

À propos de la commande, on a beaucoup changé le calendrier original parce qu'on a resté longtemps en train d'essayer plusieurs types différents de contrôleurs jusqu'à avoir un avec des performances satisfaisantes.

4 Méthodologie

4.1 Commande Angulaire et Cartésienne

Pour développer les lois de commande pour le robot, on est parti de son modèle dynamique. L'idée principale est de contrôler la configuration angulaire du robot par le couple mécanique développé dans chaque articulation. Alors, la relation entre ces deux grandeurs est définie par le modèle dynamique direct :

$$\mathbf{\Gamma} = \mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) \quad (1)$$

Où, pour un robot à n degrés de liberté (ddl), $\mathbf{\Gamma} \in \mathbb{R}^n$ est le vecteur des couples appliqués dans chaque articulation, $\mathbf{q} \in \mathbb{R}^n$ est le vecteur des positions angulaires de chaque articulation, $\mathbf{A}(\mathbf{q}) \in \mathbb{R}^{n \times n}$ est la matrice d'inertie et $\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^n$ représente les couples dissipatifs qui agissent dans le robot (Coriolis, frottements et gravité).

Les positions angulaires de chaque articulation (\mathbf{q}) définissent aussi la position et l'orientation de l'outil du robot. Il y a une relation entre eux, qui s'appelle modèle géométrique direct (MGD). Ce modèle est basé sur des changements de repère : on définit un repère dans chaque articulation du robot, un dans la base et un autre dans l'outil. L'idée est de créer une matrice homogène de transformation de repère (position et direction) d'une axe par rapport à l'axe précédente, qui dépend des paramètres constructifs du robot et des angles des articulations. En multipliant ces matrices, on peut obtenir la position et orientation de l'outil par rapport au repère dans la base du robot comme fonction des angles des articulations (\mathbf{q}) [8].

Sur la base de ces équations, on a développé dans ce projet les lois de commande dans l'espace des articulations bien comme dans l'espace opérationnel.

4.1.1 Commande Angulaire

Il existe plusieurs techniques de commande pour les robots dans l'espace angulaire, soit par une approche fréquentielle (basée sur les fonctions de transfert du système), soit par une approche temporelle (basée sur la représentation d'états du système). Dans ce projet, on n'a étudié que l'approche fréquentielle pour développer les lois de commande, parce qu'elle est plus simple de comprendre et mettre en œuvre et elle fournit des résultats satisfaisants. Parmi ces méthodes fréquentielles, on a choisi la commande des articulations avec un PID (*Proportional Integral Derivative*) pour utiliser dans notre application parce que les performances obtenues avec cette commande étaient acceptables. De plus, comme les calculs nécessaires pour cette commande sont simples et le contrôleur a été développé numériquement, il permet au robot de suivre des consignes en temps réel, sans aucun délai pour faire des calculs. On a mis en œuvre aussi la commande angulaire par couple calculé, qui nous a fourni des performances encore mieux que les performances de la commande PID. Par contre, la commande par couple calculé est plus complexe, avec beaucoup de paramètres qui dépendent du modèle du robot, et de cette façon le temps de calcul est considérable et peut réduire la performance du système en ce qui concerne le suivi des consignes en temps réel.

En fait, la commande PID est mieux pour notre application du Jeu du Morpion en raison du temps d'exécution du programme. Par contre, du point de vue de la commande, les deux sont satisfaisants. Dans la suite, on explique en plus de détail

le concept de chaque méthode.

Commande angulaire PID

Cette méthode de commande s'agit de considérer que chaque articulation du robot est découplée des autres, c'est à dire que, pour déterminer la dynamique de chaque articulation, on va négliger l'influence de la configuration des autres articulations. De cette façon, la matrice $\mathbf{A}(\mathbf{q})$ devient une matrice diagonale et donc on peut réécrire l'équation 1 pour chaque articulation comme :

$$\Gamma_i = a_i \ddot{q}_i + F_{v_i} \dot{q}_i + \gamma_i \quad (2)$$

Où a_i est la magnitude maximale de l'élément (i,i) de la matrice d'inertie, F_{v_i} est le coefficient de frottement visqueux par rapport à l'articulation i et γ_i est une perturbation (due, par exemple, au couple gravitationnel et aux effets des autres axes sur l'axe i) [8].

Si on connaît le modèle dynamique du robot, on peut trouver les paramètres de l'équation 2 et développer une loi de commande pour chaque axe. On a utilisé un contrôleur PID pour générer les couples dans chaque articulation. Soit $q_{i_{ref}}(t)$ la consigne angulaire pour l'axe i du robot, $q_i(t)$ la position angulaire de l'axe i dans l'instant t et on dénote $\epsilon_i(t) = q_{i_{ref}}(t) - q_i(t)$ l'erreur de position pour l'axe i par rapport à la consigne. La fonction de transfert de notre contrôleur est :

$$C_i(p) = \frac{\Gamma_i(p)}{\epsilon_i(p)} = K_{p_i} + K_{d_i}p + \frac{K_{i_i}}{p} \quad (3)$$

De cette façon, on détermine le schéma bloc de commande angulaire PID pour le robot, qui est montré dans la figure 5. Sur la base de ce schéma bloc, on peut obtenir la fonction de transfert du système en boucle fermée pour chaque axe, montrée dans l'équation 4.

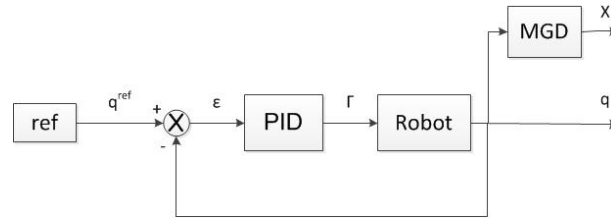


Figure 5 – Schéma bloc du système commandé dans l'espace angulaire par un PID

$$H_i(p) = \frac{q_i(p)}{q_{i_{ref}}(p)} = \frac{K_{d_i}p^2 + K_{p_i}p + K_{i_i}}{a_i p^3 + (K_{d_i} + F_{v_i})p^2 + K_{p_i}p + K_{i_i}} \quad (4)$$

À partir de la fonction de transfert en boucle fermée, on trouve que l'équation caractéristique du système commandé dans l'espace angulaire par un PID est $d(p) = a_i p^3 + (K_{d_i} + F_{v_i})p^2 + K_{p_i}p + K_{i_i}$. On peut choisir, pour chaque axe, K_{p_i} , K_{d_i} et K_{i_i} pour fixer la dynamique du robot en boucle fermée [9]. On veut que le dépassement de la réponse indicielle du système en boucle fermée soit nul, alors on a besoin que $d(p) = a_i(p + \omega_i)^3$, où ω_i est la pulsation de brisure en boucle fermée, choisi de façon à contrôler le temps de réponse du système. Cependant,

il y a un compromis dans le choix des ω_i , parce que s'ils sont petits, le système sera trop lent, mais s'ils sont trop importants, on peut arriver à la fréquence de résonance du système due aux élasticités non modélisés du robot, et cela peut générer des signaux de commande d'amplitude très important et aussi très bruité en présence de bruit dans le système. Un bon compromis est de choisir ω_i comme la moitié de la pulsation de résonance de l'axe i . Alors, une fois que ω_i soit défini, on peut calculer les gains K_{p_i} , K_{d_i} et K_{i_i} du PID selon les formules suivantes :

$$K_{p_i} = 3a_i(\omega_i)^2 \quad (5a)$$

$$K_{d_i} = 3a_i(\omega_i) - F_{v_i} \quad (5b)$$

$$K_{i_i} = a_i(\omega_i)^3 \quad (5c)$$

Les matrices de gain du PID \mathbf{K}_p , \mathbf{K}_d , $\mathbf{K}_i \in \mathbb{R}^{n \times n}$ sont des matrices diagonales, où les éléments non nulles (i, i) sont respectivement K_{p_i} , K_{d_i} et K_{i_i} .

Commande angulaire par couple calculé

La commande par couple calculé s'agit de compensé les non-linéarités du modèle dynamique du robot par l'estimation de la matrice d'inertie $\hat{\mathbf{A}}(\mathbf{q}) \in \mathbb{R}^{n \times n}$ et du vecteur des couples dissipatifs $\hat{\mathbf{H}}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^n$. Alors, on utilisera ces paramètres pour calculé les couples selon la relation suivante, où $\mathbf{w}(t) \in \mathbb{R}^n$ est une variable de commande :

$$\mathbf{\Gamma} = \hat{\mathbf{A}}(\mathbf{q}) * \mathbf{w} + \hat{\mathbf{H}}(\mathbf{q}, \dot{\mathbf{q}}) \quad (6)$$

Si $\hat{\mathbf{A}}$ et $\hat{\mathbf{H}}$ modélisent bien la dynamique du robot et on a $\mathbf{w}(t) = \ddot{\mathbf{q}}(t)$, cela correspond exactement au modèle dynamique direct. Avec cette loi de commande, on peut calculer le couple nécessaire pour mettre le robot dans la configuration angulaire qu'on veut avant de l'alimenter dans le robot.

Si on utilise une loi de commande PID pour obtenir $\mathbf{w}(t) = \ddot{\mathbf{q}}(t)$ à partir de l'erreur de position angulaire, qui est défini comme dans le cas précédent ($\epsilon(t) = \mathbf{q}_{ref}(t) - \mathbf{q}(t)$), et étant \mathbf{K}_p , \mathbf{K}_d et $\mathbf{K}_i \in \mathbb{R}^{n \times n}$ les matrices diagonales des gains du PID, on obtient :

$$\mathbf{w}(t) = \ddot{\mathbf{q}}_{ref}(t) + \mathbf{K}_p(\mathbf{q}_{ref}(t) - \mathbf{q}(t)) + \mathbf{K}_d(\dot{\mathbf{q}}_{ref}(t) - \dot{\mathbf{q}}(t)) + \mathbf{K}_i \int_0^t (\mathbf{q}_{ref}(\tau) - \mathbf{q}(\tau)) d\tau \quad (7)$$

On peut développer cette équation dans le domaine de Laplace en fonction de l'erreur de position, sachant que $\mathbf{w}(t) = \ddot{\mathbf{q}}(t)$:

$$(\ddot{\mathbf{q}}_{ref}(t) - \ddot{\mathbf{q}}(t)) + \mathbf{K}_p(\mathbf{q}_{ref}(t) - \mathbf{q}(t)) + \mathbf{K}_d(\dot{\mathbf{q}}_{ref}(t) - \dot{\mathbf{q}}(t)) + \mathbf{K}_i \int_0^t (\mathbf{q}_{ref}(\tau) - \mathbf{q}(\tau)) d\tau = 0$$

$$\ddot{\epsilon}(t) + \mathbf{K}_p\epsilon(t) + \mathbf{K}_d\dot{\epsilon}(t) + \mathbf{K}_i \int_0^t \epsilon(\tau) d\tau = 0$$

$$p^3 \mathbf{E}(p) + p^2 \mathbf{K}_d \mathbf{E}(p) + p \mathbf{K}_p \mathbf{E}(p) + \mathbf{K}_i \mathbf{E}(p) = p^2 \epsilon(0) + p \dot{\epsilon}(0) + \epsilon(0)$$

Alors, cette méthode nous a permis de linéariser l'équation de la loi de commande. Cela rend plus facile la détermination des gains du PID pour avoir les performances

désirées en boucle fermée. On peut écrire l'équation de l'erreur dans le domaine de Laplace pour chaque axe, ce qui nous permet de déterminer la dynamique du système en boucle fermée [9] :

$$E_i(p) = \frac{p^2 \epsilon_i(0) + p(\epsilon_i(0) + \dot{\epsilon}_i(0))}{p^3 + K_{d_i} p^2 + K_{p_i} p + K_{i_i}} \quad (9)$$

Comme il a été dit, on veut que le dépassement de la réponse indicielle soit nulle, et pour une pulsation de brisure en boucle fermée ω_i , on a :

$$K_{p_i} = 3\omega_i^2 \quad (10a)$$

$$K_{d_i} = 3\omega_i \quad (10b)$$

$$K_{i_i} = \omega_i^3 \quad (10c)$$

Dans la figure suivante, on a le schéma bloc de cette méthode de commande angulaire.

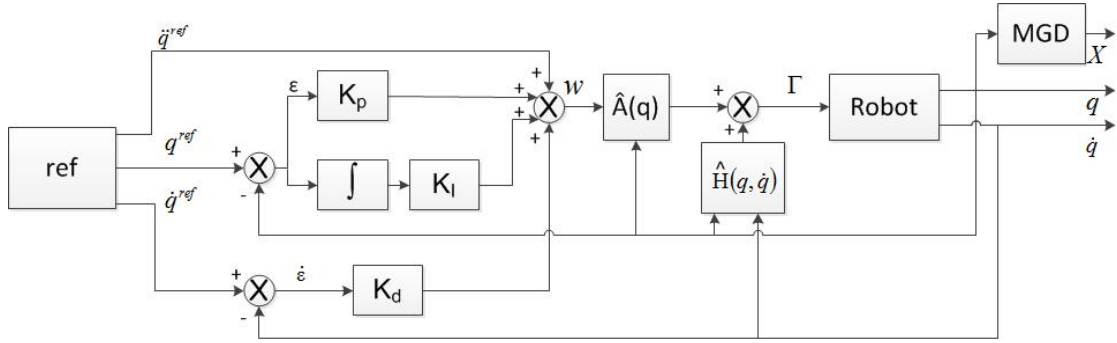


Figure 6 – Schéma bloc du système commandé en espace angulaire par couple calculé

4.1.2 Commande Opérationnelle

Comme pour la commande dans l'espace articulaire, il existe déjà plusieurs façon de faire la commande dans l'espace opérationnel, soit pour une approche fréquentielle, soit pour une approche temporelle. Pour question de simplicité, on a décidé d'étudier juste les approches fréquentielles. Parmi tous les méthodes fréquentielles, on étudié quatre : la commande PID opérationnelle, la commande opérationnelle par couple calculé, la commande cascade avec boucle interne de vitesse angulaire et la commande par le modèle géométrique inverse (MGI).

Dans la suite, on va présenter juste les deux derniers méthodes, avec lesquelles on a obtenu des performances les plus satisfaisantes. La commande PID opérationnelle est la plus simple, mais les lois de commande qu'on obtient sont non linéaires, ce qui rend très difficile de régler les gain du PID pour avoir des performances désirées, alors on l'a supprimé. Pour la commande opérationnelle par couple calculé, on a besoin de calculer des paramètres qui sont très instables et bruités, comme la dérivé et la pseudo-inverse de la matrice Jacobien du système, qui sera expliquée dans la suite. Ces opérations, en plus de coûteuses en ressources informatiques, apportent des imprécisions à la réponse du système et affectent ses performances, alors on l'a supprimé aussi.

À propos des autres deux méthodes de commande, ils sont équivalents par rapport aux performances et au temps de calcul numérique. Par contre, la commande

cascade avec boucle interne de vitesse angulaire est plus robuste, exactement pour utiliser deux boucle de commande : une boucle externe pour la commande de position de l'outil et une boucle interne pour la commande en vitesse angulaire du robot. C'est la raison pour laquelle on a choisi d'utiliser cette commande dans notre application.

Commande cascade avec boucle interne de vitesse angulaire

Comme il a déjà été dit, dans cette méthode on va contrôler le robot dans l'espace opérationnel en utilisant deux boucles de commande : la boucle interne sert à commander le robot en vitesse angulaire ($\dot{\mathbf{q}}$), et la boucle externe sert à commander la position et l'orientation de l'outil du robot [7].

Pour comprendre la boucle externe, on imagine que la commande de la boucle interne est bien réglée, c'est à dire que si la boucle interne reçoit comme consigne $\dot{\mathbf{q}}_{ref}$, on aura comme sortie $\dot{\mathbf{q}} = \dot{\mathbf{q}}_{ref}$. De plus, quand on fait la commande opérationnelle du robot, il faut souvent convertir les variables de l'espace opérationnel à l'espace angulaire. Pour faire cette conversion, on utilise le Jacobien. Le Jacobien est une matrice formée par les dérivées partielles du MGD par rapport à chaque articulation du robot. Il est défini par la relation :

$$\mathbb{V}_o = \mathbf{J}_{tot}(\mathbf{q})\dot{\mathbf{q}} \quad (11)$$

$\mathbb{V}_o = (V_{x_o}, V_{y_o}, V_{z_o}, \omega_{x_o}, \omega_{y_o}, \omega_{z_o})^T$ est le torseur cinématique, où les trois premiers termes représentent la vitesse en X, Y et Z de l'outil par rapport au repère dans la base du robot et les trois derniers termes représentent la vitesse de rotation du repère défini dans l'outil par rapport au repère dans la base du robot, autour de X, Y et Z. Pour un robot à n degrés de liberté, $\mathbf{J}_{tot}(\mathbf{q}) \in \mathbb{R}^{6 \times n}$ est le Jacobien. On peut simplifier cette expression si on prend la vitesse de l'outil $\dot{\mathbf{X}} = (V_{x_o}, V_{y_o}, V_{z_o})^T$ au lieu du torseur cinématique, et le Jacobien devient $\mathbf{J} = \mathbf{J}_{tot}(1 : 3, :)$, $\mathbf{J} \in \mathbb{R}^{3 \times n}$. Alors, on peut écrire les équations :

$$\dot{\mathbf{X}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \quad (12a)$$

$$\dot{\mathbf{q}} = \mathbf{J}^*(\mathbf{q})\dot{\mathbf{X}} \quad (12b)$$

Où $\mathbf{J}^* = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}$ est la pseudo inverse du Jacobien.

Soit \mathbf{X}_{ref} et $\dot{\mathbf{X}}_{ref}$ les références de position et vitesse respectivement pour l'outil du robot, $\boldsymbol{\epsilon}_X = \mathbf{X}_{ref} - \mathbf{X}$ l'erreur de position de l'outil, K un gain et $\boldsymbol{\omega}_{ref}$ la référence de vitesse angulaire qui sera fournie à la boucle interne, on définit la loi de commande de la boucle externe :

$$\boldsymbol{\omega}_{ref} = \mathbf{J}^*(\dot{\mathbf{X}}_{ref} + K\boldsymbol{\epsilon}) \quad (13)$$

Le schéma bloc de la boucle externe de cette méthode de commande est montré dans la figure 7.

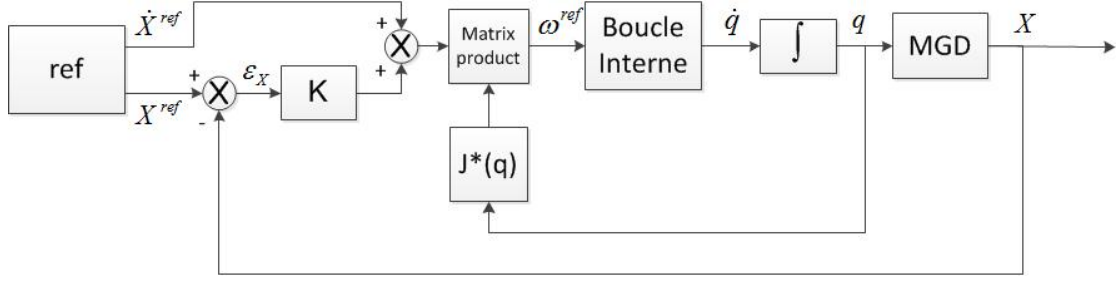


Figure 7 – Schéma bloc de la boucle externe du commande cascade

Si on a une boucle interne bien réglée ($\dot{q} = \omega_{ref}$), on multiplie l'équation 13 par J et on utilise la relation 12a, on obtient :

$$\dot{\epsilon}_X = -K\epsilon_X \quad (14)$$

Alors, on peut voir dans la relation 14 que l'erreur tend à zéro de façon exponentielle avec une constant de temps K qui nous permet de contrôler la dynamique du système. Mais pour que la boucle externe fonctionne, on doit bien régler la boucle de vitesse angulaire.

Pour faire la commande de la boucle interne, on utilise la méthode du couple calculé, comme décrite dans la section 4.1.1, mais au lieu de fixer des consignes de position angulaire, on fixe des consignes de vitesse angulaire. On définit la loi de commande dans l'équation suivant.

$$\Gamma = \hat{A}(q) * w + \hat{H}(q, \dot{q}) \quad (15a)$$

$$w = \dot{\omega}_{ref} + K_p \epsilon_{\dot{q}} + K_i \int_0^t \epsilon_{\dot{q}} d\tau = \ddot{q} \quad (15b)$$

Où $\epsilon_{\dot{q}} = \omega_{ref} - \dot{q}$ est l'erreur de vitesse angulaire du robot et K_p et K_i sont deux gain utilisés pour régler la dynamique de la boucle de vitesse, équivalents à une commande PI (*Proportional Integral*).

Le schéma bloc de la boucle interne de cette méthode de commande est montré dans la figure 8.

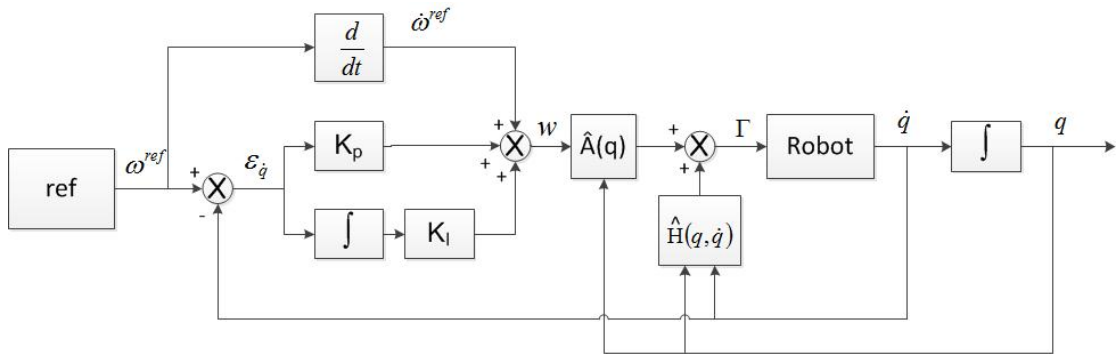


Figure 8 – Schéma bloc de la boucle interne du commande cascade

Commande opérationnelle par le modèle géométrique inverse (MGI)

Le MGI sert à convertir une configuration (position plus orientation) faisable de l'outil du robot à des angles des articulations qui laissent l'outil dans cette

configuration. Si on a le MDI, on peut faire la commande du robot dans l'espace opérationnel en utilisant le schéma de commande angulaire (comme par exemple, celle de la figure 5), parce qu'on convertit les consignes de position aux consignes angulaires. Le problème est que générer le MDI n'est pas facile, et parfois il y a plus d'une configuration angulaire qui conduit à la configuration de l'outil qu'on veut.

Pour développer le MDI, on s'est basé sur la méthode *Yasukawa Motomart L-3* [6]. Premièrement, on doit écrire les matrices homogènes de changement de repère d'une axe du robot à l'axe suivante [8]. Soit $\mathbf{T}_{i,j}$ la matrice de changement du repère de l'axe i au repère de l'axe j (end représente le repère de l'outil du robot), d_i la distance du repère i au repère $(i-1)$ par rapport à l'axe X du repère $(i-1)$, r_i la distance du repère i au repère $(i-1)$ par rapport à l'axe Z du repère $(i-1)$ et θ_i est la position angulaire de l'axe i , on trouve les matrices de changement de repère selon le diagramme physique du robot [2] :

$$\begin{aligned} \mathbf{T}_{0,1} &= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & d_1 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & r_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \mathbf{T}_{3,4} &= \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & d_4 \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{T}_{1,2} &= \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & d_2 \\ 0 & 0 & -1 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \mathbf{T}_{4,5} &= \begin{bmatrix} \cos(\theta_5) & -\sin(\theta_5) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(\theta_5) & -\cos(\theta_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{T}_{2,3} &= \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & d_3 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \mathbf{T}_{5,end} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_{end} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

On sait que, pour obtenir la matrice de changement du repère i au repère k étant données $\mathbf{T}_{i,j}$ et $\mathbf{T}_{j,k}$, il suffit de faire $\mathbf{T}_{i,k} = \mathbf{T}_{i,j} * \mathbf{T}_{j,k}$. Alors, avec les matrices de changement de repère du robot, on peut trouver la relation suivante :

$$\mathbf{T}_{1,5} = \mathbf{T}_{0,1}^{-1} * \mathbf{T}_{0,5} = \mathbf{T}_{1,2} * \mathbf{T}_{2,3} * \mathbf{T}_{3,4} * \mathbf{T}_{4,5} \quad (16)$$

Soit $\mathbf{T}_{0,5}$ la configuration du repère de l'axe 5 par rapport à la base du robot, qu'on peut trouver à partir de la configuration désirée pour l'outil $\mathbf{T}_{0,end}$ avec la relation $\mathbf{T}_{0,5} = \mathbf{T}_{0,end} * \mathbf{T}_{5,end}^{-1}$, définie par :

$$\mathbf{T}_{0,5} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & p_x \\ r_{2,1} & r_{2,2} & r_{2,3} & p_y \\ r_{3,1} & r_{3,2} & r_{3,3} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (17)$$

Alors, on obtient de (16) :

$$\begin{bmatrix} * & * & r_{1,3}c1 + r_{1,3}s1 & * \\ r_{2,1}c1 - r_{1,1}s1 & r_{2,2}c1 - r_{1,2}s1 & * & k_1 \\ * & * & r_{3,3} & * \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} * & * & -s234 & * \\ s5 & c5 & * & 0 \\ * & * & c234 & * \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (18)$$

Où $c1 = \cos(\theta_1)$, $s1 = \sin(\theta_1)$, $s234 = \sin(\theta_2 + \theta_3 + \theta_4)$, $c234 = \cos(\theta_2 + \theta_3 + \theta_4)$, $c5 = \cos(\theta_5)$, $s5 = \sin(\theta_5)$, et $k_1 = p_y c1 - (p_x - d_1)s1$.

Pour comparaison des deux matrices dans la relation précédente, on trouve déjà que :

$$\theta_1 = \text{atan2}(p_y, p_x - d_1) \quad (19a)$$

$$\theta_5 = \text{atan2}(r_{2,1}\cos(\theta_1) - r_{1,1}\sin(\theta_1), r_{2,2}\cos(\theta_1) - r_{1,2}\sin(\theta_1)) \quad (19b)$$

$$\theta_2 + \theta_3 + \theta_4 = \text{atan2}(-(r_{1,3}\cos(\theta_1) - r_{2,3}\sin(\theta_1)), r_{3,3}) \quad (19c)$$

Maintenant, pour trouver les angles θ_2 et θ_3 on doit utiliser une approche géométrique. Si on observe d'un repère situé sur l'axe 2 du robot (ce qu'on peut faire parce que, une fois qu'on a θ_1 , il suffit d'obtenir $\mathbf{T}_{1,5}$ et de faire une translation de $-d_2$ par rapport à l'axe X pour obtenir \mathbf{T}_{proj2}), la position de l'axe 5 est défini comme dans la figure 9, où \bar{p}_x , \bar{p}_y et \bar{p}_z sont, respectivement, les éléments (1,4), (2,4) et (3,4) de la de matrice \mathbf{T}_{proj2} .

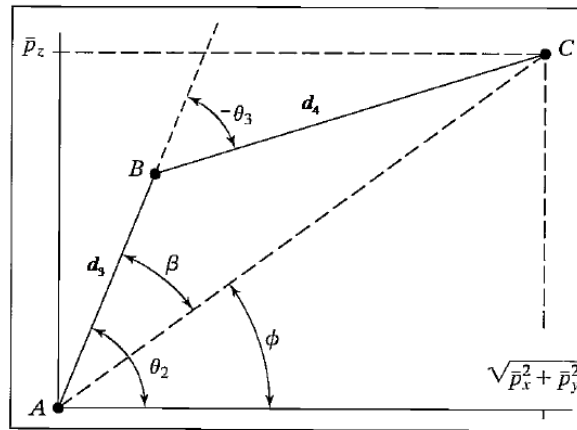


Figure 9 – Configuration du robot youBot KUKA vue de l'articulation 2.

Si on applique la loi des cosinus, on peut trouver les angles θ_2 et θ_3 , selon les équations suivantes. Une fois qu'on a ces deux angles et on a aussi $(\theta_2 + \theta_3 + \theta_4)$, on peut obtenir facilement le θ_4 et de cette façon, on a retrouvé tous les angles.

$$\cos(-\theta_3) = \frac{\overline{p}_x^2 + \overline{p}_y^2 + \overline{p}_z^2 - d_3^2 - d_4^2}{2d_3d_4} \quad (20a)$$

$$\theta_3 = -\text{atan2}(\sqrt{1 - \cos^2(-\theta_3)}, \cos(-\theta_3)) \quad (20b)$$

$$\theta_2 = \text{atan2}(\overline{p}_z, \sqrt{\overline{p}_x^2 + \overline{p}_y^2}) + \text{atan2}(r_4 \sin(\theta_3), r_3 + r_4 \cos(\theta_3)) \quad (20c)$$

Il est possible pour le robot de atteindre une position désirée avec le coude haut ou bas et avec la poignée haute ou basse, alors quatre configurations angulaires différentes pour la même position. Avec cette méthode, on peut retrouver la configuration coude bas et poignée basse par défaut. Si on veut changer la configuration, il suffit d'ajouter ou de soustraire un décalage de π à l'axe 1 pour le coude et à l'axe 3 pour la poignée.

Le problème que nous reste à propos de ce modèle est qu'il marche juste si on essaye une configuration faisable de l'outil. Comme le robot youBot KUKA a 5 degrés de libertés, une fois qu'on a fixé la position de l'outil, il ne peut pas avoir n'importe quelle orientation par rapport à la base du robot. En fait, la position fixée pour l'outil fixe aussi le plan où le bras du robot va rester par l'angle θ_1 . Alors, juste les orientations dont l'axe Z du repère situé dans l'outil est contenue dans ce plan sont faisables.

Pour éviter ce problème, on a décidé de faire tous les trajets du robot au cours du Jeu avec l'outil tourné vers l'avant. Il est facile de déterminer pour n'importe quelle position quelle est l'orientation qui correspond à l'outil vers l'avant. Dans le repère de la première articulation, les axes X et Z déterminent le plan du bras du robot. Par rapport à ce repère, si on veut fixer cette orientation, on a besoin d'utiliser la matrice de rotation suivante :

$$\mathbf{R}_{avant} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

Comme juste la position de l'outil définit l'angle θ_1 , il suffit pour construire $\mathbf{T}_{0,1}$. Alors, on peut rajouter \mathbf{R}_{avant} à la position désirée pour obtenir la matrice homogène $\mathbf{T}_{1,end}$ avec une configuration faisable pour l'outil et revenir au repère de la base en multipliant cette matrice pour $\mathbf{T}_{0,1}^{-1}$.

4.2 Trajectoires

Dans le cadre d'une tâche robotique, le mouvement de l'outil dans l'espace est défini par une séquence de points traversés dans le temps. Cette séquence et son historique temporel est désigné par **trajectoire**. Une tâche est généralement définie comme le mouvement entre une série de points visés, par exemple « Aller de la position initiale au point A, s'approcher de la pièce jusqu'au point B, fermer l'outil en saisissant la pièce, aller au point C en portant la pièce », et l'objectif de la génération de trajectoire est de calculer le chemin traversé dans le temps parmi des points intermédiaires et ses respectifs temps. Une fois calculée, la trajectoire est mise comme référence à l'entrée de la commande afin que l'effecteur suive le chemin souhaité.

La figure 10 montre le schéma-bloc usuel pour un générateur de Trajectoire dans l'espace opérationnel avec une commande articulaire, où $\mathbf{q}(t)$ est le vecteur de positions articulaires et $\mathbf{p}(t)$ est le vecteur des positions cartésiennes de l'outil.

Le schéma est assez simplifié, il n'inclue pas le vecteur d'orientation, mais la commande d'orientation est faite d'une façon assez pareille.

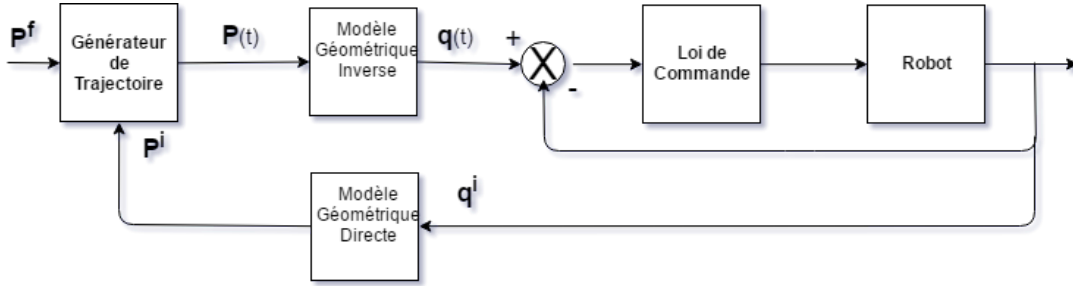


Figure 10 – Schéma-bloc du système avec le générateur de trajectoires

La trajectoire peut être définie en termes de coordonnées articulaires ou cartésiennes, où la première est mieux adaptée pour un chemin libre entre deux points et la deuxième pour un chemin contraint. Dans ce travail on a décidé de générer les trajectoires en coordonnées cartésiennes, ce qui demande une commande en espace opérationnel ou l'utilisation du modèle géométrique inverse, pour les convertir en coordonnées articulaires.

À part les contraintes géométriques de la trajectoire, il y a les contraintes temporelles, c'est-à-dire, contraintes par rapport aux vitesses et accélérations en chaque point de la trajectoire et durées maximales du mouvement. Un des principes de conception d'un générateur de trajectoire c'est l'utilisation de trajectoires lisses, une fois que physiquement c'est impossible de traverser l'espace de façon non-continue. En plus on utilise souvent courbes continues en vitesse et accélération afin de minimiser les soucis par rapport aux vibrations et résonances mécaniques. Afin d'améliorer les vitesses, plusieurs générateurs de trajectoire utilisent techniques d'interpolation polynomiale et spline, mais dans le cadre de ce travail on ne les utilisera pas.

4.2.1 Choix de conception et Mise en Œuvre

Inspiré par les choix de conception des robots Fanuc, on a décidé de mettre en œuvre un générateur de trajectoires avec trois fonctions basiques : trajectoire linéaire entre 2 points, trajectoire circulaire entre 2 points avec un point intermédiaire et trajectoire en arc de cercle entre 2 points avec un point intermédiaire. Avec ces trois types de trajectoire le robot sera capable de réaliser toutes ses tâches.

4.2.2 Trajectoire Linéaire

Le type plus simple de trajectoire, la trajectoire linéaire consiste en aller d'un point initial P^i au point final P^f en suivant une ligne droite, où tous les deux points sont vecteurs tridimensionnels et appartiennent à l'espace opérationnel du robot. Soit t_f le temps total du mouvement, la trajectoire peut être décrite analytiquement par :

$$p(t) = (p^f - p^i)r(t) + p^i \quad , \quad 0 \leq t \leq t_f$$

Où $r(t)$ est une fonction monotone continue du temps avec les conditions limites suivantes :

$$r(0) = 0$$

$$r(t_f) = 1$$

On peut facilement observer que $\mathbf{p}(0) = \mathbf{p}^i$ et que $\mathbf{p}(t_f) = \mathbf{p}^f$, alors la contrainte géométrique est satisfaite. Pour satisfaire les contraintes de temps on définira $r(t)$ comme un polynôme d'interpolation de 5^{ème} degré avec les caractéristiques suivantes :

$$\dot{r}(0) = 0$$

$$\dot{r}(t_f) = 0$$

$$\ddot{r}(0) = 0$$

$$\ddot{r}(t_f) = 0$$

Avec ces 6 conditions limites le polynôme est bien défini :

$$r(t) = 10 \cdot \left(\frac{t}{t_f}\right)^3 - 15 \cdot \left(\frac{t}{t_f}\right)^4 + 6 \cdot \left(\frac{t}{t_f}\right)^5$$

La figure 11 montre l'allure du polynôme d'interpolation $r(t)$, bien comme ses dérivées.

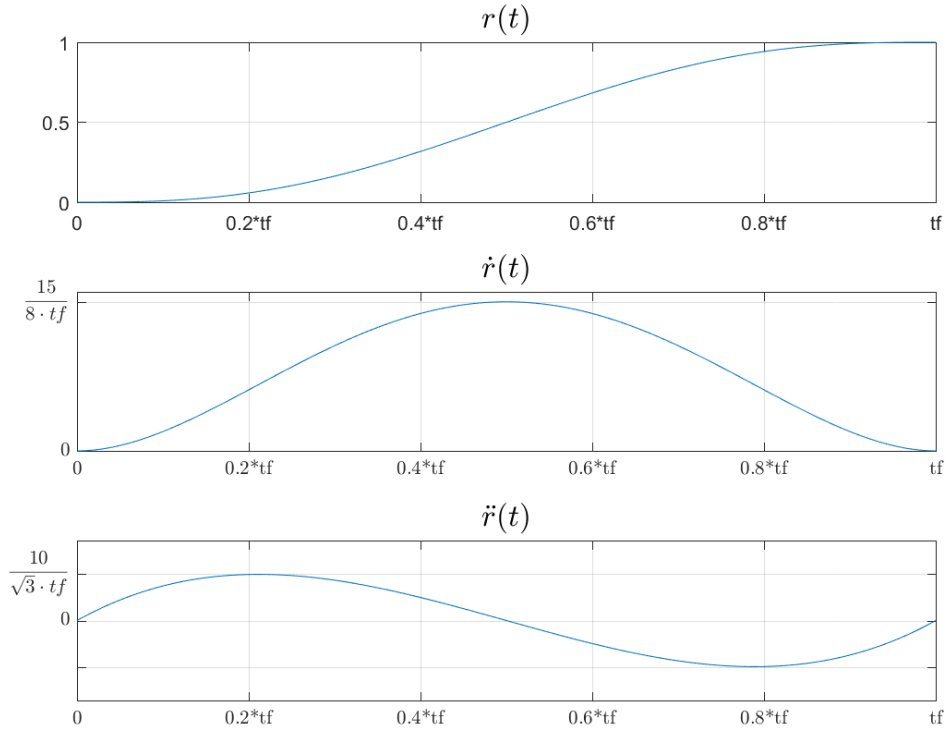


Figure 11 – $r(t)$ et ses dérivées

La trajectoire dans l'espace est montrée dans la figure 12. Les points sont espacés d'un même intervalle de temps afin d'indiquer l'effet de la vitesse sur la trajectoire.

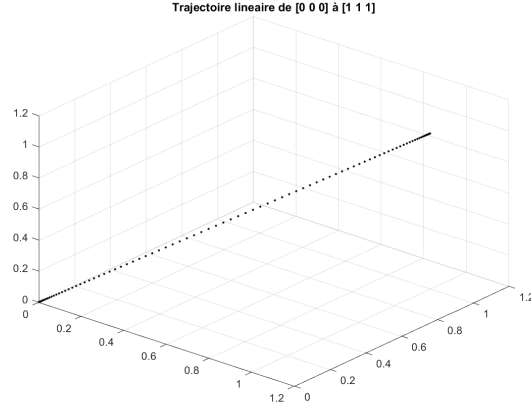


Figure 12 – Trajectoire lineaire dans l'espace

4.2.3 Trajectoire Circulaire

Une des trajectoires souhaitées c'est la trajectoire circulaire dans l'espace. Avec les mêmes principes de conception de la trajectoire linéaire on va définir une courbe dans l'espace et la suivre avec un profil de position donné par un polynôme de 5^{ème} degré, afin de suivre une circonférence dans l'espace de temps entre 0 et t_f . On utilisera la fonction $r(t)$, définie auparavant.

La courbe sera définie par l'utilisateur parmi trois points, que, sauf en cas de colinéarité, définissent toujours un cercle dans l'espace. Une fois défini en termes d'un rayon R et d'un centre \mathbf{p}^c on peut définir la trajectoire dans le temps, dont l'expression est donnée ci-dessous :

$$\mathbf{p}(t) = \mathbf{p}^c + \begin{bmatrix} R \cos(2\pi r(t) + \varphi_0) \\ R \sin(2\pi r(t) + \varphi_0) \end{bmatrix}$$

$$\text{Dont la condition limite est : } \mathbf{p}^c + \begin{bmatrix} R \cos(\varphi_0) \\ R \sin(\varphi_0) \end{bmatrix} = \mathbf{p}^i \text{ (Point initial)}$$

L'expression est donnée en termes du plan x y, mais le résultat est général, une fois qu'il faut seulement multiplier les points résultants par des matrices de rotation pour obtenir une configuration n'importe laquelle, ce qui est en effet la méthode utilisée lors de la mise en œuvre.

La vitesse dans la direction du mouvement est donnée par :

$$\left\| \frac{d\mathbf{p}(t)}{dt} \right\| = \sqrt{\left(\frac{dp_x(t)}{dt} \right)^2 + \left(\frac{dp_y(t)}{dt} \right)^2} = 2\pi R r(t)$$

Ce qui veut dire que la vitesse, bien comme l'accélération, sont limitées et peuvent être facilement réglées en termes de vitesses ou accélérations maximales. La figure 13 montre la trajectoire dans l'espace, où les points sont espacés d'un même intervalle de temps afin de montrer la variation de la vitesse au long de la courbe.

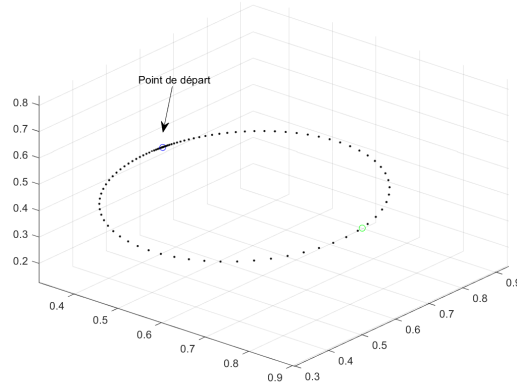


Figure 13 – Trajectoire circulaire dans l'espace

Et la figure 14 montre les positions, vitesses et accélérations par rapport au centre du cercle.

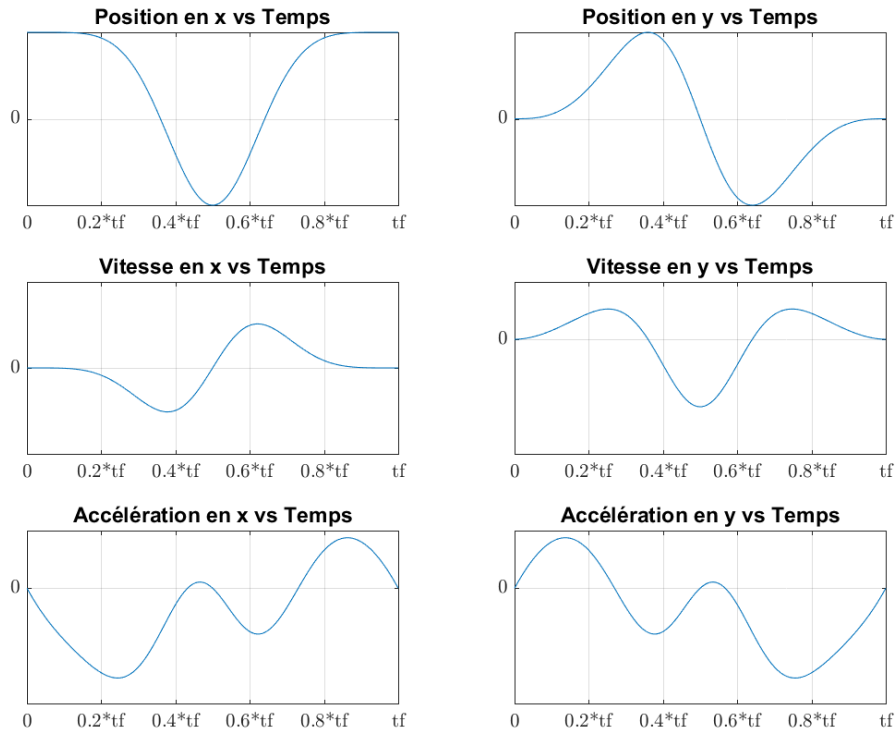


Figure 14 – Positions, vitesses et accélérations en x et y

En plus, si l'utilisateur donne un point initial \mathbf{p}^i , un point final \mathbf{p}^f et un point intermédiaire \mathbf{p}^m on peut définir la trajectoire d'arc de cercle qui passe pour ces trois points.

$$\mathbf{p}(t) = \mathbf{p}^c + \begin{bmatrix} R \cos([\varphi_f - \varphi_0] \cdot r(t) + \varphi_0) \\ R \sin([\varphi_f - \varphi_0] \cdot r(t) + \varphi_0) \end{bmatrix}$$

Où la nouvelle condition limite est :

$$\mathbf{p}^c + \begin{bmatrix} R \cos(\varphi_f) \\ R \sin(\varphi_f) \end{bmatrix} = \mathbf{p}(t_f) = \mathbf{p}^f$$

Où une fois de plus, on a représenté la courbe dans le plan x y. La figure 15 montre un exemple de ce genre de trajectoire de façon générale.

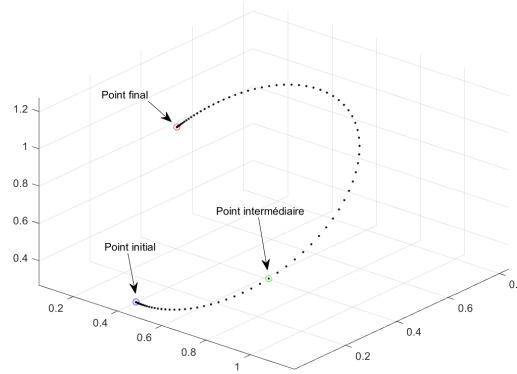


Figure 15 – Trajectoire d'arc de cercle dans l'espace

4.3 Communication avec le robot

A fin de établir la communication entre l'ordinateur et le robot on a choisi d'utiliser l'outil ROS pour faciliter cette communication. ROS est un système d'exploitation open source que a compatibilité avec diverses robots. Plus de détails sur les résultats, comment les tests ont été faites en 5.3 et quelques tutoriels en A.

4.3.1 Études initiaux

Pour meilleur comprendre le paradigme et fonctionnement du ROS nous avons étudié toute la structure des nœuds (nodes) , services, messages, topiques, publieurs (publishers) et abonnés (subscribers). Nous avons utilisés la documentation du ROS [1] pour ces études.

4.3.2 Premiers Tests

Comme les premiers tests, nous avons utilisé les tutoriels des sites [4] et [1] pour apprendre comment utiliser le robot turtle_bot. Après nous avons fait tests en utilisant les tutoriels du turtle_bot en [4], première utilisant le bash, après bash et matlab, et après, finalement utilisant le bash et simulink.

4.3.3 Tests du you_Bot

Après les tests basiques, nous avons commencé a réaliser les tests utilisant le bash, après bash et matlab, et finalement bash et simulink.

4.4 Interface Homme ↔ Machine

L'objectif de cette structure de travail a été concevoir une interface graphique entre l'utilisateur et l'ordinateur qui permet à deux joueurs de réaliser séquentiellement leurs mouvements que puissent être réalisés par le robot sur un tableau réel. En prenant en compte cet objectif et les limitations pour l'exécution de ces tâches, on a construit une interface fonctionnelle, intuitive et déployée de façon plus simple. Alors, on a choisi de développer l'interface par programmation en langage Matlab et de manière modularisée, à savoir, la séparation des différentes tâches en différentes fonctions et avec un haut niveau d'indépendance.

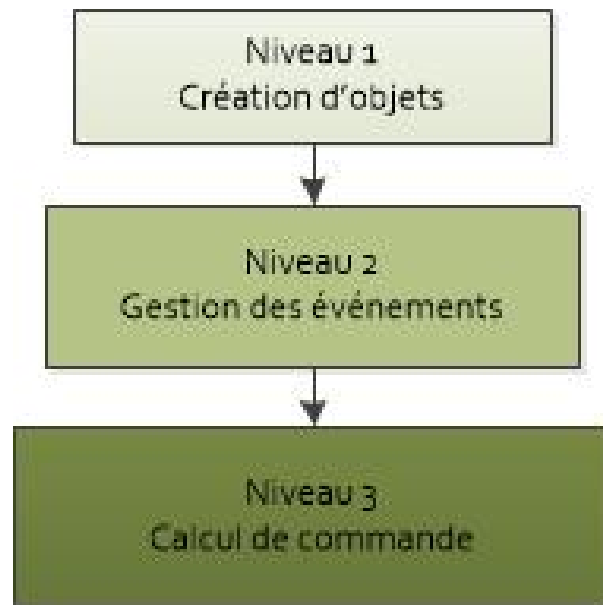


Figure 16 – Les trois différents niveaux des fonctions de construction de l'IHM

Les fonctions du niveau 1 sont concernant l'insertion et réglage des éléments graphiques dans l'interface. Ainsi, au début on a besoin de définir les variables du jeu que doivent être paramétrés et reçus par l'interface et ensuite on a pu mettre les objets graphiques dans l'interface et faire le réglage.

Les fonctions du niveau 2 sont concernant la gestion des événements générés en chaque objet de l'interface. Ainsi, ces fonctions sont responsables d'appeler les fonctions d'intégration avec les algorithmes de calcul de trajectoire et commande, en plus de gérer la structure de données et mettre à jour les paramètres d'entrée des différentes fonctions d'interface.

Les fonctions du niveau 3 sont composées par les fonctions qui intègrent le calcul de trajectoire et le module de commande. Ainsi, elles calculent les trajectoires et couples nécessaires pour le robot réaliser les différentes mouvements générés sur l'interface graphique.

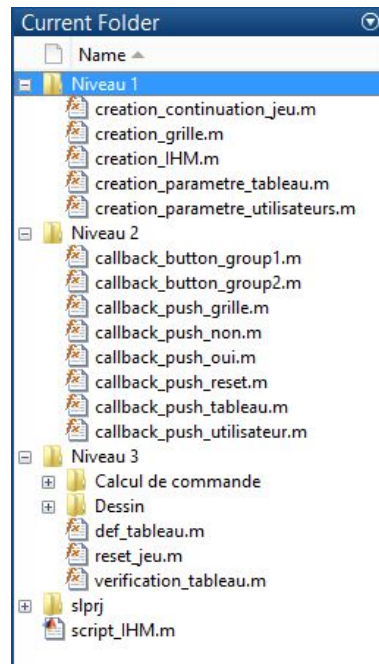


Figure 17 – Le dossier des fonctions de l'IHM

5 Résultats et Discussions

Dans la suite, les principaux résultats qu'on a obtenus tout au long de ce projet sont présentés et discutés.

5.1 Commande Angulaire et Cartésienne

5.1.1 Commande Angulaire

Pour la commande dans l'espace des articulations, on a mis en œuvre sur Matlab et Simulink les techniques de commande montrées dans le subsection 4.1.1. On a évalué les performances de chaque méthode en tenant compte de la réponse à une consigne de position angulaire en échelon filtrée par un système de premier ordre et constante de temps de 0.5s, ainsi que des efforts de commande nécessaires.

On a choisi comme pulsation de brisure $\omega = 10 \text{ rad/s}$ pour toutes les articulations, pour des critères de comparaison. Il est possible de augmenter la fréquence de brisure afin de réduire le temps de réponse du système, mais comme il a été dit, cela peut apporter des efforts de commande trop importants. La choix définitive de cette pulsation doit être fait après l'intégration de la commande avec le robot.

Commande angulaire PID

Les résultats obtenus pour la commande angulaire PID, avec une consigne de $[0, 0, -\frac{\pi}{2}, 0, \frac{\pi}{2}]$ en ce qui concerne l'évolution temporelle des positions angulaire et des efforts de commande sur chaque articulation sont montrés dans la suite.

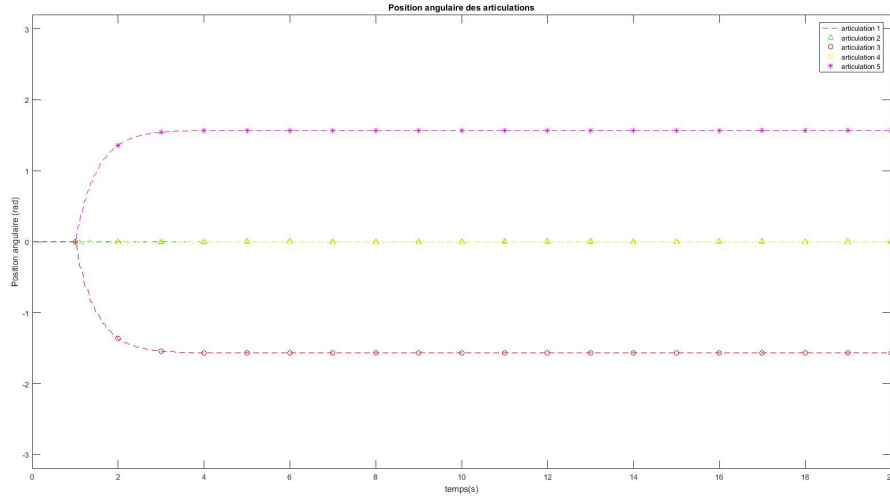


Figure 18 – Réponse temporelle du système commandé par un PID dans l'espace articulaire

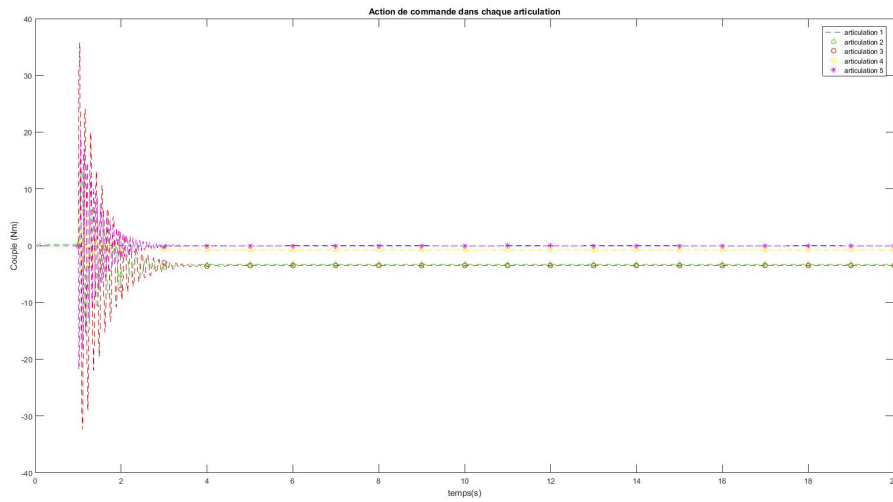


Figure 19 – Efforts de commande appliqués au système commandé par un PID dans l'espace articulaire

On peut noter dans les graphiques que le robot atteint bien la position voulue avec des efforts de commande en régime qui sont non nuls pour compenser l'action de la gravité, mais qui ne sont pas aussi trop importants. Il existe des petites fluctuations dans la réponse transitoire de la position angulaire, mais elles ne gênent pas beaucoup la dynamique du système. Le seul problème est que l'effort de commande devient plus important au début pour démarrer le mouvement du robot.

On a refait les graphiques en supposant que les mesures de position sont bruitées. Le bruit considéré est un bruit blanc avec une variance de 10^{-4} . On n'a pas trouvé d'information à propos des capteurs du robot, mais il serait mieux si on avait choisi cette variance pour être équivalent à la précision de ce capteur.

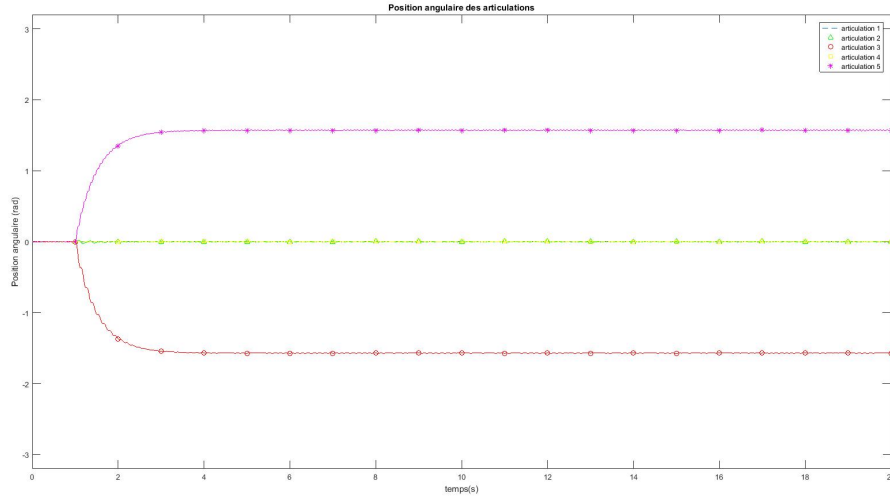


Figure 20 – Réponse temporelle du système commandé par un PID dans l'espace articulaire avec perturbation de mesure

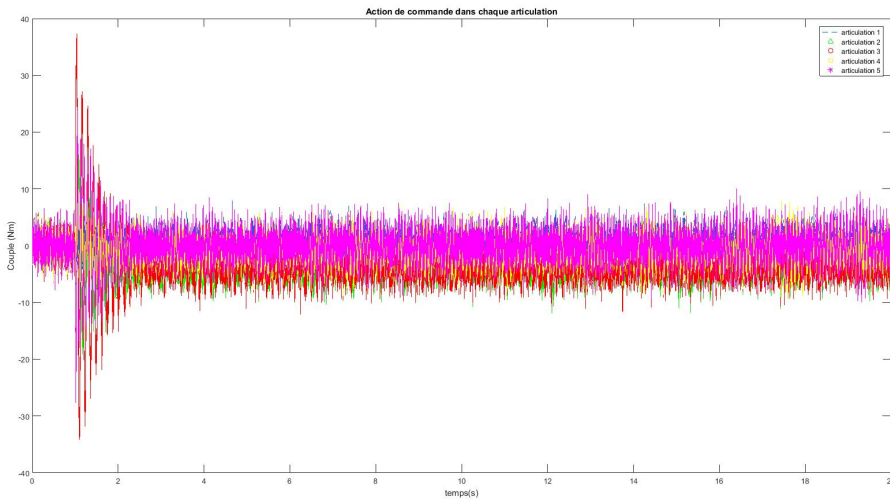


Figure 21 – Efforts de commande appliqués au système commandé par un PID dans l'espace articulaire avec perturbation de mesure

Malgré le petit bruit présent au régime, la réponse de la position angulaire reste stable. Par contre, les efforts de commande deviennent très bruit pour maintenir cette stabilité de position, en dépit de ne pas atteindre des valeurs beaucoup plus élevées que dans le cas précédent. De toute façon, il y a des soucis pour la commande PID si la précision des capteurs est faible.

Commande angulaire par couple calculé

Pour la commande angulaire par couple calculé, on a généré les mêmes graphiques en simulation que pour la commande PID, avec la même consigne de $[0, 0, -\frac{\pi}{2}, 0, \frac{\pi}{2}]$. Les résultats sont montrés dans la suite.

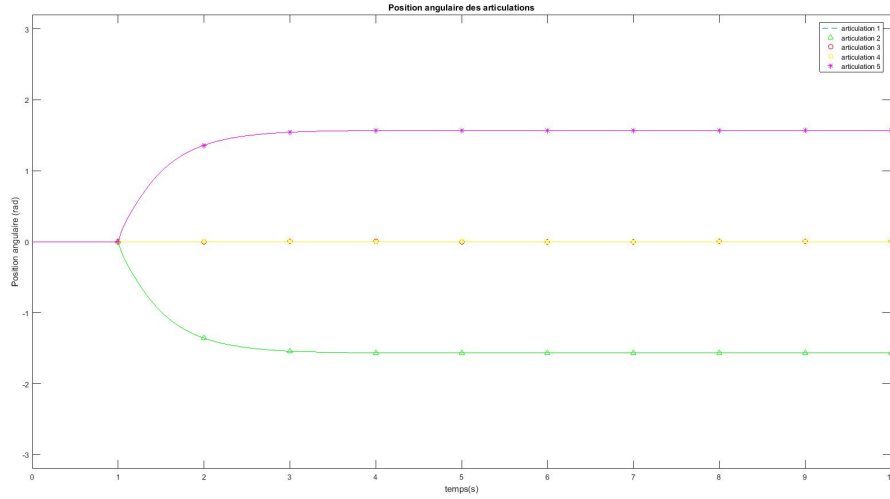


Figure 22 – Réponse temporelle du système commandé par la méthode du couple calculé dans l'espace articulaire

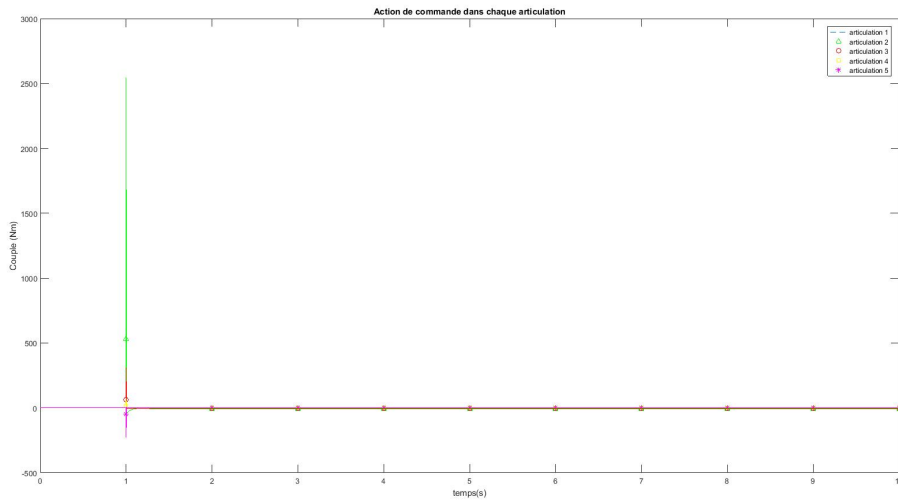


Figure 23 – Efforts de commande appliqués au système commandé par la méthode du couple calculé dans l'espace articulaire

On peut noter dans ces graphiques qu'on a réduit les fluctuations de la réponse, une fois qu'on considère que le modèle est bien adapté au système et de cette façon, on compense les non-linérités. Par contre, cet avantage vient avec le coût d'augmenter considérablement les efforts de commande dans le transitoire de la réponse.

Ensuite, on a essayé aussi de simuler des problèmes qu'on peut avoir lorsque la commande est intégrée au robot : on a ajouté un bruit blanc avec une variance de 10^{-2} dans la mesure des positions angulaires, on a déterminé la vitesse angulaire par dérivation numérique de la position et on a ajouté une désadaptation de 5% dans quelques paramètres du modèle par rapport au système. Les résultats sont montrés dans les graphiques suivants.

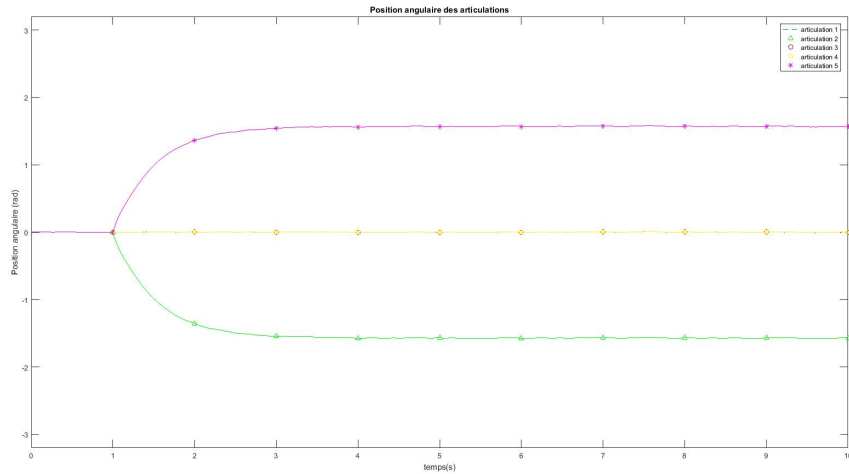


Figure 24 – Réponse temporelle du système commandé par la méthode du couple calculé dans l'espace articulaire avec perturbation de mesure et desadaptation du modèle

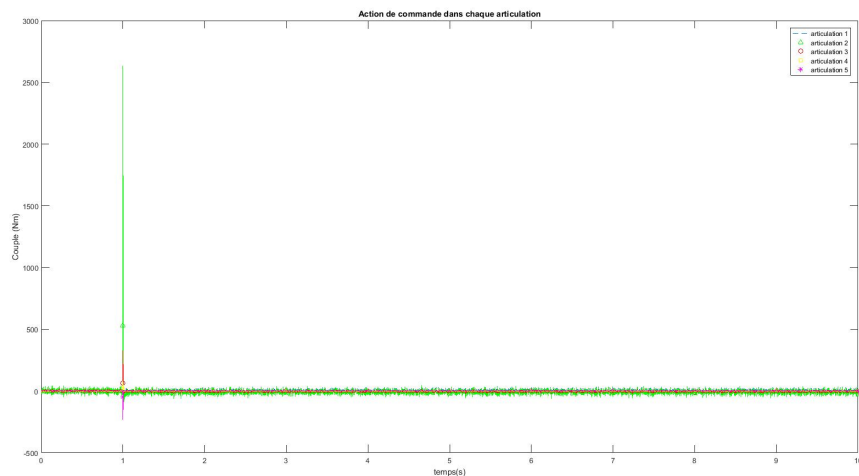


Figure 25 – Efforts de commande appliqués au système commandé par la méthode du couple calculé dans l'espace articulaire avec perturbation de mesure et desadaptation du modèle

Cette méthode de commande est moins sensible aux bruits de mesure que la dernière, parce qu'on a ajouté un bruit avec une variance plus importante et le signal de sortie est moins bruité. Par contre, les efforts de commande sont aussi bruités que pour la commande PID.

Une prochaine étape pour ce projet serait d'ajouter les limitations angulaires, de vitesse et de couples des moteurs des articulations dans le modèle Simulink pour analyser les performances de la commande dans un cas un peu plus réaliste avant de l'intégrer au robot. On n'a pas ajouté ses limitations parce qu'on a trouvé dans le manuel du fabricant juste les limitations angulaires. Il faudra faire une recherche plus détaillée, ou même contacter les fabricants pour obtenir ces informations.

5.1.2 Commande Cartésienne

Pour la commande dans l'espace opérationnel, on a mis en œuvre sur Matlab et Simulink les techniques de commande montrées dans le subsection 4.1.2. En plus de tenir en compte la réponse temporelle des positions angulaires et de la position de l'outil à des consignes en échelon filtré et les efforts de commande pour évaluer les performances de chaque méthode, on a considéré aussi la capacité du système commandé de suivre des trajectoires linéaires ou circulaires.

Commande cascade avec boucle interne de vitesse angulaire

La commande cascade avec boucle interne de vitesse angulaire a été mise en œuvre sur Matlab Simulink pour commander la position de l'outil du robot. Par contre, on n'a pas eu le temps d'adapter cette méthode pour commander aussi l'orientation de l'outil, parce qu'on n'a pas réussi encore de calculer les vitesses de rotation du repère associé à l'outil par rapport au repère associé à la base du robot. On a toute la trajectoire de référence pour le robot décrite par des matrices homogènes, il manque juste de les utiliser pour calculer les vitesses de rotation.

Dans la suite, on a les graphiques de la position de l'outil par rapport à la base du robot, des positions angulaires et des efforts appliqués à chaque articulation pour une trajectoire linéaire et pour une trajectoire circulaire.

On observe que, pour les deux types de trajectoires, le robot les suit très bien, presque sans aucune erreur. Les mouvements angulaires réalisés sont lisses et se stabilisent après la fin de la trajectoire. Les efforts de commande restent avec des valeurs peu importantes malgré quelque bruit existant. Ce bruit est dû à la détermination de la consigne de vitesse angulaire pour le robot, qui est faite de façon numérique avec un bloc "dérivé" du Simulink. Il existe d'autres manières numériques de faire cette estimation, qui conduisent à des résultats plus fiables et moins bruités, mais on n'a pas eu le temps de les mettre en œuvre.

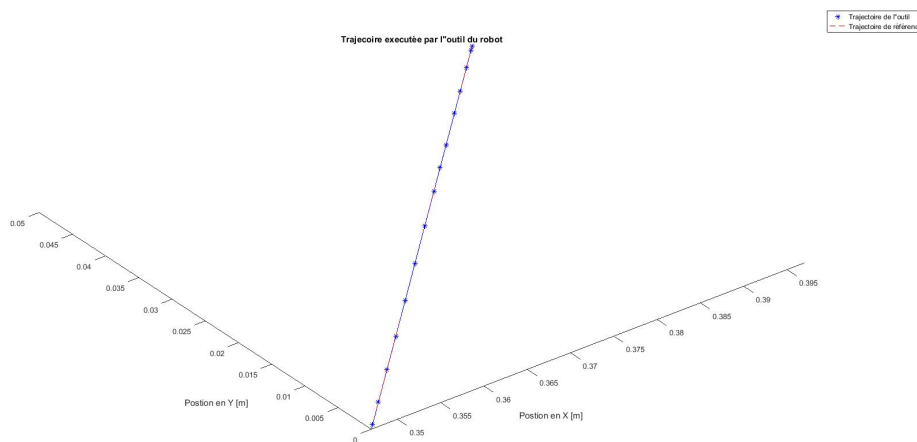


Figure 26 – Comparaison de la trajectoire exécutée par l'outil du robot commandé par la méthode cascade et la trajectoire de référence linéaire

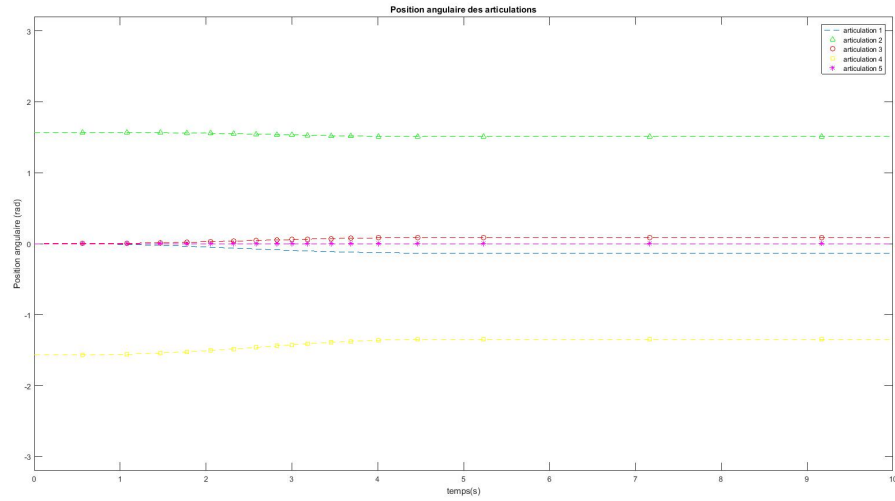


Figure 27 – Réponse temporelle des positions angulaires des axes du robot commandé par la méthode cascade pour suivre une trajectoire linéaire

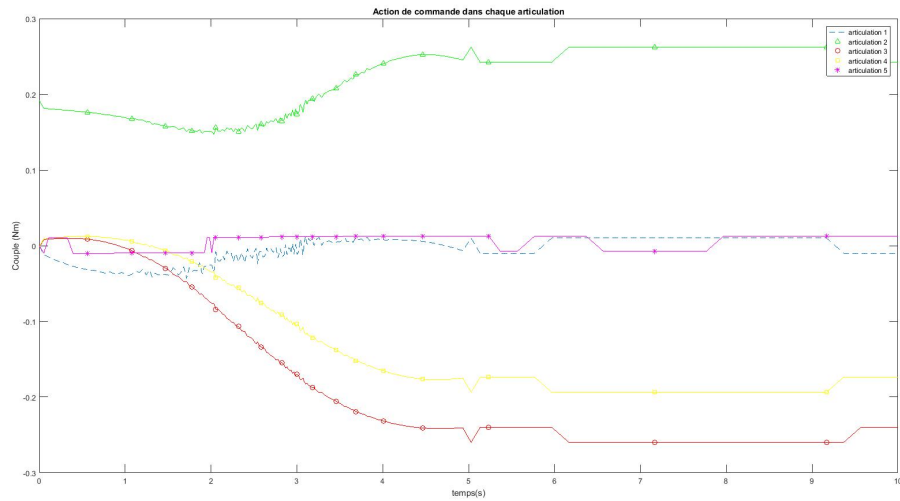


Figure 28 – Efforts de commande appliqués aux articulations du robot commandé par la méthode cascade pour suivre une trajectoire linéaire

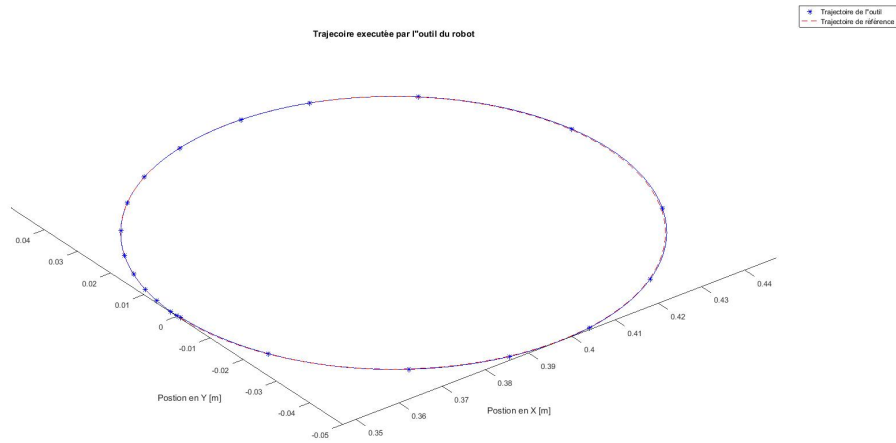


Figure 29 – Comparaison de la trajectoire exécutée par l'outil du robot commandé par la méthode cascade et la trajectoire de référence circulaire

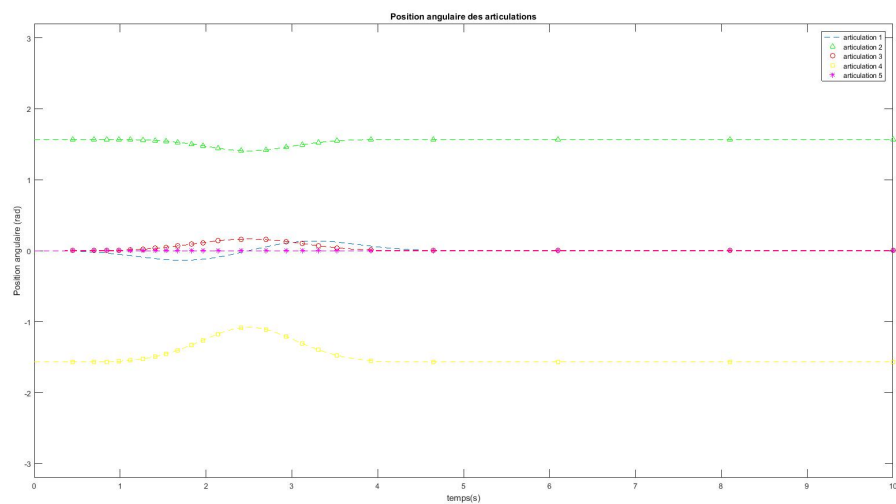


Figure 30 – Réponse temporelle des positions angulaires des axes du robot commandé par la méthode cascade pour suivre une trajectoire circulaire

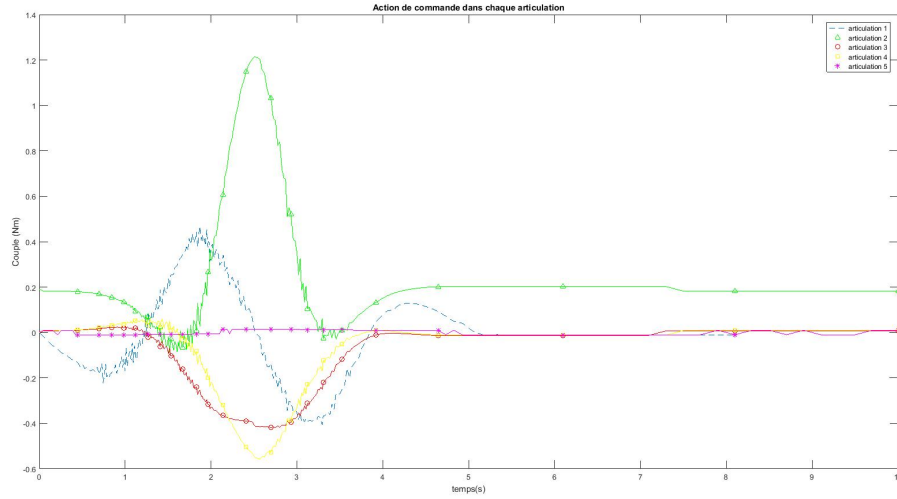


Figure 31 – Efforts de commande appliqués aux articulations du robot commandé par la méthode cascade pour suivre une trajectoire circulaire

Commande par MDI

On a mis en œuvre cette commande sur Matlab et Simulink en utilisant la commande PID angulaire qui était déjà près. Premièrement, on a besoin de convertir tous les trajectoires de l'espace articulaire à l'espace angulaire en utilisant le MDI. Ensuite, on peut utiliser ces trajectoires comme consignes pour la commande angulaire PID développée sur Simulink. Cela nous donne des résultats similaires pour ces deux méthodes de commande. Dans la suite, on a les graphiques de la réponse temporelle de la position de l'outil à une consigne de variation de position en échelon filtré par un système de premier ordre de valeur $[0.1, 0, 0]$, des positions angulaire des articulation et des efforts de commande.

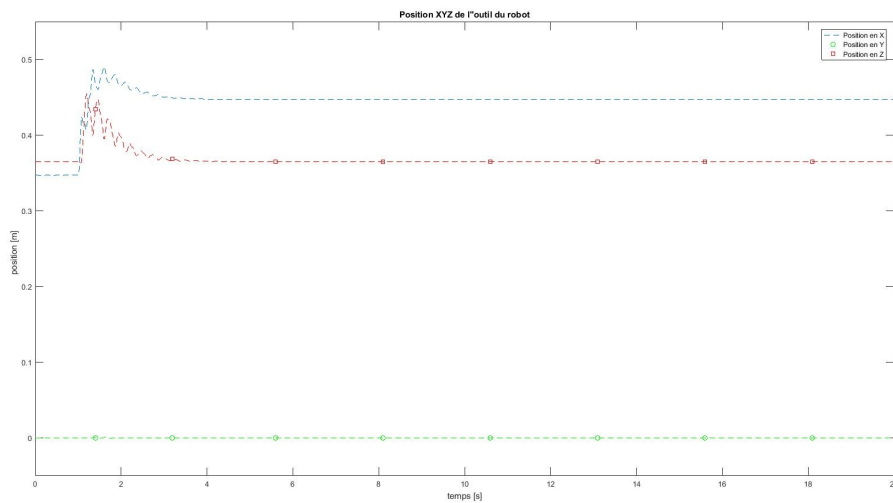


Figure 32 – Réponse temporelle de la position de l'outil du robot commandé par le MDI et un PID articulaire à une consigne en échelon filtré

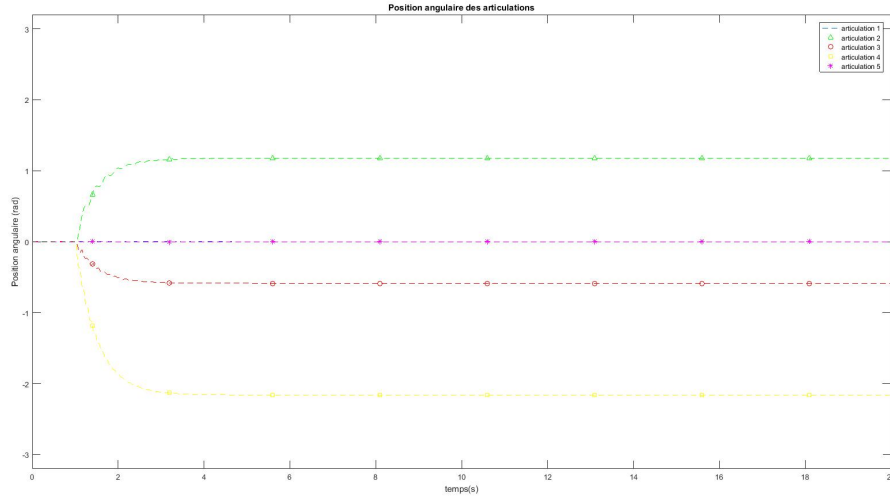


Figure 33 – Réponse temporelle des positions angulaires des axes du robot commandé par le MDI et un PID articulaire à une consigne en échelon filtré

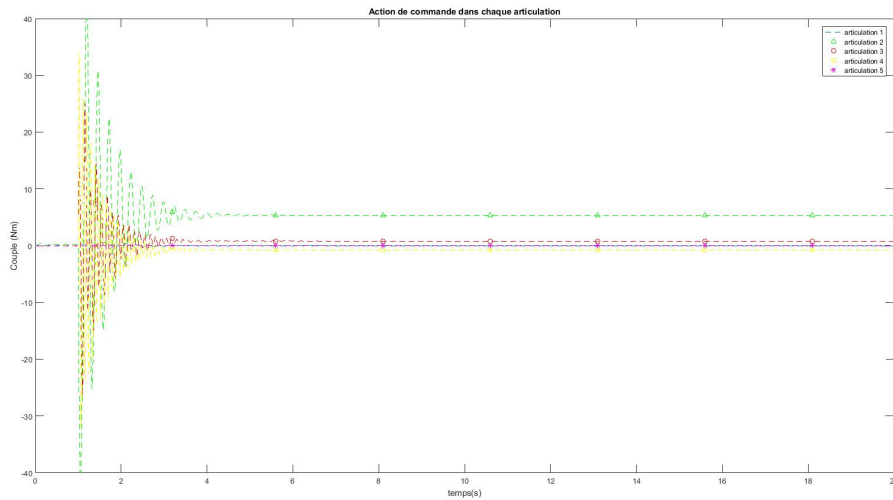


Figure 34 – Efforts de commande appliqués aux articulations du robot commandé par le MDI et un PID articulaire à une consigne en échelon filtré

On observe dans ces graphiques que, au régime, le système suit bien la consigne, avec les positions angulaires stables et les efforts de commande qui ne sont pas trop importants. Par contre, il y a beaucoup de fluctuation dans le transitoire, et les efforts de commande arrivent à des valeurs plus importantes. Cela était attendu une fois que ces fluctuations sont présentes aussi dans la commande angulaire PID.

Dans la suite, on a les réponses de ce système à des trajectoires linéaire et angulaire de consigne, qui représentent mieux la façon dont on veut que le robot se déplace dans notre application.

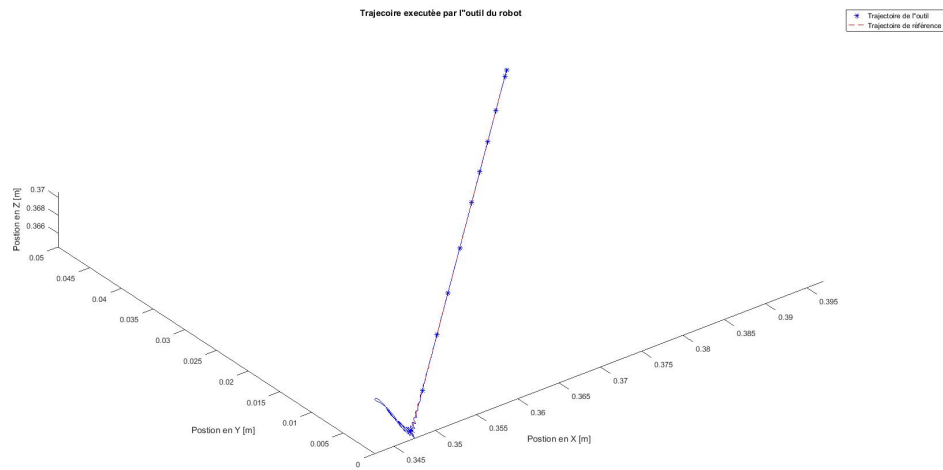


Figure 35 – Comparaison de la trajectoire exécutée par l'outil du robot commandé par le MDI et un PID angulaire et la trajectoire de référence linéaire

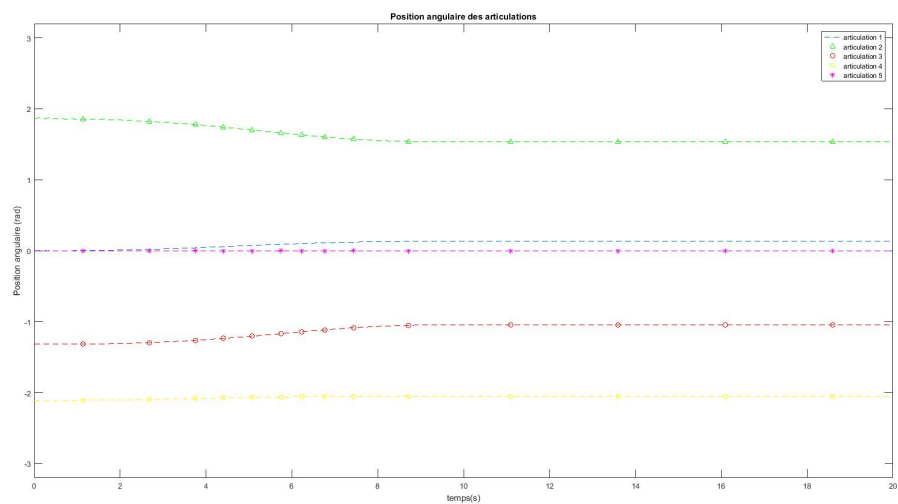


Figure 36 – Réponse temporelle des positions angulaires des axes du robot commandé par le MDI et un PID angulaire pour suivre une trajectoire linéaire

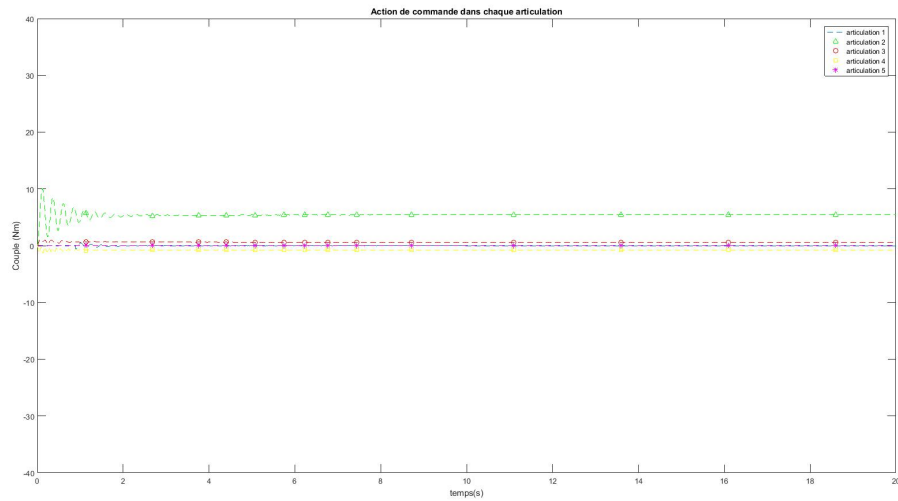


Figure 37 – Efforts de commande appliqués aux articulations du robot commandé par le MDI et un PID angulaire pour suivre une trajectoire linéaire

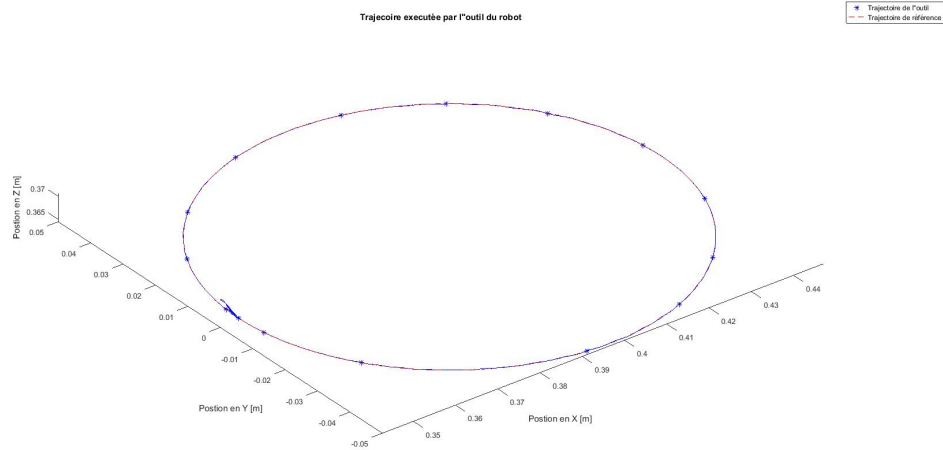


Figure 38 – Comparaison de la trajectoire exécutée par l'outil du robot commandé par le MDI et un PID angulaire et la trajectoire de référence circulaire

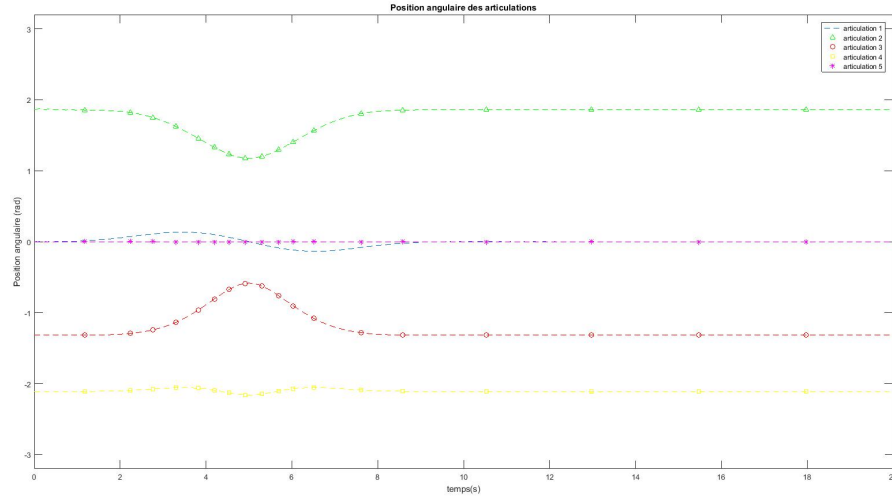


Figure 39 – Réponse temporelle des positions angulaires des axes du robot commandé par le MDI et un PID angulaire pour suivre une trajectoire circulaire

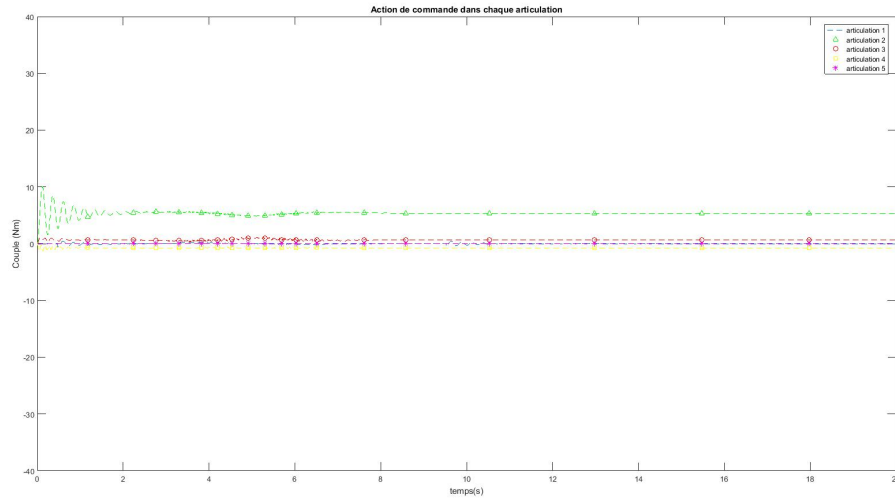


Figure 40 – Efforts de commande appliqués aux articulations du robot commandé par le MDI et un PID angulaire pour suivre une trajectoire circulaire

On peut noter que le système suit bien les deux types de trajectoire, avec des variations angulaires qui sont lisses et stabilisent bien après le mouvement et avec des efforts de commande qui ne sont pas trop importants. Le seul problème sont des fluctuations qui apparaissent au début du déplacement, associées aux fluctuations dans les efforts de commande. En dépit d'être de petite amplitude (de l'ordre de 5mm), ces fluctuations peuvent gêner le mouvement du robot. Un meilleur réglage du PID est nécessaire pour réduire ces problèmes.

On a dit dans la section 4.1.2 que cette méthode de commande prend en compte aussi l'orientation de l'outil et que pour un robot de 5 axes, une fois fixée la position, il ne sont faisables que les orientations dans lesquelles l'axe Z du repère associé à l'outil est contenu dans le plan du bras du robot. Pour résoudre ce

problème, on a décidé de faire les trajectoires avec l'outil tourné vers l'avant. En conséquence, il est nécessaire une étape au début du Jeu pour mettre le robot dans la même position initiale mais avec l'orientation désirée. On peut après laisser le robot pendant tout le jeu avec cette orientation. Dans la suite, on a les graphiques avec les réponses obtenues pour cette étape de orientation de l'outil.

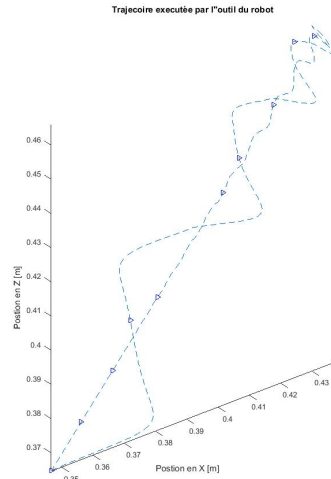


Figure 41 – Trajectoire exécutée par l'outil du robot commandé par le MDI et un PID angulaire dans l'étape de orientation de l'outil

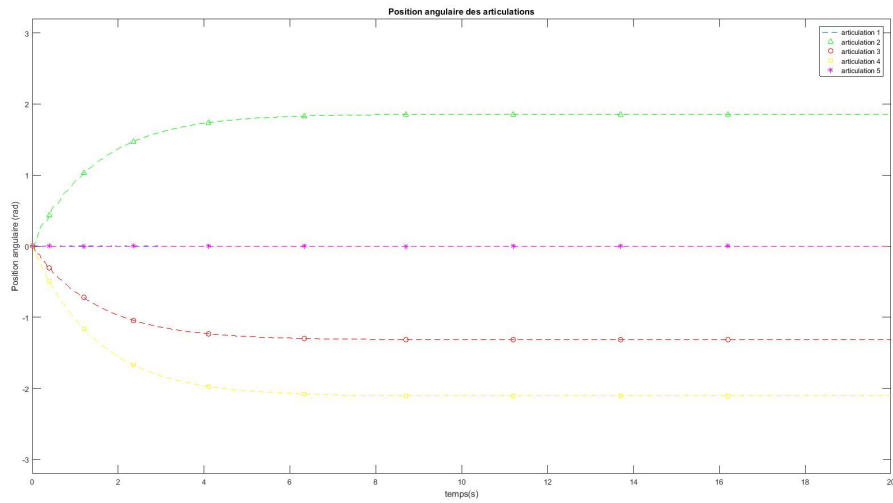


Figure 42 – Réponse temporelle des positions angulaires des axes du robot commandé par le MDI et un PID angulaire dans l'étape de orientation de l'outil

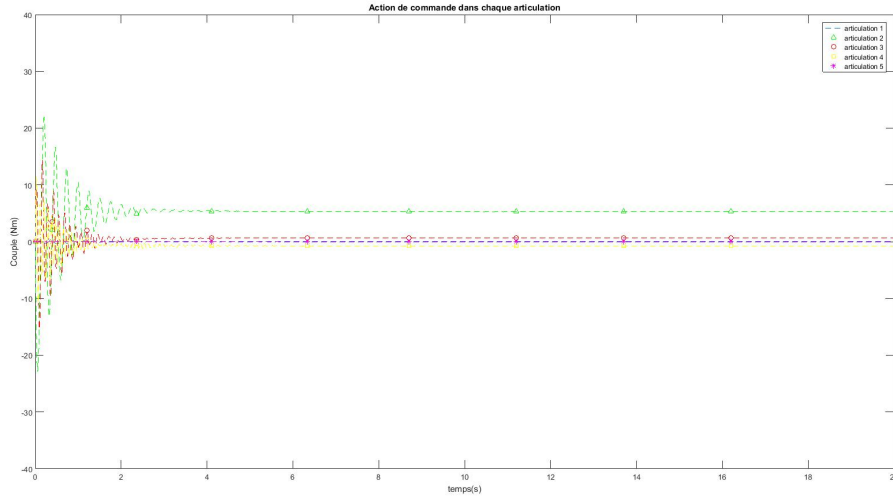


Figure 43 – Efforts de commande appliqués aux articulations du robot commandé par le MDI et un PID angulaire dans l'étape de orientation de l'outil

On observe que le robot retourne bien au point initial après un déplacement d'environ 14cm, les réponses des positions angulaires des axes sont lisses et les efforts de commande appliqués ne sont pas trop importants, malgré les fluctuations au début dues à une réglage insuffisante du PID angulaire.

Commande PID Cartésienne

On a essayé aussi de faire la commande du robot dans l'espace opérationnel avec juste un PID. Cette méthode est le plus simple sur le plan conceptuel, par contre il est vraiment difficile de le mettre en œuvre parce que il n'y a pas de méthode pour régler le PID, une fois que les équations qui décrivent le système sont non-linéaires et bien accouplé.

Dans la suite, on montre les réponse obtenues par cette méthode avec une consigne de variation de position de $[X, Y, Z] = [0.2, 0.2, -0.1]$ en échelon filtré par un système de premier ordre de constante de temps de 0.5s.

Premièrement, le robot n'arrive pas exactement à la consigne donnée, il reste toujours une erreur en régime permanent. De plus, le système est trop lente : il prend presque 7s pour arriver au régime, contre 2s des méthodes précédentes. Les positions angulaires et les efforts de contrôlen'arrivent jamais à stabiliser, malgré ces efforts restent dans les valeurs peu importants. Pours ces motifs, on a supprimé cette méthode.

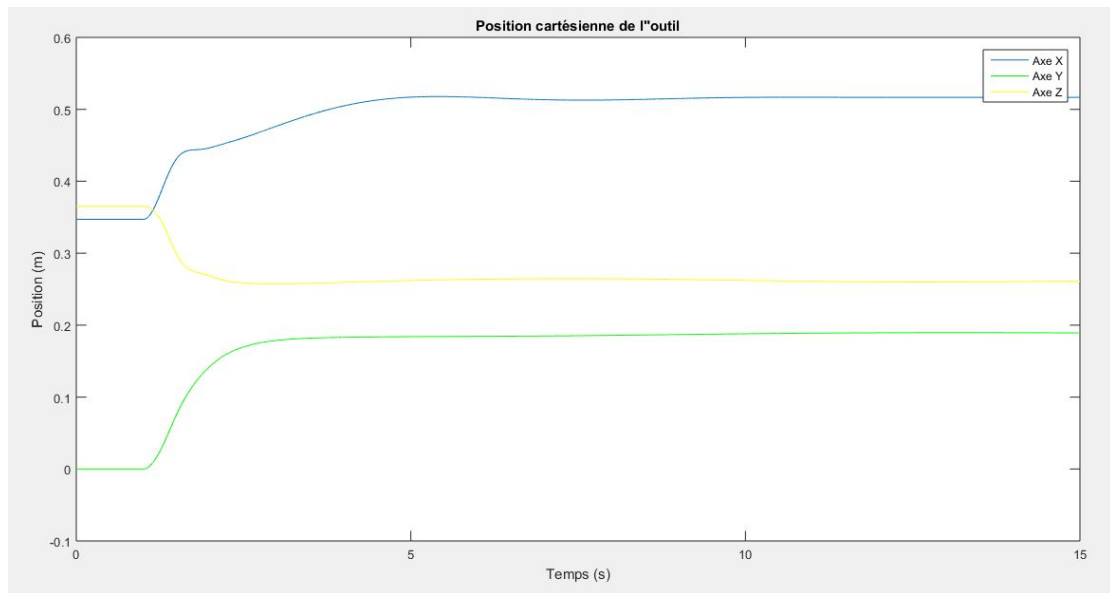


Figure 44 – Réponse temporelle des positions de l'outil à un échelon filtré de consigne pour le système commandé par un PID dans l'espace opérationnel

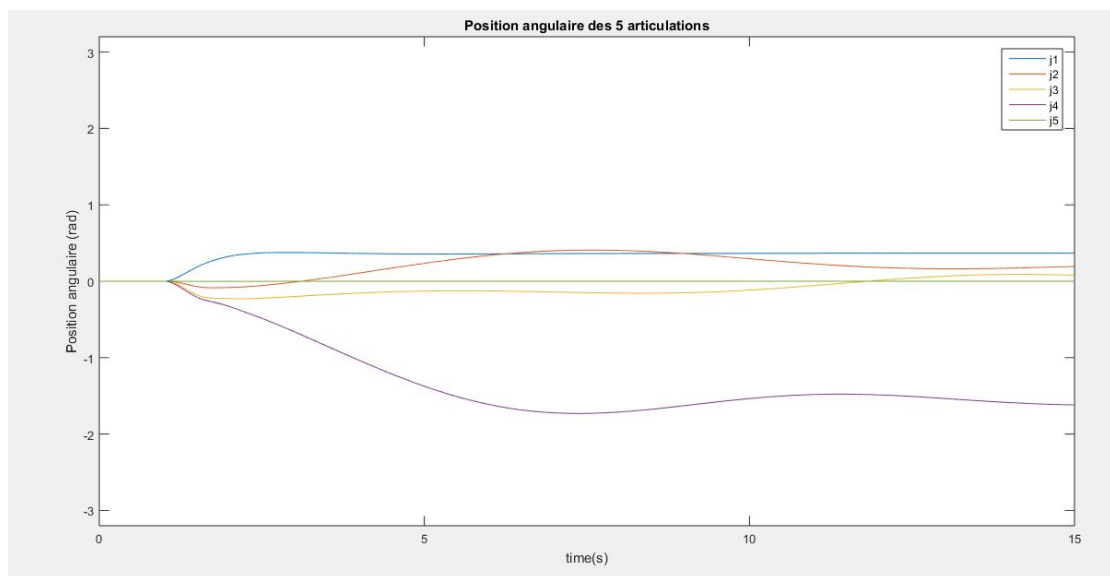


Figure 45 – Réponse temporelle des positions angulaires des axes du robot à un échelon filtré de consigne pour le système commandé par un PID dans l'espace opérationnel

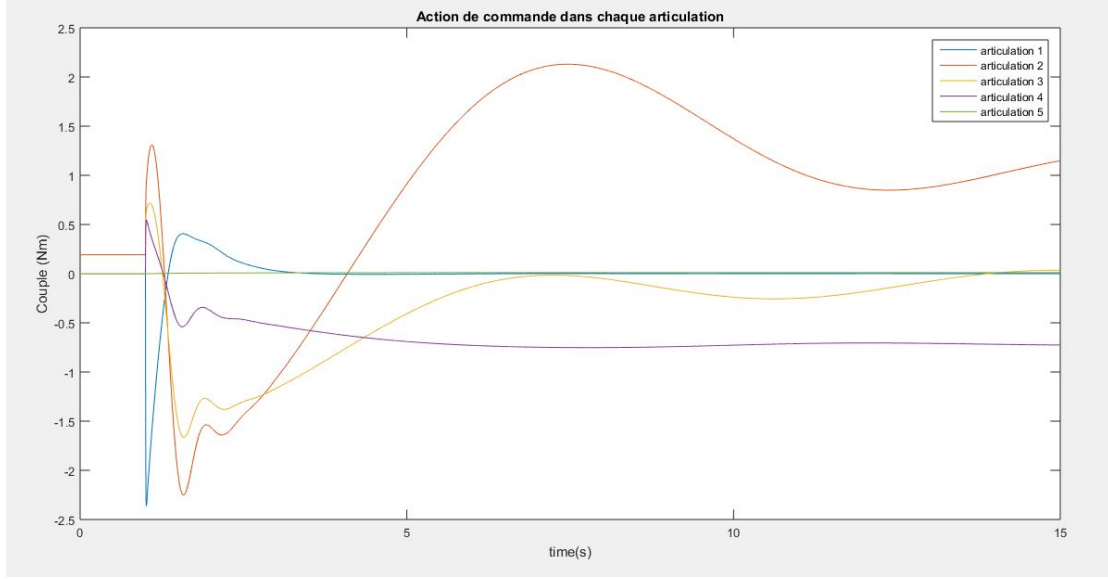


Figure 46 – Efforts de commande pour le système commandé par un PID dans l'espace opérationnel

5.2 Trajectoires

Linéaire Pour faire la trajectoire linéaire, on a utilisé le polynôme de cinquième degré montré en [8], $10(\frac{t}{t_f})^3 - 15(\frac{t}{t_f})^4 + 6(\frac{t}{t_f})^5$, et modifié pour utiliser les entrées, de point initial, point final et le temps total du mouvement.

Par exemple, si le point initial p_i , le point final p_f et le temps t_f sont connus, donc la position suit la fonction suivante :

$$P(t) = (p_f - p_i)(10(\frac{t}{t_f})^3 - 15(\frac{t}{t_f})^4 + 6(\frac{t}{t_f})^5) + p_i \quad (21)$$

Pour un point initial $p_i = [10, 5, 9]$, un point final $p_f = [7, 9, 8]$ et temps final $t_f = 10s$

$$X(t) = (7 - 10)(10(\frac{t}{10})^3 - 15(\frac{t}{10})^4 + 6(\frac{t}{10})^5) + 10 \quad (22a)$$

$$Y(t) = (9 - 5)(10(\frac{t}{10})^3 - 15(\frac{t}{10})^4 + 6(\frac{t}{10})^5) + 5 \quad (22b)$$

$$Z(t) = (8 - 9)(10(\frac{t}{10})^3 - 15(\frac{t}{10})^4 + 6(\frac{t}{10})^5) + 9 \quad (22c)$$

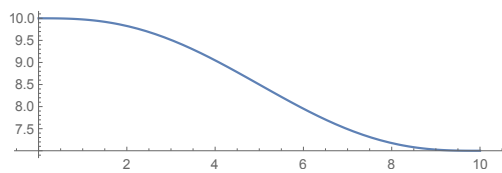


Figure 47 – Graphique de la paramétrisation de X.

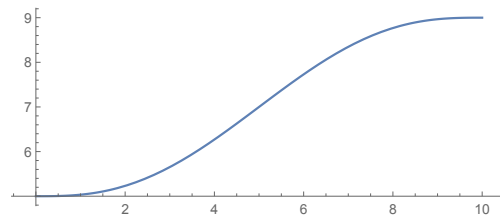


Figure 48 – Graphique de la paramétrisation de Y.

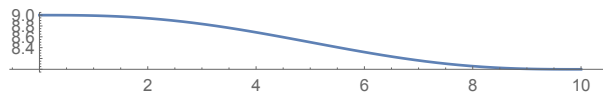


Figure 49 – Graphique de la paramétrisation de Z.

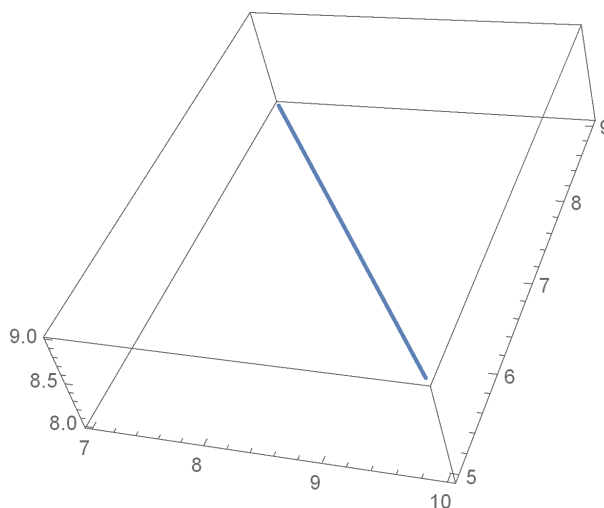


Figure 50 – Représentation tridimensionnelle de la trajectoire

Arc de cercle Dans un première moment on a choisi essayer de résoudre le problème en deux dimensions.

Comme paramétrisation du cercle on a pensé en utiliser tel :

$$X = 1 - \cos(t) \quad (23a)$$

$$Y = \sin(t) \quad (23b)$$

Modifiant la fonction pour la généraliser et en faisant t varier de 0 à la valeur du angle entre les points initial et final et le centre du cercle.

Mais après nous avons utiliser une variable intermédiaire pour lisser les courbes de position, vitesse et accélération, comme vue en 3.2.

Les résultats sont lesquels ont été déjà indiquées et le graphique dupliqué est le suivant :

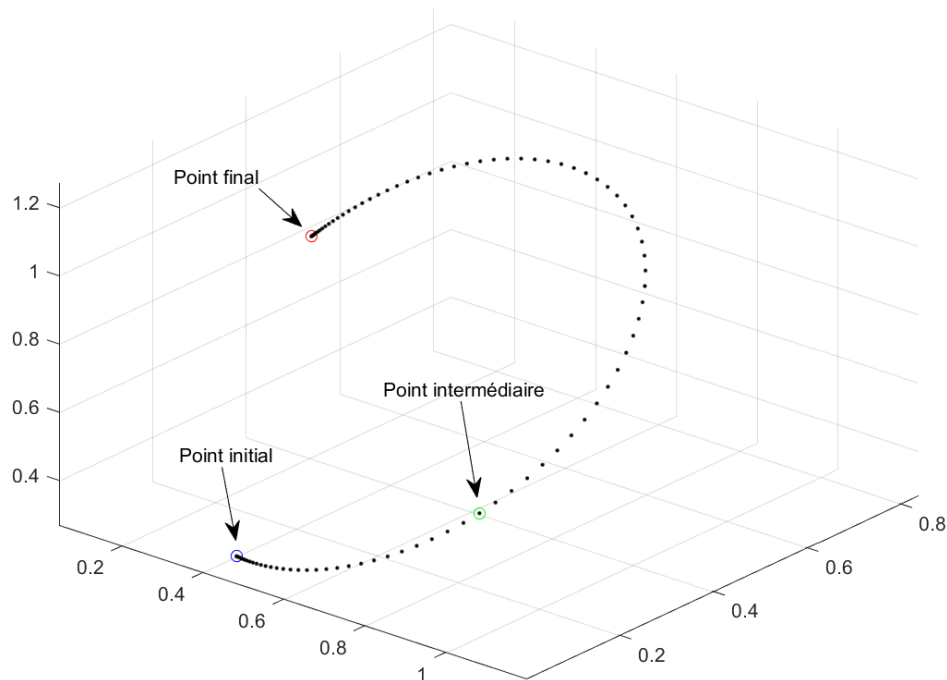


Figure 51 – Trajectoire d'arc de cercle dans l'espace

5.3 Communication avec le robot

5.3.1 Premiers Tests

Comme les ressources pour apprendre comment faire pour connecter le Matlab/Simulink et you_Bot pour un débutant a presque aucune documentation, donc nous avons utilisé les turtle_bot pour apprendre. Nous avons installé tout les drivers nécessaires et le gazebo, simulateur des robots. Comme premier exemple nous avons utilisé le script `teleop`, où il'est possible de télécommander le robot en utilisant le clavier. Après nous avons utilisé les tutoriels de Matlab pour faire la communication entre matlab/simulink et le robot en gazebo, la communication a été en deux sens : Nous avons réussi d'acquérir la position du robot, la image de sa camera et au même temps envoyer une direction de mouvement pour le robot.

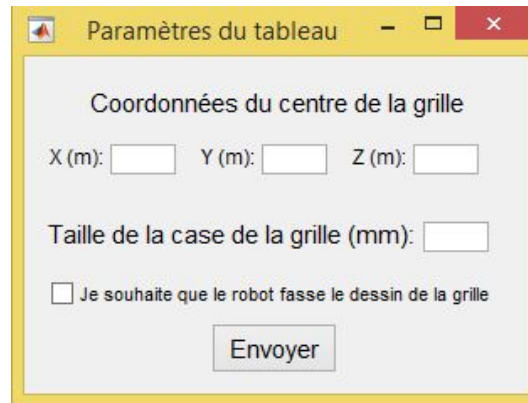
5.3.2 Tests du you_Bot

Après nous avons essayé de faire les mêmes tests en utilisant les drivers you_Bot, mais malheureusement a cause des contraintes des temps la communication a été juste en un sens, nous n'avons réussi qu'acquérir la position du robot.

5.4 Interface Homme ↔ Machine

L'interface est composée par une fenêtre où les utilisateurs réalisent les différents mouvements qui composent le jeu. De plus, elle doit recevoir les paramètres concernant la construction du tableau et l'identification des joueurs. Ainsi, l'interface est composée par trois fenêtres principales que sont ouvertes séquentiellement.

5.4.1 Première fenêtre : Paramètres du tableau

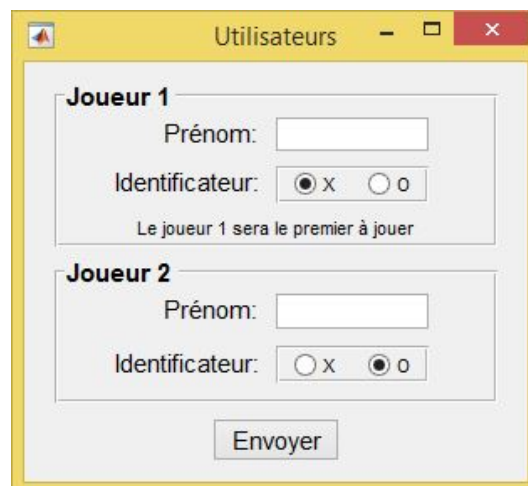


The screenshot shows a window titled "Paramètres du tableau" with a yellow border. Inside, the text "Coordonnées du centre de la grille" is followed by three input fields for "X (m):", "Y (m):", and "Z (m):". Below these is a single input field for "Taille de la case de la grille (mm):". A checkbox labeled "Je souhaite que le robot fasse le dessin de la grille" is present, followed by an "Envoyer" button.

Figure 52 – Fenêtre que reçoit les paramètres du tableau

La position de l'outil et de chaque élément du jeu est décrit dans le repère de la base du robot. Cette fenêtre reçoit les positions du centre du tableau dans le repère de la base du robot. Par ailleurs, pour définir complètement le tableau, la taille de la case de la grille doit être déterminé. Par convention, on a adopté la position initiale du robot comme le centre X et Y du tableau et avec la hauteur Z égal à la taille de la case de la grille. De plus, cette fenêtre offre une option pour calculer le couple et la trajectoire de l'outil génère pour dessiner la grille du tableau.

5.4.2 Deuxième fenêtre : Utilisateurs



The screenshot shows a window titled "Utilisateurs" with a yellow border. It contains two sections for player identification. "Joueur 1" has a "Prénom:" input field and an "Identificateur:" section with radio buttons for 'x' (selected) and 'o'. Below this is the text "Le joueur 1 sera le premier à jouer". "Joueur 2" has a "Prénom:" input field and an "Identificateur:" section with radio buttons for 'x' and 'o' (selected). An "Envoyer" button is at the bottom.

Figure 53 – Fenêtre que reçoit l'identification des utilisateurs

Pour dérouler le jeu entre deux différents utilisateurs, on a créé une fenêtre que reçoit les paramètres d'identification de chaque joueur, comme son prénom et le symbole associé à chaque joueur. Par convention, on a adopté que le jeu est initié par le joueur 1.

5.4.3 Troisième fenêtre : Jeu de Morpion

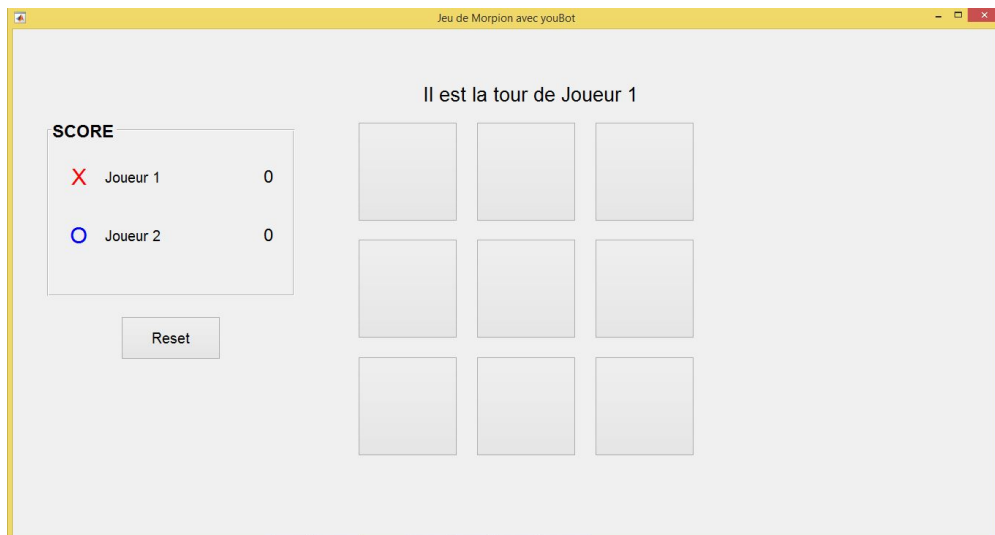


Figure 54 – Fenêtre que reçoit les mouvements du jeu de Morpion

Cette fenêtre reçoit les mouvements souhaités par chaque joueur. Lorsque le joueur clique sur une case souhaitée, les fonctions de calcul de trajectoire et couple sont appelés et la trajectoire à être exécuté par le robot avec les déplacements sur les cases du tableau et le dessin du symbole de chaque joueur est tracé sur un graphique 3D.

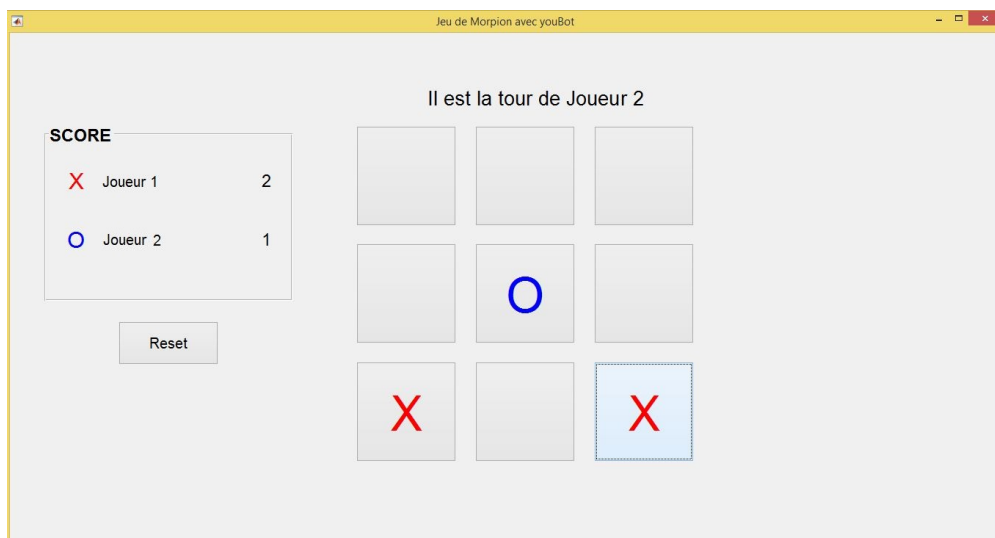


Figure 55 – Tableau du jeu avec les mouvements des joueurs

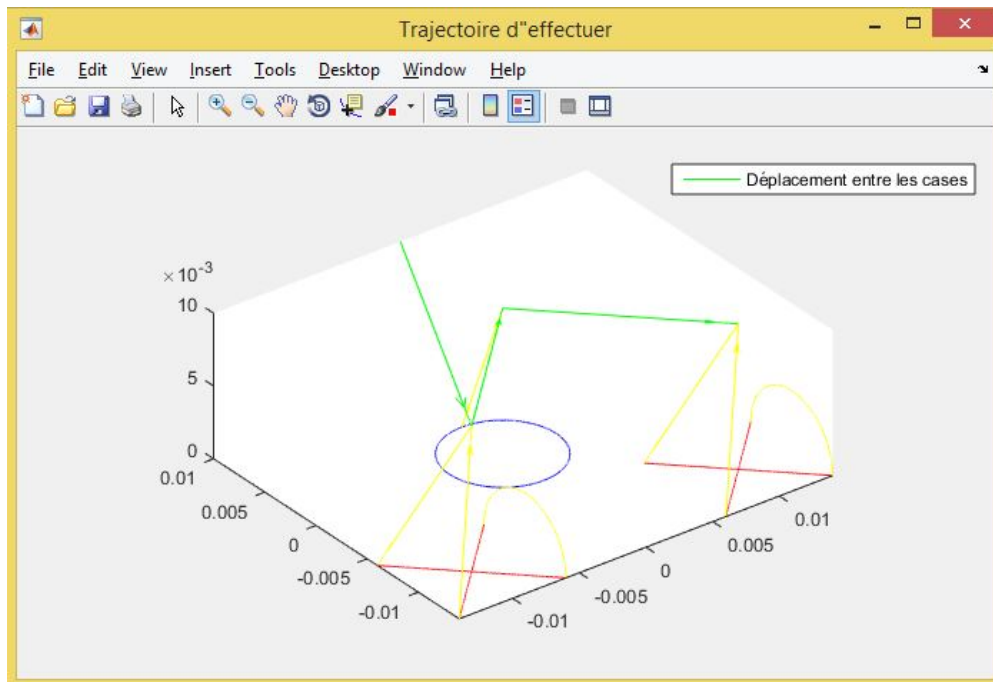


Figure 56 – Graphique 3D que décrit la trajectoire du robot pour faire les mouvements du jeu

Ainsi, ce graphique permet la visualisation du mouvement que le robot fera dans une mise en oeuvre réel du jeu. Comme l'activité d'intégration et synchronisation du module de commande avec l'interface n'est pas été fini, les courbes tracées dans le graphique sont concernant seulement le résultat de la génération de trajectoire, au lieu de la position finale générée par le module de commande calculée à partir de la trajectoire souhaitée. Cependant, comme le module de commande est déjà bien réglé et mis en oeuvre, pour un travail futur suffira de bien régler la synchronisation.

Quand un tour du jeu est terminé, l'interface annonce le gagnant ou un possible match nul et elle offre aux utilisateurs l'option d'initier un nouveau tour. Si un nouveau tour est initié, le score de chaque joueur est calculé et actualisé sur le panneau de résultats à gauche de l'interface. De plus, il existe l'option de redémarrer un tour en cliquant sur le bouton Reset.

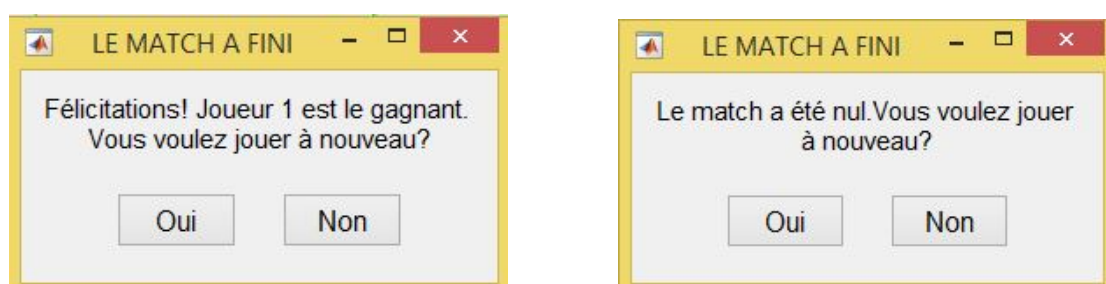


Figure 57 – Option d'initier un nouveau tour quand un match finit

6 Conclusion

Si on se rappelle des objectifs définis lors du début de ce document on peut, enfin vérifier dans quels aspects on les a atteints. A propos de la commande du robot on a bien atteint l'objectif d'avoir des lois de commande fonctionnelles, soit dans l'espace articulaire soit dans l'espace opérationnel. On a arrivé à suivre les trajectoires proposés, avec un système assez stable. Ce qui manque maintenant c'est un réglage plus affiné des paramètres des contrôleurs et vérifier les performances en prenant en compte les désadaptations des modèles et des bruits de mesure. Parmi tous ces schémas de commande, ce qu'on a retenu pour notre application c'est la commande cascade. Il nous permet de commander le robot dans l'espace opérationnel et les réponses sont assez performantes. Par rapport à la génération de trajectoires on a bien atteint l'objectif principal de générer les trajectoires nécessaires pour l'application. Cependant, il faut encore vérifier les points de chaque trajectoire afin d'éviter des points de singularité. Pour l'interface homme-machine, elle est déjà prête et on peut y lancer notre application, tout en générant la séquence de trajectoires de l'outil du robot. Il faut maintenant l'intégrer avec la partie commande et le ROS. Alors, en ce qui concerne la communication entre MATLAB et le robot simulé, il reste encore des points très importantes à conclure. Dans un travail futur, il sera nécessaire de mettre en œuvre la boucle de commande en utilisant les subscribers pour mesurer les données des capteurs, et les publishers pour agir sur le robot. En plus il faudra se débrouiller sur le code source du modèle du youbot afin d'accéder directement à l'effort, de façon à imposer notre commande sur le robot.

Appendices

A Tutoriels you_Bot / Matlab

A.1 Installation you_Bot

Pour installer le pilote you_Bot, il faut premièrement avoir ROS installé dans la machine, dans notre cas nous avons utilisés une machine virtuel avec ROS Indigo pré-installé. La machine est disponible utilisant le lien en [3]. Après, il faut courir les commandes au terminal :

```
sudo apt-get install ros-indigo-youbot-driver
sudo ldconfig /opt/ros/indigo/lib
sudo apt-get install ros-indigo-youbot-driver-ros-interface \
    ros-indigo-youbot-description
sudo setcap cap_net_raw+ep \
    /opt/ros/indigo/lib/youbot_driver_ros_interface\
    /youbot_driver_ros_interface
sudo ldconfig /opt/ros/indigo/lib
```

Maintenant le pilote et le «wrapper» sont installés. Pour installer le simulateur gazebo et les modèles du you_Bot il faut courir :

```
sudo apt-get install ros-indigo-ros-control ros-indigo-ros-controllers\
    ros-indigo-gazebo-ros-control
cd <your-catkin-folder>/src
git clone http://github.com/youbot/youbot_description.git -b indigo-devel
git clone http://github.com/youbot/youbot_simulation.git
catkin_make
source <your-catkin-folder>/devel/setup.bash
```

Où <your-catkin-folder> est le nom du «catkin workspace» utilisé, [5]. Pour tester il faut courir :

```
roslaunch youbot_gazebo_robot youbot.launch
```

Et dans autre terminal :

```
roslaunch youbot_driver_ros_interface youbot_keyboard_teleop.py
```

Maintenant utilisant les commandes donnés par le programme, on peut mou-
vementer le robot. Pour faire le test des articulations il faut courir le code suivant
au lieu du antérieur :

```
roslaunch youbot_driver_ros_interface youbot_arm_test
```

Chacun de ces programmes peuvent être arrêtés par ^C.

Remarque : Le code `roslaunch youbot_gazebo_robot youbot.launch` charge
dans le simulateur le modèle du robot avec sa base roulante, au lieu de ce code il
faut utiliser le code suivant, pour le modèle sans la base :

```
roslaunch youbot_gazebo_robot youbot_arm_only.launch
```

A.2 Configuration Matlab/ROS/you_Bot

Avec une distribution récent du Matlab (R2015a par exemple) et que aie le «Robotics System Toolbox» installé (lien et tutoriels en [4]). Il faut le configurer, pour commencer un node ROS au matlab et le connecter a un master (dans nos cas, le robot) il faut courir cet code :

```
rosinit('host_ip')
```

Où `host_ip` est l'adresse ip du master, 192.168.1.1 par exemple. Pour s'assurer de la valeur du adresse ip, il faut courir dans la machine où le robot est connecté :

```
ifconfig
```

Et dans le terminal il aura la valeur du ip de la machine. Cas il n'y est pas là, consulter la connexion de la machine avec le réseaux.

Après la connexion entre Matlab et la machine connecté au robot, il faut le tester, donc on peut créer un abonné a partir du matlab pour lire les informations données par le robot :

```
joints_subs = rossubscriber('/joint_states')
```

Et après pour recevoir les valeurs :

```
joint_data = receive(joints_subs,10)
```

Et maintenant, nous avons les valeurs momentanées de position, vitesse et effort, dans nos cas l'angle, la vitesse angulaire et le torque en chaque joint.

Remarque : Pour envoyer les torques il faut plus des temps d'étude, a fin de déterminer que topique a utiliser.

A.3 Configuration Simulink/ROS/you_Bot

La connexion entre Simulink et la Machine est un peu similaire au qu'a été fait antérieurement mais purement graphique utilisant la «GUI» du Simulink.

Dans une nouvelle fenêtre du Simulink, et après sélectionner dans le menu **Tools > Robot Operating System > Configure Network Addresses**, après saisir l'adresse ip du master, dans le champ **Hostname/IP Address** et cliquer en **Ok**.

Dans le modèle insérer un bloc «Subscribe» , un bloc «Terminator», un «Bus Selector» et 3 «Scopes».

Sélectionner le bloc «Subscribe» et le configurer pour recevoir le topique `joint_states`.

Lier **IsNew** dans le terminal du bloc «Terminator» et **Msg** dans la entrée du «Bus Selector».

Dans «Bus Selector», sélectionner Position, Vitesse et Effort.

Lier chacune sortie du «Bus Selector» a un «Scope» différent.

Après, faire courir la simulation. Chaque «Scopes» aura 7 courbes, chacune caractérisant les 7 articulations, 5 joint plus 2 acteurs des pinces.

Remarque : Pour envoyer les torques il faut plus des temps d'étude, a fin de déterminer que topique a utiliser.

Références

- [1] Documentation du ros. <http://wiki.ros.org/>. Accessed : 2017-05-20.
- [2] Kuka youbot kinematics, dynamics and 3d model. <http://www.youbot-store.com/developers/kuka-youbot-kinematics-dynamics-and-3d-model-81>. Accessed : 2017-05-10.
- [3] Lien de téléchargement de la machine avec ros pré-installé. https://www.mathworks.com/supportfiles/robotics/ros/virtual_machines/v3/installation_instructions.htm. Accessed : 2017-05-20.
- [4] Robotics system toolbox examples. <https://fr.mathworks.com/help/robotics/examples.html>. Accessed : 2017-05-20.
- [5] Ros creation espace travaille catkin. http://wiki.ros.org/catkin/Tutorials/create_a_workspace. Accessed : 2017-05-20.
- [6] J.J. Craig. *Introduction to Robotics : Mechanics and Control*. Addison-Wesley series in electrical and computer engineering : control engineering. Pearson/Prentice Hall, 2005.
- [7] Rafael Kelly and Javier Moreno. Manipulator motion control in operational space using joint velocity inner loops. *Automatica*, 41(8) :1423 – 1432, 2005.
- [8] W. Khalil and E. Dombre. *Modeling, Identification and Control of Robots*. Kogan Page Science paper edition. Elsevier Science, 2004.
- [9] K. Ogata. *Modern Control Engineering*. Instrumentation and controls series. Prentice Hall, 2010.