UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMMANDE PID ET PAR LOGIQUE FLOUE POUR L'AMÉLIORATION DE LA FINESSE ET DE LA PRÉCISION DES MOUVEMENTS D'UN BRAS ROBOTISÉ À PLUSIEURS AXES ET DEGRÉS DE LIBERTÉ.

MÉMOIRE PRÉSENTÉ
COMME EXIGENCE PARTIELLE DE LA

MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
HAMDI FRIKHA

JUIN 2021

UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

MAÎTRISE EN GÉNIE ÉLECTRIQUE (M. Sc. A.)

Direction de recherche :

Alben Cardenas, Professeur	Directeur de recherche

Jury d'évaluation

Alben Cardenas, Professeur	Évaluateur
Mamadou Doumbia, Professeur	Évaluateur
Tahar Tafticht, Professeur	Évaluateur externe

Résumé

La majorité des processus industriels nécessite le contrôle d'un certain nombre de grandeurs telles que la température, la pression, la vitesse, la position, etc.

Ce projet de Maitrise concerne, tout d'abord, l'étude des méthodes de la commande des bras robotisés pour améliorer la finesse et la précision des mouvements et la réalisation d'un bras robot manipulateur à trois degrés de liberté. Puis, nous avons effectué une exploration générale du domaine de la robotique dans laquelle nous avons abordé le vif du sujet en étudiant les différents organes constituant le robot manipulateur. Par la suite, nous nous sommes penchés sur sa modélisation en procédant à l'étude pratique et l'implémentation.

Afin de réaliser ce travail, plusieurs techniques ont été utilisées pour contrôler les grandeurs physiques, de telle sorte que celles-ci gardent constamment leurs valeurs ou restent près des valeurs désirées, quelles que soient les perturbations qui peuvent subvenir. La régulation d'un système physique est basée sur trois opérations fondamentales, à savoir, Mesure, Décision et Action. Quant à la quatrième partie de ce travail, elle a pour but d'appliquer les commandes PID et Logique Floue sur le robot manipulateur, c'est ce que nous avons, d'ailleurs, testé directement sur le bras robot. Nous avons, donc, réalisé une étude comparative sur l'efficacité de chaque commande ayant un lien avec l'objectif d'améliorer la finesse des mouvements.

Ainsi, cette étude a permis de mettre en évidence, par voie expérimentale, les avantages de la commande par Logique Floue pour l'amélioration de la finesse des mouvements dans le contexte des bras robots à faible coût.

Remerciement

Je tiens d'abord à remercier mon directeur de recherche M. Alben Cardenas, qui a joué un rôle important dans la réalisation de ce travail, pour la qualité dans l'enseignement, le suivi, les conseils judicieux, sa disponibilité permanente et surtout ses qualités pro-pédagogiques et humaines typiques.

Mes remerciements s'adressent également à Mme. Nathalie Tourigny, pour ses conseils toujours très utiles, son soutien, ainsi que son aide et disponibilité en ce qui concerne les démarches et orientations relatives à la scolarité et au suivi administratif des étudiants.

J'exprime ensuite ma gratitude au personnel et aux professeurs de l'Université du Québec à Trois-Rivières et en particulier du département de génie électrique et génie informatique (GEGI).

Je tiens à remercier mon père Mohamed Frikha et ma mère Nedra Frikha pour leur soutien bienveillant et financier, je souhaiterais remercier également mes deux frères Imed Frikha, Skander Frikha et ma sœur Randa Frikha pour leur soutien inconditionnel tout au long de la réalisation de ce mémoire.

Table des matières

Liste des tableaux

Liste des figures

Liste des symboles

SMA	Système Mécanique Articulé
CAO	Conception Assistée par l'Ordinateur
C	Cosinus
S	Sinus
θ	Vecteur de variable articulaire du bras manipulateur
$\dot{\theta}$	Vecteur de vitesse articulaire
ω	Vitesse angulaire
q	Vecteur de position
T_i^{i-1}	Matrice de transformation homogène.
DH	Denavit-Hartenberg
ai	Distance entre zi-1 et zi , le long de xi.
$\boldsymbol{\alpha}$i	Angle entre zi-1 et zi , autour de xi.
θi	Angle entre xi-1 et xi, autour de zi-1.
di	Distance de Oi-1 à l'intersection de zi-1 avec xi.
E_c	Erreur
MGD	Modèle Géométrique Direct
MGI	Modèle Géométrique Inverse

J(q)	Matrice Jacobienne
MCD	Modélisation cinématique direct
MCI	Modèle cinématique inverse
PID	Proportionnel, Intégral, Dérivé
Kp	Gain Proportionnel.
Ki	Gain Intégral.
Kd	Gain Dérivé.
PWM	Pulse Width Modulation
A.F.R.I	Association Française de Robotique Industriel
RPP ou PRP	Une articulation Rotoide et deux articulation Prismatiques
RRP	Deux articulations Rotoides et une articulation Prismatique
RRR	Trois articulation Rotoide consécutives
SCARA	Selective Compliance Articulated Robot for Assemblage

Chapitre 1 - Introduction

1.1 Contexte général

Lorsque nous parlons de la robotique, plusieurs idées viennent à l'esprit de chacun de nous. Historiquement, nous pourrions nous référer aux premiers concepts et automates de l'antiquité ou aux premiers robots comme à des personnages de la mythologie.

Même le mot 'robot' a sa propre histoire. Séparer la science-fiction n'est pas une chose aisée, notamment qu'en robotique, nous cherchons parfois à faire réaliser la fiction, un exemple de l'influence des fictions nous est donné par les lois de la robotique [1].

Depuis la révolution industrielle, la robotique comme discipline a marqué l'évolution du monde technologique. L'avènement des robots dans l'industrie a permis de soulager l'homme des travaux répétitifs et difficiles tels que : le déplacement d'objets lourds, les tâches d'assemblages, les microsoudures, etc. [2].

1.2 Problématique

Depuis leur introduction dans l'industrie, les robots manipulateurs ont beaucoup évolué, et aujourd'hui deviennent de plus en plus complexes, intelligents et utilisés dans plusieurs domaines. Il existe plusieurs types de robots conçus pour des tâches bien spécifiques. Ces dernières sont associées généralement à des travaux qui peuvent être pénibles, fatigants, dangereux ou irréalisables par l'homme. Citons à titre d'exemple, ceux devant être exécutés dans les environnements hostiles dont l'air est irrespirable, à savoir, dans

les centrales nucléaires, dans l'espace, dans les océans, ou certains travaux répétitifs faisant intervenir les capacités intellectuelles de l'être humain. Diverses raisons ont, également, contribué au développement de l'intelligence artificielle et de la robotique.

Dans la plupart des processus industriels, il est indispensable de maîtriser certains paramètres physiques. En automatique, lorsque nous souhaitons atteindre une certaine vitesse, température, position, angle..., il est souvent nécessaire d'avoir recours à un asservissement, c'est à dire un système capable d'atteindre et de maintenir une consigne en utilisant une mesure. Les nouvelles applications des robots offrent des performances très élevées notamment en termes de précision. Toutefois, elles demandent des coûts de plusieurs milliers de dollars par servomoteur. En revanche, la finesse et la précision des mouvements sont souvent une limite dans les robots à faible coût en raison de la qualité des servomécanismes, de la qualité des encodeurs et des conditions variantes de l'environnement d'action du robot. Ceci limite leur déploiement à des prix accessibles au grand public.

Les techniques de contrôle dont le contrôle proportionnel-intégral-dérivée (PID), et plusieurs types de contrôleurs PID non linéaires (NPID) ont été présentés et utilisés efficacement. Bohn et Atherton [3] ont développé un contrôleur NPID avec un schéma anti-emballement (anti-windup) qui peut résoudre le problème d'enroulement de l'intégrateur. Han [4] a proposé un nouveau contrôleur NPID avec une structure simple, dans lequel trois paramètres de contrôleur étaient des fonctions de puissance concernant l'erreur, l'intégrale de l'erreur et la dérivée de l'erreur, respectivement. Afin d'améliorer la capacité de rejet des perturbations du contrôle PD linéaire lorsqu'il était dans un système d'ordre deux typique. Xu et al. [5] ont proposé un contrôleur PD non linéaire (NPD) et l'ont appliqué au contrôle de force des robots. Armstrong et al. [6] ont étendu le contrôleur proposé par Xu et al. à un

système à trois ordres qui a prouvé la stabilité asymptotique du système en boucle fermée par la méthode de Lyapunov. De plus, avec le développement de la théorie du contrôle, des stratégies de contrôle intelligentes telles que le contrôle flou et le système expert sont combinées avec le contrôleur PID conventionnel, et les performances du contrôleur PID conventionnel sont considérablement améliorées.

Le contrôle par Logique floue est l'un des résultats les plus importants de la recherche dans le domaine de l'intelligence artificielle à faible complexité [46]. Ce type de contrôleur a été introduit efficacement dans une vaste gamme d'applications, de la reconnaissance vocale et d'image aux grille-pains et aux transmissions automobiles.

La logique floue est une technique pour incarner la pensée humaine dans un système de contrôle [47-48]. Le contrôle flou a été principalement appliqué au contrôle de processus à travers des performances linguistiques floues, car il donne les meilleures performances dynamiques ainsi que la réduction des erreurs.

Lorsque le contrôleur PID est utilisé dans le contrôle de position pour contrôler le système d'asservissement, certains obstacles apparaissent tels que les comportements en termes de système de non-linéarité, de temps de réponse et enfin d'objectifs d'ingénierie tels que le coût et la fiabilité. Les non-linéarités et le fait de ne pas connaitre certains paramètres sont de facteurs qui motivent nombreux chercheurs à utiliser la théorie et les techniques de contrôle non conventionnelles. Ces paramètres peuvent ne pas être estimés avec précision en l'absence de données expérimentales fiables, et les retards présents dans le processus du système peuvent compliquer la réalisation d'un contrôle de haute performance. Le contrôleur à logique floue est utilisé dans le système de contrôle lorsque le modèle mathématique du

processus concerné est vague ou présente des incertitudes. Un avantage d'utiliser la commande à logique floue est que le contrôleur développé peut gérer un système de plus en plus complexe. Il peut également se mettre en œuvre sans connaissances précises de la structure du modèle de système dynamique. Néanmoins, le problème essentiel est d'étudier le problème du robot manipulateur sous deux aspects : le premier est la modélisation mathématique du manipulateur et des actionneurs, qui comprend une analyse pour la cinématique directe, la cinématique inverse et la modélisation du servomoteur car c'est une question importante dans un robot manipulateur. Le deuxième problème concerne la conception d'un contrôleur pour le mouvement du robot manipulateur afin de répondre à l'exigence de l'entrée de la trajectoire souhaitée avec des valeurs d'erreur et de perturbation appropriées.

1.3 Objectifs

Dans ce travail de recherche, nous nous sommes concentrés à déterminer la fonction permettant de corriger la commande en fonction de la consigne initiale et de l'erreur mesurée, tout en nous intéressant aux commandes; afin d'obtenir de meilleures performances et plus de robustesse pour générer un mouvement doux dans les bras robotisés à faible coût.

La précision (p.ex. manipulation des objets de petite taille), la rapidité (p.ex. dans une chaine de montage), et la finesse (p.ex. applications médicales ou manipulation de produits dangereux) des actions et des mouvements ont été alors l'objet d'analyse dans ce mémoire de maîtrise sur l'asservissement de la position et la vitesse des bras robotisés. Ayant dans l'esprit l'utilisation finale des robots à faible coût, nous analyserons dans ce mémoire la commande PID (classique) et la commande par Logique Floue (moderne) et présenterons des

résultats de simulation et expérimentaux des tests réalisés avec chaque méthode. Cette analyse est faite dans le contexte d'un robot manipulateur à faible coût que nous avons construit au laboratoire.

1.4 Méthodologie

Dans tout projet de recherche, la mise en position de la problématique et des objectifs fait suite à l'analyse de la littérature et à l'étude bibliographique. Nous continuons ainsi la recherche à partir des résultats antérieurs, nous pouvons ainsi ouvrir de nouvelles pistes de la recherche. En même temps, c'est un moyen d'analyse critique des travaux de recherches précédents qui rapportent les diverses méthodes de modélisation et les multiples techniques, classiques et modernes, de commande des manipulateurs robotiques flexibles présentées dans la littérature du domaine étudié.

Puis, et dans le but de compléter l'étude théorique par une étude expérimentale de différentes stratégies de contrôle, nous disposons d'un manipulateur constitué d'un moteur à courant continu, un bras flexible, un encodeur qui mesure l'angle. La description du montage suivie d'une étude détaillée sur les différentes méthodes de modélisation développées est présentée au troisième chapitre de ce rapport.

Les diverses techniques de commande peuvent être regroupées en deux catégories : les méthodes classiques et les méthodes modernes. Les techniques de contrôle classiques comme le contrôle proportionnel-intégral-dérivé (PID), les techniques de contrôle modernes comme le contrôle par Logique Floue. Dans ce chapitre sont aussi présentés : les résultats de simulation et les résultats expérimentaux obtenus suite à l'application de ces techniques de commande. Nous clôturons, ce chapitre par une comparaison entre les résultats obtenus par

les méthodes classiques et ceux obtenus par les techniques modernes. Après cette comparaison, nous déterminerons la commande la plus robuste, et proposerons une application sous LabVIEW pour contrôler le bras robotique.

Finalement, ce travail se termine par une conclusion générale qui résume les résultats obtenus et une brève description du travail futur pouvant avoir lieu dans ce domaine de recherche comme une suite à ce projet.

Chapitre 2 - État de l'Art

2.1 Introduction

Dans le domaine passionnant de la robotique, les exigences dans la commande des machines deviennent de plus en plus importantes en fonction des particularités de chaque application. L'utilisation de la robotique consiste à l'automatisation de nombreux secteurs de l'activité humaine afin, par exemple, d'augmenter la productivité dans les entreprises des domaines médical, spatial, industriel, militaire et dans le domaine de l'agriculture. Celle-ci permet de manipuler, avec beaucoup de précision et par conséquent, de manière plus sûre, divers produits ou objets, neutres ou dangereux.

Dans ce chapitre, nous donnerons un bref historique sur l'évolution de la robotique industrielle et un aperçu non exhaustif sur les robots.

2.2 Définitions

2.2.1 Définition d'un robot

Appareil automatique capable de manipuler des objets ou d'exécuter des opérations selon un programme fixe, modifiable ou adaptable [7].

2.2.2 Définition de la robotique

La branche de la technologie qui traite la conception, la construction, l'exploitation et l'application des robots [7].

2.3 Histoire de la robotique

Soldats, musiciens, ou aides-soignants, les robots sont de plus en plus présents et humanisés, ils nous fascinent mais également nous inquiètent comment sont-ils nés et quel est leur futur. La Figure 2-1 montre quelques exemples de robots de fiction.

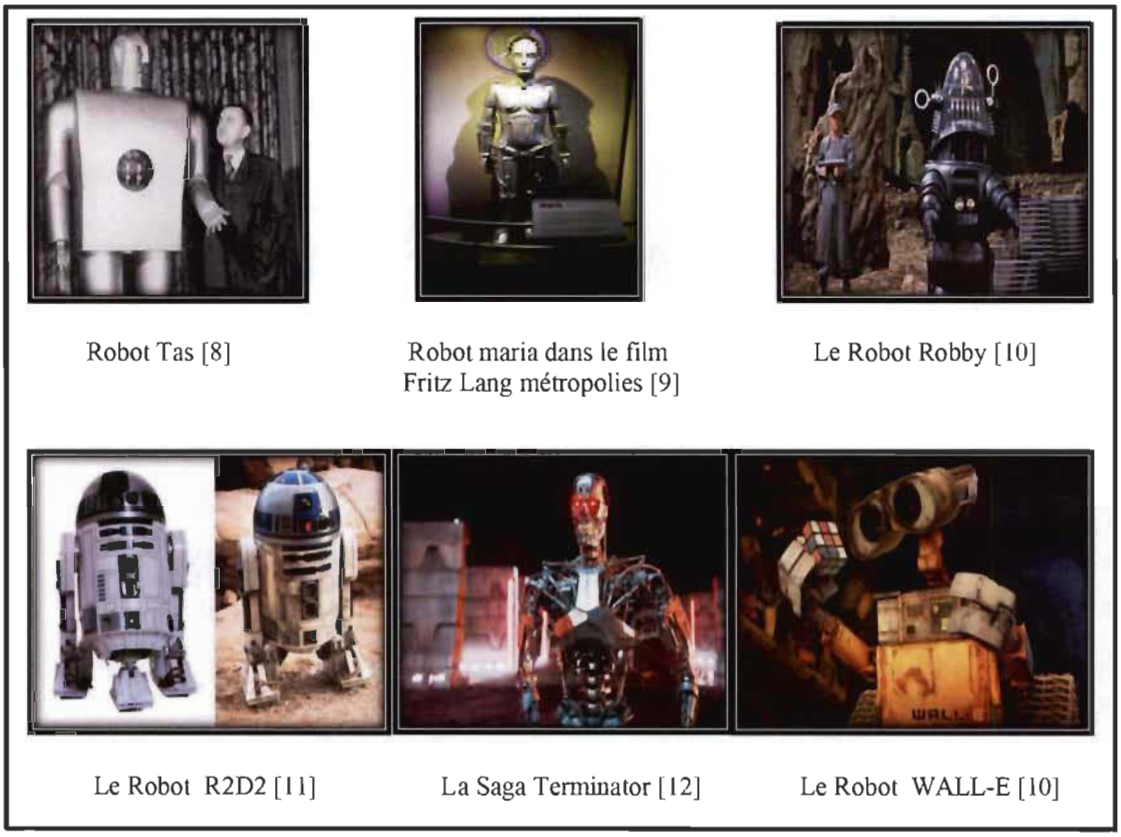

Figure 2-1 Les robots de fiction.

Les robots de fiction : C'est en 1920 qu'apparait pour la première fois le mot robot dans une pièce de théâtre de science-fiction écrite par l'écrivain tchèque Karel Capek [8]. Le terme robot dérive de mot slave « robot tas » qui signifie travail corvée utilisé pour nommer un des personnages comme un serviteur mécanique à l'aspect humain.

En 1927, d'autres figures robotisées apparaissent, les spectateurs découvrent le robot Maria dans le film de Fritz Lang Metropolis [9]. Avant de devenir des acteurs du monde réel, les robots sont en premier lieu des stars de la littérature des arts graphiques et du cinéma souvent pleins d'humanité comme Robby [10], le robot WALL-E [6] ou R2D2 [11]; ils sont aussi parfois les destructeurs et des ennemis, parfois amis, du genre humain comme les robots de la saga Terminator [8].

Aux années 1950, les mathématiciens et les ingénieurs se sont intéressés avec les robots de manipulation au sujet tel l'américain Joseph Engelberger [13]. C'est ainsi qu'une nouvelle science a vu le jour: la robotique; le robot est désormais une machine qui bouge, commandée par un ordinateur. Elle se distingue des autres machines par une capacité propre à décider de son mouvement. Au fil des décennies, le robot a trouvé sa place dans nombreux domaines d'application industriels, d'exploration spatiale et maritime, médecine etc…

En 1969, le premier robot du monde réel s'appelle UNIMATE (Figure 2-2), il s'agit d'un robot industriel crée par Victor Scheinman de l'Université de Stanford qui inventa un bras articulé sur 6 axes. Sheinman venda son concept à UNIMATE plus tard [13].

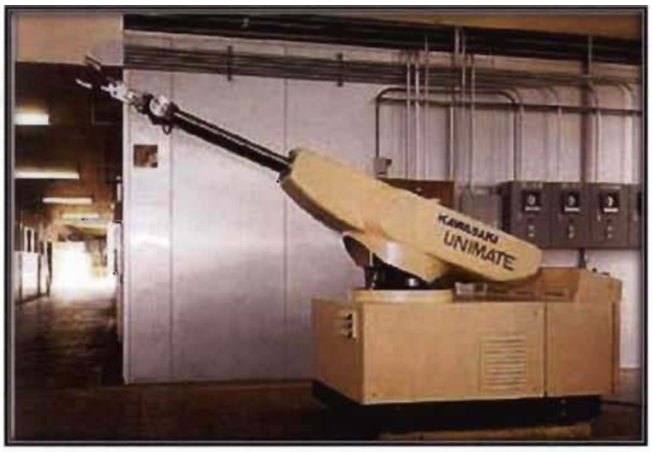

Figure 2-2 UNIMATE [13], premier robot industriel.

En 1996, le robot part dans l'espace, Sojourner (Figure 2-3) explore la planète Mars, découvre son sol et l'analyse des roches [14].

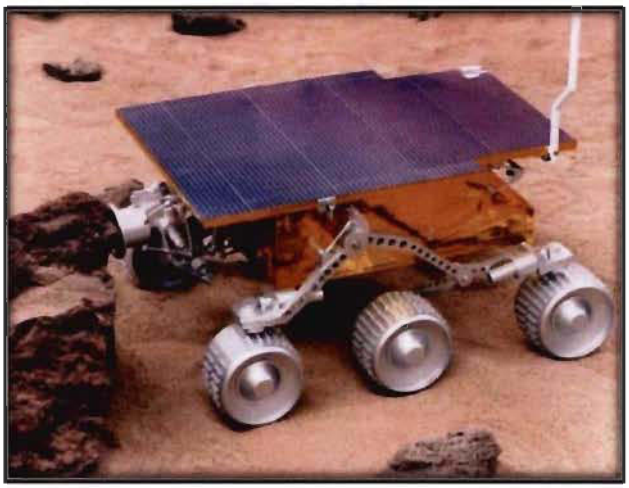

Figure 2-3 Le premier robot mobile (Rover) sur la planète Mars [14].

En 2000, le Da VinCi (Figure 2-4) est un robot assistant du chirurgien qui apparaît dans les blocs opératoires [15].

Figure 2-4 Une table d'opération surmontée des bras d'un *Da Vinci* [15].

Par la suite, aujourd'hui, des robots mobiles sont conçus pour intervenir dans des milieux hostiles comme Big Dog (Figure 2-5) un quadrupède porteur de charges lourdes testé par l'armée américaine en Afghanistan [16].

Figure 2-5 Big Dog un Quadrupède [16].

Tel que montré par la Figure 2-6, les expéditions annuelles de robots industriels ont vu une croissance assez importante au cours des dernières années. Plus particulièrement, en provenance des pays asiatiques.

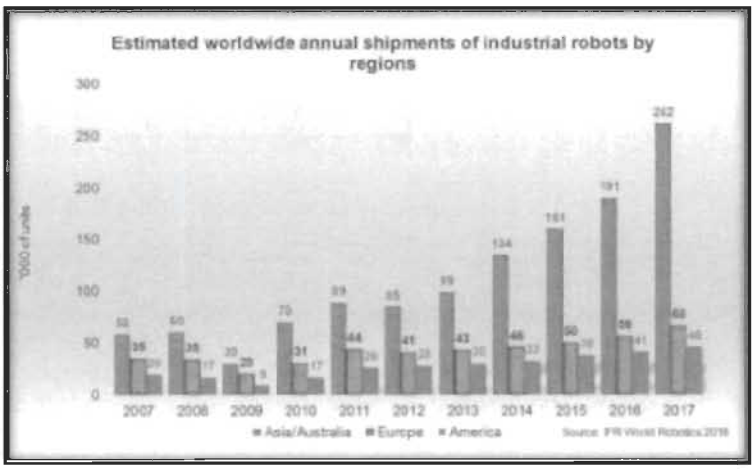

Figure 2-6 Graphique représentant l'estimation des expéditions annuelles mondiales de robots industriels par régions [17].

2.4 Types de robots

Il existe deux grandes catégories de robots : les robots mobiles et les robots manipulateurs. Toutefois, il peut avoir des configurations hybrides mobiles+manipulateurs pour des applications spécifiques, p.ex., agricoles, d'exploration, sauvetage.

2.4.1 Robots mobiles

Les robots mobiles ont la capacité de se déplacer dans leur environnement et ne sont pas fixés à un seul emplacement physique. Les robots mobiles peuvent être « autonomes », ce qui signifie qu'ils sont capables de naviguer dans un environnement incontrôlé sans avoir besoin de dispositifs de guidage physiques ou électromécaniques. Un exemple est illustré dans la Figure 2-7.

Figure 2-7 Robot mobile OMRON [18].

2.4.2 Robots manipulateurs

Un bras manipulateur est le bras d'un robot généralement programmable, avec des fonctions similaires à un bras humain. Les liens de ce manipulateur sont reliés par des axes permettant, soit de mouvement de rotation (comme dans un robot articulé) et/ou de translation (linéaire) de déplacement.

Dans le cas d'une imitation complète d'un bras humain, un bras manipulateur a donc 3 mouvements de rotation et 3 mouvements de translation sur son élément terminal.

Il peut être autonome ou contrôlé manuellement et peut être utilisé pour effectuer une variété de tâches avec une grande précision selon la qualité des servomécanismes et des méthodes de contrôle employées [19]. Un exemple est illustré par la Figure 2-8.

Figure 2-8 Bras Manipulateur [19].

2.5 Constituant d'un robot

Tel que montré par la Figure 2-9, nous distinguons classiquement 4 parties principales dans un robot manipulateur [20]: les actionneurs, le système mécanique articulé et organe terminal, les capteurs et la commande.

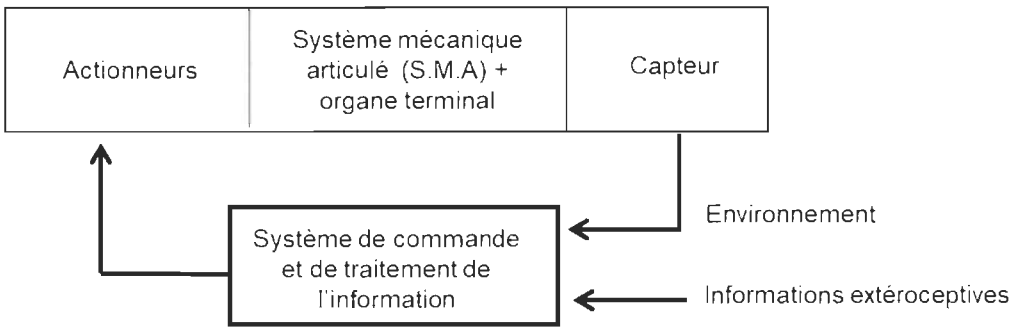

Figure 2-9 Éléments d'un robot manipulateur.

2.5.1 Les actionneurs

Les composants permettant de mettre en mouvement les organes des machines sont appelés actionneurs. Ce sont essentiellement des moteurs et des vérins. Ils produisent de l'énergie mécanique à partir d'énergie électrique, hydraulique ou pneumatique, mais sont presque toujours contrôlés par des signaux de commande électriques. Avec l'évolution technologique progressive dans le domaine de l'informatique et l'électronique tel que les micro-ordinateurs, l'automate programmable, les microprocesseurs et les microcontrôleurs ainsi que les composantes semi-conductrices, etc. ; les actionneurs électriques sont de plus en plus utilisés dans le domaine de la robotique [20].

2.5.2 Système mécanique articulé

Le système mécanique articulé (SMA) est un mécanisme ayant une structure plus ou moins proche de celle du bras humain, il peut être représenté par une architecture composée de plusieurs chaînes de corps rigides assemblés par des liaisons appelées articulations, généralement les uns à la suite des autres où chaque solide est mobile par rapport au précédent. Cette mobilité s'exprime en termes de degrés de liberté (d.d.l) qui est par définition le nombre de mouvements indépendants possibles d'un solide Sn-1 par rapport au solide qui lui est directement relié Sn (Fig. 2-10). Le rôle du SMA est d'amener l'organe terminal dans une situation (position et orientation) donnée, selon des caractéristiques de vitesse et d'accélération données. Sa motorisation est réalisée par des actionneurs électriques, pneumatiques ou hydrauliques qui transmettent leurs mouvements aux articulations par des systèmes appropriés [20].

Figure 2-10 Schématisation d'un système mécanique articulé [21].

2.5.3 Les articulations

Une articulation lie deux corps successifs en limitant le nombre de degrés de liberté de l'un par rapport à l'autre, peut se ramener à une combinaison d'articulations prismatiques ou rotoïdes.

2.5.3.1 Articulation Rotoïde

Il s'agit d'une articulation de type pivot, notée R, réduisant le mouvement entre deux corps à une rotation autour d'un axe qui leur est commun. La situation relative entre les deux corps est donnée par l'angle autour de cet axe (Fig.2.11).

Figure 2-11 Représentation d'une articulation rotoïde [20].

2.5.3.2 Les articulations prismatiques

Il s'agit d'une articulation de type glissière (translation), notée P, réduisant le mouvement entre deux corps à une translation le long d'un axe commun. La situation relative entre les deux corps est mesurée par la distance le long de cet axe (Fig.2.12).

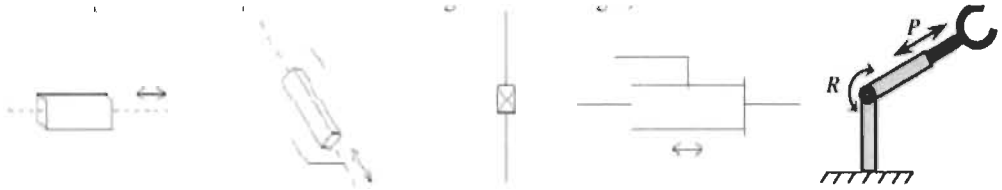

Figure 2-12 Représentation d'une articulation prismatique [20].

2.5.4 Organe terminal

Nous regroupons tous les dispositifs destinés à manipuler des objets (dispositifs de serrage, dispositifs magnétiques, à dépression, ...), ou à les transformer (outils, torche de soudage, pistolet de peinture, ...). En d'autres termes, il s'agit d'une interface permettant au robot d'interagir avec son environnement. Un organe terminal peut être multifonctionnel, au sens où il peut être équipé de plusieurs dispositifs ayant des fonctionnalités différentes. Il peut aussi être monofonctionnel, mais interchangeable. Un robot, enfin, peut-être multi-bras: chacun des bras portant un organe terminal différent. On utilise, indifféremment, le terme organe terminal, préhenseur, outil ou effecteur pour nommer le dispositif d'interaction fixé à l'extrémité mobile de la structure mécanique [20].

2.5.5 Les capteurs

Les informations extéroceptives ou plus simplement les perceptions permettent de gérer les relations entre le robot et son environnement. Les organes de perception sont des capteurs

dits proprioceptifs lorsqu'ils mesurent l'état interne du robot (positions et vitesses des articulations) et extéroceptifs lorsqu'ils recueillent des informations sur l'environnement (détection de présence, de contact, mesure de distance, vision artificielle) [20].

Les principales sortes de capteurs sont :

- Les sondeurs (ou télémètres) à ultrason ou Laser. Ces derniers sont à la base des scanners laser permettant à l'unité centrale du robot de prendre « conscience » de son environnement en 3D.
- Les caméras sont les yeux des robots. Il en faut au moins deux pour permettre la vision en trois dimensions. Le traitement automatique des images pour y détecter les formes, les objets, voire les visages, demande en général un traitement matériel car les microprocesseurs embarqués ne sont pas assez puissants pour le réaliser.
- Les roues codeuses permettent au robot se déplaçant sur roues, des mesures de déplacement précises en calculant les angles de rotation (information proprioceptive) [22].

2.5.6 La commande

La partie commande synthétise les consignes des asservissements pilotant les actionneurs, à partir de la fonction de perception et des ordres de l'utilisateur.

2.6 Classification des robots

La classification des systèmes robotiques est difficile, car il existe de nombreux critères pour leurs descriptions. Toutefois, nous allons les classer selon deux structures qui sont les suivantes :

• Point de vue fonctionnel.

• Point de vue géométrique.

2.6.1 Point de vue fonctionnel

Le nombre de classes et les distinctions entre celles-ci varient de pays à pays (6 classes au Japon, 4 en France). L'A.F.R.I. distingue 4 classes illustrées ci-dessous [23] :

2.6.1.1 Manipulateur à commande manuelle

La figure suivante (Figure 2-13) représente un manipulateur à commande manuelle d'un système de télé chirurgie.

Figure 2-13 Système de télé chirurgie [15].

2.6.1.2 Manipulateur automatique à cycles préréglés

Le réglage se fait mécaniquement par cames, butées. La commande peut se faire par automate programmable (Figure 2-14) ; Nous pouvons ainsi faire la distinction entre manipulateurs à cycle fixe et entre manipulateurs à cycle programmable.

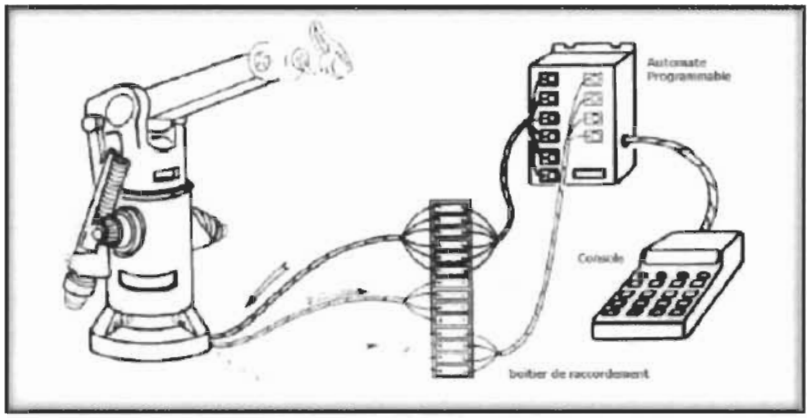

Figure 2-14 Manipulateur à cycle préréglé [23].

2.6.1.3 Robots programmables

Les mouvements d'un robot industriel sont programmés à travers deux grandes méthodes :

La méthode par apprentissage est la première historiquement apparue. Elle consiste à créer les trajectoires en faisant mémoriser au robot des points correspondant à des coordonnées cartésiennes et qui détermineront sa position. Celle-ci s'effectue directement sur le robot en utilisant le boitier de contrôle.

La seconde méthode, plus récente, est la programmation hors-ligne. Sur un ordinateur de travail dédié, l'opérateur pourra programmer la prochaine tâche via un logiciel de programmation hors-ligne en important un modèle CAO grâce auquel il pourra générer les mouvements. Tel que montré dans la Figure 2-15, il pourra par la suite visualiser le résultat de sa programmation grâce à un simulateur intégré qui est une représentation virtuelle de l'environnement de travail du robot avec toutes ses composantes [24].

Figure 2-15 Programmation des mouvements d'un bras robotique sur un ordinateur [24].

2.6.1.4 Robots intelligents

Nous trouvons actuellement des robots de seconde génération qui sont capables d'acquérir et d'utiliser certaines informations sur leur environnement (systèmes de vision, détecteurs de proximité, capteurs d'efforts, ...). Nous étudions des robots de troisième génération, capables de comprendre un langage oral proche du langage naturel et de se débrouiller de façon autonome dans un environnement complexe, grâce à l'utilisation de l'intelligence artificielle [25]. Un exemple de robot « intelligent » est montré dans la Figure 2-16.

Figure 2-16 Robot intelligent [44].

2.6.2 Classification géométrique

Il existe différentes architectures du porteur : la structure cartésienne (PPP), la structure cylindrique (RPP ou PRP), la structure sphérique ou polaire (RRP), la structure dite SCARA (RRP) et enfin la structure anthropomorphe (RRR).

2.6.2.1 Structure cartésienne (PPP)

Les caractéristiques d'une structure cartésienne (PPP) sont : trois axes, deux à deux en série avec trois degrés de liberté, une très bonne précision et une grande lenteur. Le volume de travail est un parallélépipède dont les dimensions sont les translations permises par les trois liaisons prismatiques [26]. Un schéma simplifié d'une structure cartésienne (PPP) est présenté dans la Figure 2-17.

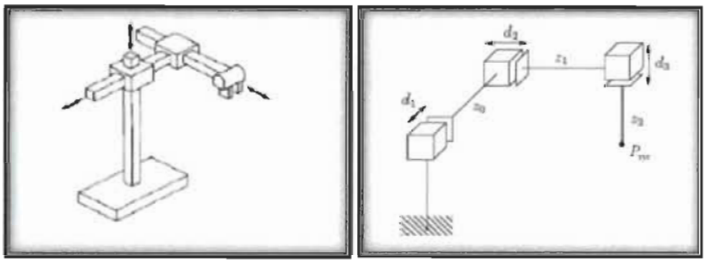

Figure 2-17 Structure PPP [45].

2.6.2.2 La structure cylindrique (RPP) ou (PRP)

Les caractéristiques d'une structure cylindrique (RPP ou PRP) sont : trois axes non perpendiculaires en série avec trois degrés de liberté et une grande rapidité. Le volume de travail est un cylindre plein ou creux, autrement dit un tore à section rectangulaire, dont la hauteur L est la translation permise par une liaison prismatique [27]. Un schéma simplifié d'une structure cylindrique (RPP ou PRP) est présenté dans la Figure 2-18.

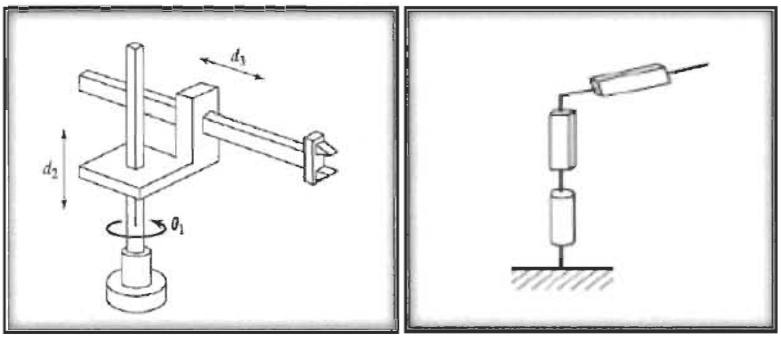

Figure 2-18 Structure RPP [45].

2.6.2.3 La structure sphérique ou polaire à axe de rotation orthogonale (RRP)

Le volume de travail est une sphère creuse, dont les rayons intérieur et extérieur sont fixés soit par la disposition de la liaison prismatique et la translation qu'elle permet, soit par les longueurs des deux parties du bras [28]. Un diagramme simplifié d'une structure polaire à axe de rotation orthogonale (RRP) est présenté dans la Figure 2-19.

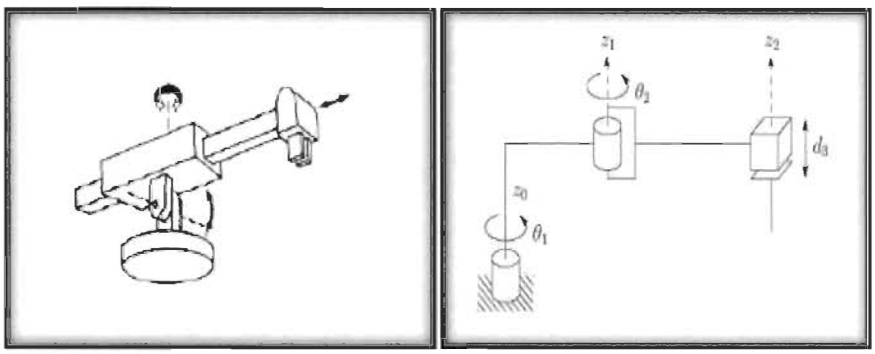

Figure 2-19 Structure RRP [45].

2.6.2.4 La structure dite SCARA

Les caractéristiques d'une structure SCARA (Selective Compliance Articulated Robot for Assemblage) sont : 3 axes en série cylindrique (RRP) ayant trois degrés de liberté. Les

particularités de cette structure sont qu'elles sont précises et très rapides [28]. Un exemple de structure SCARA est illustré dans la Figure 2-20.

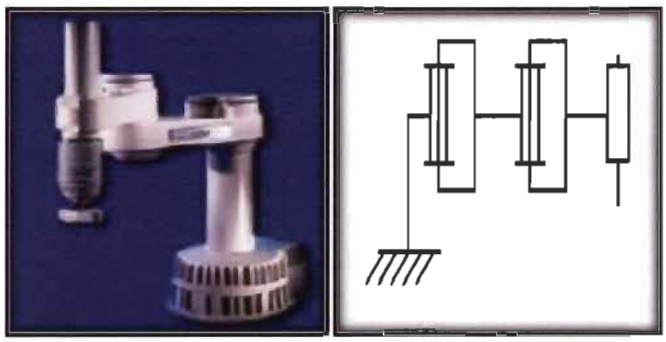

Figure 2-20 Structure SCARA [45].

2.6.2.5 La structure anthropomorphe (RRR)

Cette structure à une architecture plus généraliste reproduisant le bras humain. Son enveloppe de travail a une cinématique et dynamique complexe, mais sa configuration est plus flexible [29]. Un diagramme simplifié de la structure anthropomorphe (RRR) est présenté dans la Figure 2-21.

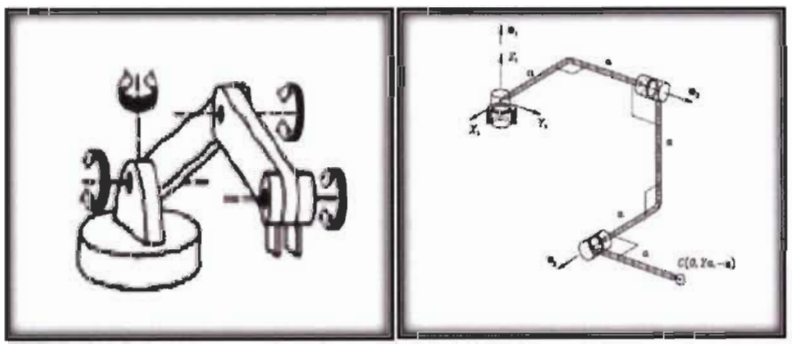

Figure 2-21 Structure RRR [45].

2.7 Méthodes de commande de servomoteurs

La méthode la plus courante pour réaliser un asservissement de servomoteurs est la commande PID qui est aujourd'hui l'un des asservissements les plus utilisés et ce pour plusieurs raisons. Tout d'abord, il est très simple à mettre en place et s'avère efficace pour la plupart des systèmes réels. Ensuite, le calcul des coefficients laisse le choix entre plusieurs méthodes de difficulté croissante. De plus, c'est une méthode expérimentale très facile à mettre en place, permet d'obtenir rapidement des coefficients corrects pour des systèmes ne nécessitant pas de très grandes précisions dans l'asservissement. Cependant, il est important de noter que ce type d'asservissement est limité par un certain nombre de contraintes. En effet, il peut s'avérer inefficace pour certains systèmes qui contiennent du bruit (coefficient Dérivé) ou qui ne sont pas linéaires (l'asservissement PID étant linéaire, la non-linéarité d'un système peut entraîner des pertes de performance et des instabilités).

Il existe diverses méthodes pour commander une boucle d'asservissement. La méthode moderne de contrôle par logique floue réunit un certain nombre d'avantages et d'inconvénients. Les avantages essentiels sont [30] :

- Pas de modèle mathématique requis pour le procédé à réguler ;
- La théorie est simple et peut s'appliquer à des systèmes complexes ;
- La commande peut facilement être auto-adaptative ;
- La commande floue conduit à un code informatique clair et lisible.

Par contre, les inconvénients sont :

- La technique de réglage est totalement empirique ;

- La précision du réglage est souvent peu élevée ;
- La cohérence des inférences n'est pas garantie à priori ; il est possible l'apparition de règles d'inférence contradictoires.

Nous nous intéressons alors à l'étude expérimentale comparative de la commande PID et la Logique Floue dans l'objectif d'assurer une réponse acceptable pour des signaux de consigne définis en fonction du temps.

2.8 Conclusion

Dans ce chapitre, nous avons donné un aperçu général sur la robotique. Nous avons présenté, également, une description sur le domaine de la robotique en mettant l'accent sur les domaines applicatifs des robots industriels, l'historique des robots, l'architecture mécanique générale des bras manipulateurs, leurs structures, leurs utilisations, leurs différents types, leurs classifications ainsi que leurs domaines d'application ce qui va nous servir pour la construction de notre bras. Nous avons fini le chapitre par l'étude des avantages et des inconvénients des deux méthodes de commande utilisées selon littérature. Nous allons par la suite commencer la partie technique et la plus concrète de cette étude.

Chapitre 3 - Description et modélisation du bras manipulateur

3.1 Introduction

L'une des premières étapes dans la réalisation d'un robot consiste à définir le mode de locomotion à mettre en œuvre ainsi que les différents éléments qui le composent, ceci se matérialise par l'étude des actionneurs des chaînes cinématiques associées et aussi par l'étude des capteurs qui constituent la source d'information.

Dans ce chapitre, nous présenterons l'architecture mécanique générale du bras manipulateur et nous donnerons également des notions théoriques qui nous aideront à l'aboutissement de la modélisation de notre système.

3.2 Description de la partie mécanique du notre bras manipulateur

3.2.1 La base

La base est le socle fixe du bras. Elle possède un servomoteur qui permet au bras de tourner un tour complet sur un angle de 360° autour de la verticale.

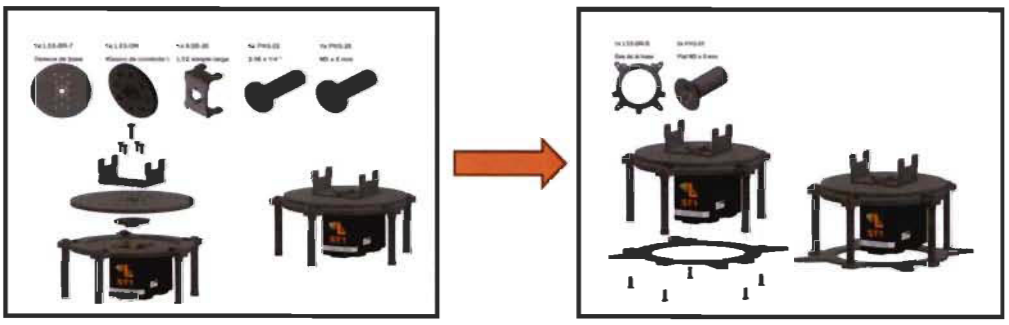

Figure 3-1 Base du robot [21].

3.2.2 L'épaule

Elle est liée à la base à travers la deuxième articulation (servomoteur), qui permet la rotation de 0° à 180°.

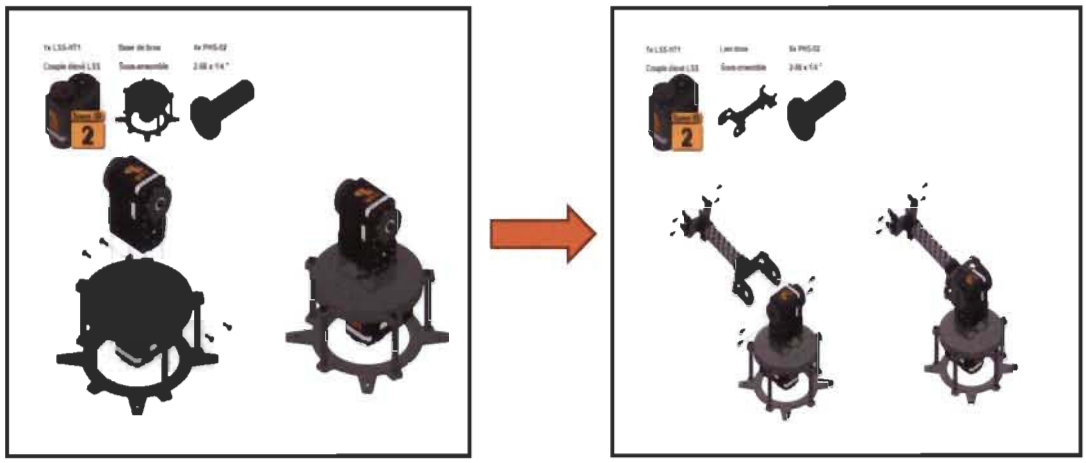

Figure 3-2 Épaule du robot [21].

3.2.3 Le coude

Il est lié à l'épaule par la troisième articulation, qui permet la rotation de 0° à 180°.

Figure 3-3 Le coude du robot [21].

3.2.4 La pince

La pince est l'organe terminal de notre robot manipulateur, un servomoteur qui permet l'ouverture et la fermeture de la pince de rotation de 0° à 180°.

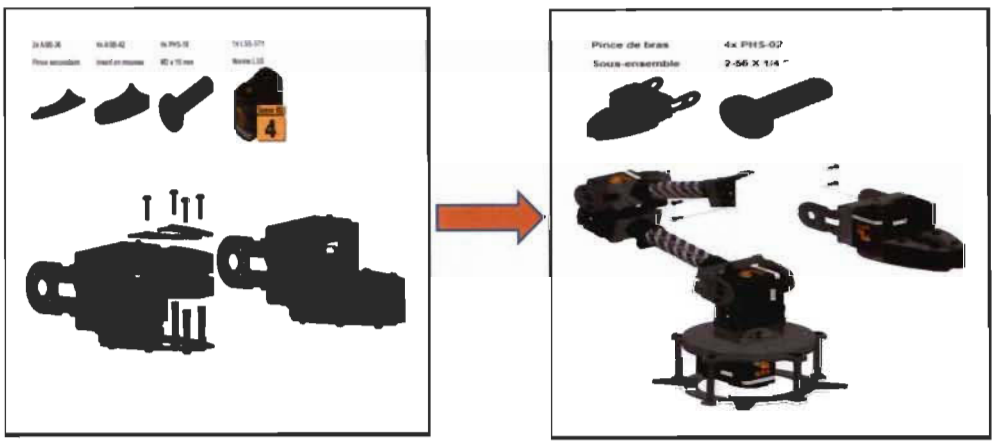

Figure 3-4 La pince du robot [21].

3.3 Le bras manipulateur assemblé

Après avoir vu les différentes pièces constituant le bras manipulateur, nous sommes passé à leur assemblage. La figure ci-dessous montre le bras manipulateur après l'assemblage.

Figure 3-5 Le robot assemblé [21].

3.4 Description des parties constitutives du bras manipulateur

Pour que le bras manipulateur puisse exécuter une tâche, il a besoin des actionneurs et des transmetteurs, ces derniers servent à transmettre de l'énergie des actionneurs des articulations rotoïdes. De plus, pour asservir ces actionneurs, nous avons besoin également de capteurs.

3.4.1 Les actionneurs du bras manipulateur

Il s'agit d'organes qui permettent au robot d'exécuter une tâche. Leur but est de produire assez de force pour provoquer le mouvement du robot. Nous avons choisi d'utiliser comme actionneurs quatre servomoteurs qui sont des moteurs à courant continu (MCC), asservis en position à l'aide d'un capteur de position et d'un circuit électronique interne au moteur. Les servomoteurs de positionnement angulaire permettent de déplacer précisément un objet dans une plage de 0° à 360° [21].

Ils sont pilotés par un fil de commande et alimentés par deux autres fils, le premier est relié à l'alimentation positive +6 ou +12 V selon le servo, le deuxième est relié à la masse (GND) [31].

Figure 3-6 Servomoteur intelligent Lynxmotion (LSS) [21].

3.4.1.1 Les paramètres techniques des servomoteurs

Les paramètres des servomoteurs utilisés dans le bras robot sont présentés au Tableau 3-1 [21].

Tableau 3-1 Paramètres techniques des servomoteurs utilisés

Référence / Spécifications	ST1	HT1
Tension	6V à 12.6	6V à 12.6
Couple Max. couple dynamique 12V (~ 25%)	2,8 kg-cm	5,8 kg-cm
Angle de fonctionnement	Jusqu'à 360 °	Jusqu'à 360 °
Poids	58,0 g	80,0 g
Vitesse Maximal	360 ° / s	360 ° / s

3.4.1.2 Transmission entre articulations : engrenage

Un moteur à courant continu tourne normalement à des vitesses de l'ordre de plusieurs milliers de tours par minute et produit un couple très faible. Afin de diminuer cette vitesse de rotation tout en augmentant significativement le couple disponible, nous plaçons entre l'axe du moteur et l'axe articulaire un réducteur par les transmissions des engrenages qui sont des organes mécaniques situés à l'intérieur de la partie supérieure du boîtier. Tel que montré dans la Figure 3-7. Le train d'engrenages est composé d'une variété d'engrenages droits en métal [21].

Figure 3-7 Schéma d'un engrenage du servomoteur Lynxmotion [21].

3.4.2 Les capteurs

L'élément qui donne l'information de rétroaction au servomoteur est le capteur de position. Notre capteur est un capteur absolu magnétique couplé sur la partie avant du moteur (voir Figure 3-8). Le rôle des capteurs est de gérer les relations entre le robot et son environnement, ils permettent ainsi de contrôler plus facilement les tâches que nous leur ordonnerons.

Figure 3-8 Capteur absolu magnétique [21].

3.5 Modélisation

La modélisation des robots consiste à établir un modèle mathématique. Outre une fonction générale d'aide à la conception, la modélisation a de multiples utilisations pour la prévision (ou anticipation) des mouvements, l'adaptation des actionneurs, la planification des taches, l'établissement des lois de commande, l'incorporation du robot dans des simulations informatiques, etc.

Dans le langage courant, la modélisation précède la simulation sans que nous puissions faire une séparation nette entre ces deux activités. Il est souvent acceptable de se contenter d'une modélisation simplifiée dans laquelle nous ne tenons pas compte des aspects qui sont, ou paraissent, secondaires tels que : les vibrations, les déformations élastiques, les jeux mécaniques, les tolérances de fabrication, etc. La modélisation des robots manipulateurs nécessite le calcul de certains modèles mathématiques, tels que : Le modèle géométrique et le modèle cinématique [49].

3.5.1 Repères et référentiels

Comme il est illustré dans la Fig.3.9, pour repérer un point M dans l'espace, il faut 3 coordonnées pour le définir parce que l'espace contient 3 dimensions, ce dernier est représenté par les valeurs algébriques des projections sur une base orthonormée [32].

3.5.1.1 Coordonnées cartésiennes

Soit un point fixe O (appelé origine), constitué de trois axes rattachés, à ce repère, nous associons une base orthonormée directe ($\vec{U}x, \vec{U}y, \vec{U}z$). Les vecteurs$\vec{U}x, \vec{U}y, \vec{U}z$ sont alors les vecteurs unitaires des axes OX, OY et OZ respectivement. Les coordonnées cartésiennes sont

les référentiels les plus utilisées en robotique, car elles sont les plus simples pour la mesure des distances.

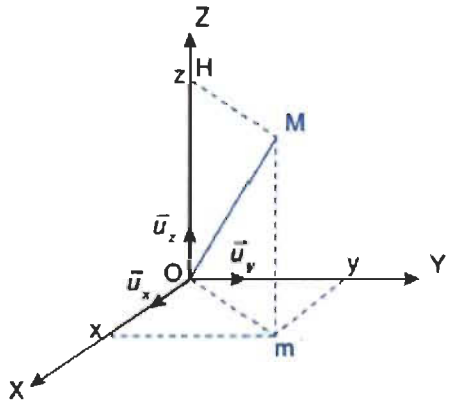

Figure 3-9 Le système de coordonnées cartésiennes [32].

3.5.1.2 Coordonnées cylindriques

La position du point M est définie dans un repère (O, $\vec{U}_p$, $\vec{U}_\theta$, $\vec{U}_z$). Nous introduisons la base ($\vec{U}_p$, $\vec{U}_\theta$, $\vec{U}_z$) orthonormée directe, associée aux coordonnées cylindriques (ρ, θ, z). Les relations entre les coordonnées cylindriques et cartésiennes sont les suivantes [32].

$$\begin{cases} x = \rho \cos(\theta) \\ y = \rho \sin(\theta) \\ z = z \end{cases}$$

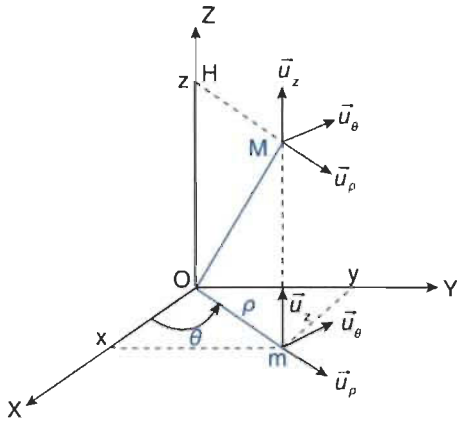

Figure 3-10 Le système de coordonnées cylindriques [32].

3.5.1.3 Coordonnées sphériques

Prenons comme exemple un point M repéré à la surface de la Terre (O, $\vec{U}_p$, $\vec{U}_\theta$, $\vec{U}_\varphi$). La méthode la plus facile pour repérer ce point est de repérer M par 3 coordonnées qui sont:

- La distance depuis O.
- Deux angles de rotation autour de O.

Les relations entre les coordonnées sphériques et cartésiennes sont les suivantes [33] :

$$\begin{cases} x=\rho \sin(\theta)\cos(\varphi) \\ y=\rho \sin(\theta)\sin(\varphi) \\ z = \rho \cos(\theta) \end{cases}$$

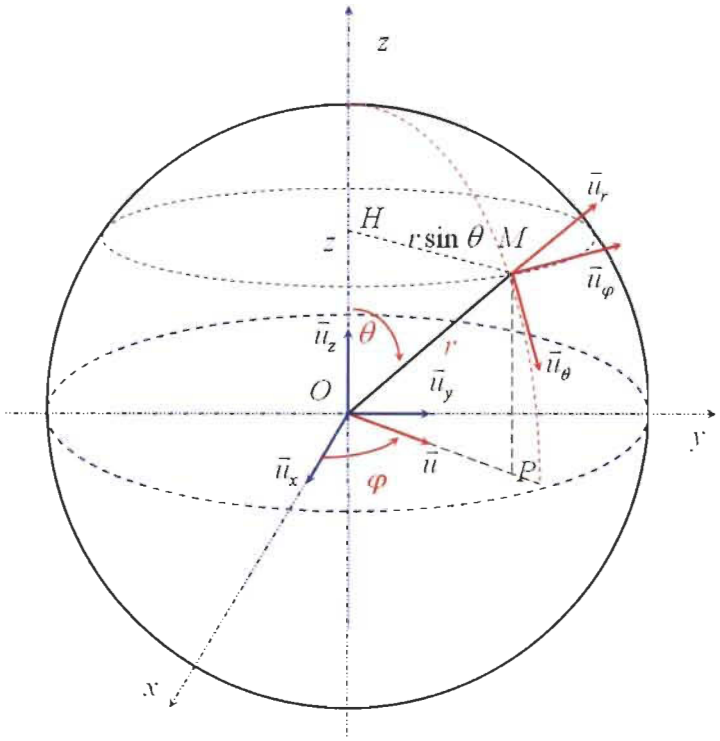

Figure 3-11 Le système de coordonnées sphériques [33].

3.5.1.4 Position et orientation d'un solide

Nous pouvons facilement paramétrer la position d'un point dans un repère avec 3 trois dimensions quel que soit le système de coordonnées utilisé par contre le repérage de la position d'un solide nécessite plus de dimensions. Il faut trois coordonnées pour positionner le centre d'inertie des solides et trois autres coordonnées pour son orientation dans l'espace. Six coordonnées sont donc nécessaires pour placer un objet dans l'espace [33].

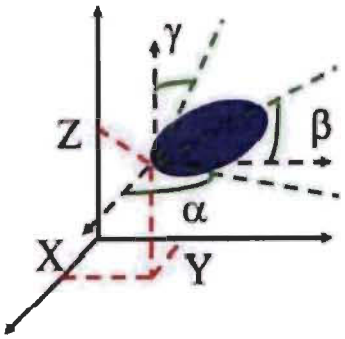

Figure 3-12 Position d'un solide dans l'espace et son repère associé [34].

3.5.2 Modèle Géométrique Direct (MGD)

Le modèle géométrique direct d'un robot permet de calculer les coordonnées opérationnelles donnant la situation de l'organe terminal en fonction des coordonnées articulaires, l'expression de la situation de l'organe terminal du bras manipulateur en fonction de sa configuration est obtenue à l'aide de l'équation suivante :

$$X = f(q) \quad [35]$$

Où X est le vecteur des coordonnées opérationnelles exprimées dans le repère de référence R_0 et q les variables articulaires.

Dans le cas d'une chaine cinématique simple ouverte, il peut être représenté par la matrice $T_{0.n}$ qui se calcule par :

$$T_{0,n} = T_{0,1}(q_1) \times T_{0,2}(q_2) \times \ldots\ldots \times T_{j-1,j}(q_j)$$

La matrice $T_{0,n}$ représente la position et l'orientation, elle est exprimée dans le repère de référence R_0 de l'organe terminal (effecteur) du robot.

3.5.2.1 Paramètre de Denavit-Hartenberg (DH)

Les paramètres DH sont utilisés pour systématiser la modélisation de n'importe quel type de robot série. Elle fut introduite par Jacques DENAVIT et Richard HARTENBERG [35]. En génie mécanique, les paramètres Denavit – Hartenberg (également appelés paramètres DH) sont les quatre paramètres associés à une convention particulière pour attacher des cadres de référence aux maillons d'une chaîne cinématique spatiale, ou manipulateur de robot. Les paramètres DH permettent de simplifier le modèle géométrique, mais également d'établir une norme reconnue par tous.

Afin d'établir les paramètres de DH, chacun des repères est établi en se basant sur trois règles :

- L'axe Zi-1 est selon l'axe de mouvement de l'art-1.
- L'axe Xi-1 est aligné entre les articulations i et (i+1). Si ai est nul, alors Xi-1 est perpendiculaire à Zi-1 et Zi.
- L'axe Yi complète le repère i pour former un repère « main droite ».

Pour passer de Ri-1 a Ri, nous pouvons suivre les quatre paramètres géométriques les suivants :

1- Rotation (Zi-1, θi) Avec, ai : Distance entre Zi-1 et Zi, le long de Xi.

2- Translation (Zi-1, di) αi : Angle entre Zi-1 et Zi, autour de Xi.

3- Translation (Xi, ai) θi : Angle entre Xi-1 et xi, autour de Zi-1.

4- Rotation (Xi, αi) di : Distance de Oi-1 à l'intersection de Zi-1 avec Xi.

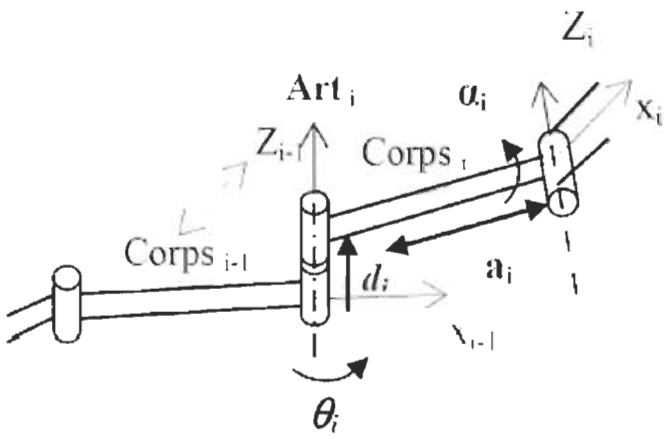

Figure 3-13 Schématisation du lien entre le corps i-1 et i [28].

La matrice de passage d'une articulation est la suivante donc :

$$T_i^{i-1} = Rot\,(x, \alpha i) \times Trans\,(x, di) \times Rot\,(z, \theta i) \times Trans\,(z, ri) \quad (3.1)$$

$$T_i^{i-1} = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & di \\ C\alpha_i.S\theta_i & C\alpha_i.C\theta_i & -S\alpha_i & -r_i.S\alpha_i \\ S\alpha_i.S\theta_i & S\alpha_i.C\theta_i & C\alpha_i & r_iC\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

Pour systématiser la modélisation et simplifier le modèle géométrique de notre bras manipulateur, nous utilisons les paramètres de Denavit Hartenberg (DH) par la relation entre le modèle géométrique direct et le modèle géométrique inverse afin de suivre une trajectoire précise lors du déplacement d'un objet à un endroit désigné par les coordonnées X, Y, et Z.

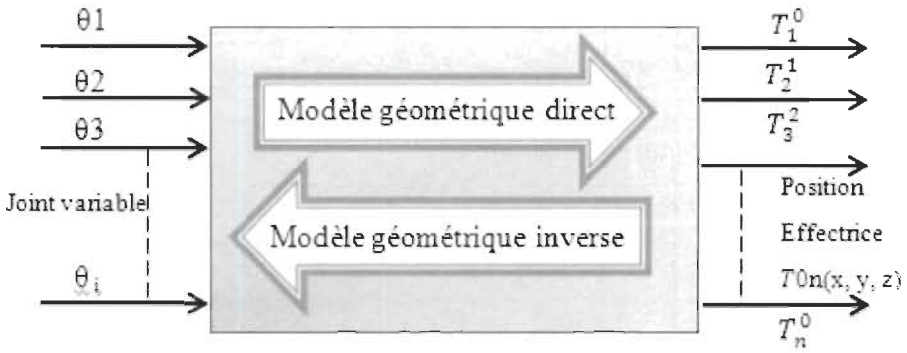

Figure 3-14 La relation entre le modèle géométrique direct et le modèle géométrique inverse avec paramètre Denavit-Hartenberg (DH).

3.5.2.2 Modélisation du notre bras manipulateur

Notre bras manipulateur réalise des mouvements avec 3 degrés de liberté et il est constitué de deux sous-ensembles distincts ;

- Un organe terminal : c'est un servomoteur qui contrôle l'ouverture et la fermeture de la pince.
- Structure mécanique articulée contient trois servomoteurs qui contrôlent les articulations de types rotoïdes. Le rôle de cette structure est d'amener l'organe terminal dans une situation définie par une position et orientation donnée.

Tableau 3-2 Caractéristiques techniques du notre bras manipulateur.

Caractéristiques techniques du bras manipulateur			
Nombre d'articulations		3	
Nombre d'actionneurs		4 servomoteurs	
Rotations		Angles	Types
La base		360°	Rotoïde
Le bras	L'épaule	180°	Rotoïde
	Le coude	180°	Rotoïde
Ouverture de la pince (en degré)		120°	

3.5.2.3 Identification des paramètres de DH de notre bras manipulateur

Nous affectons des repères fictifs à chaque articulation représentée dans la figure ci-dessous :

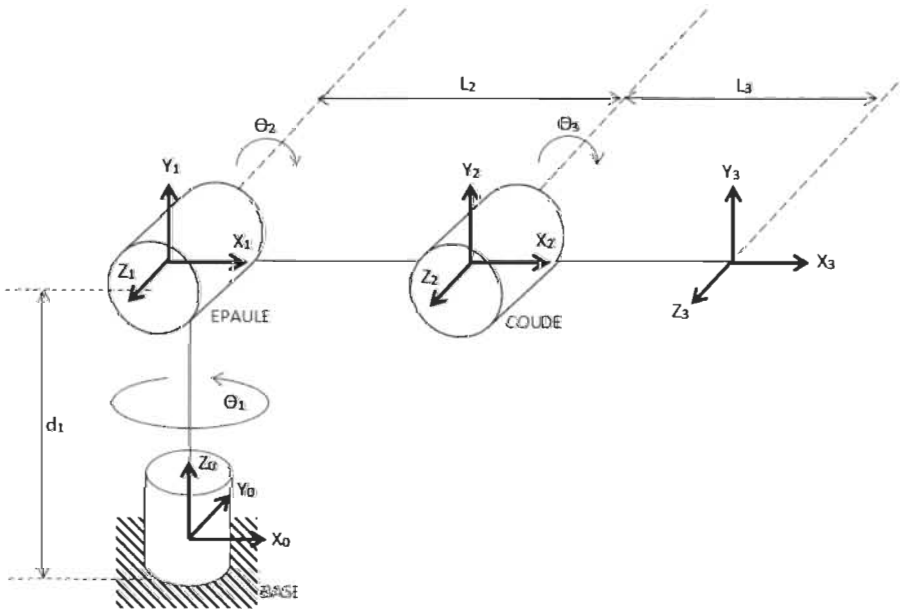

Figure 3-15 Placement des repères selon le modèle DH.

Les paramètres, selon DH, sont présentés au Tableau 3-3

Tableau 3-3 Tableau des paramètres de DENAVIT- HATENBERG de notre réalisation.

Segment	**θi**	**di**	**ai**	**αi**
1	θ1	d_1	0	90°
2	θ2	0	L_2	0
3	θ3	0	L_3	0

$$\mathbf{T^1_0} = \begin{bmatrix} C\theta_1 & -C\alpha_1.S\theta_1 & S\alpha_1.S\theta_1 & a1.C\theta_1 \\ S\theta_1 & C\alpha_1.C\theta_1 & -S\alpha_1.C\theta_1 & a1.S\theta_1 \\ 0 & S\alpha_1 & C\alpha_1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

$$\mathbf{T^2_1} = \begin{bmatrix} C\theta_2 & -C\alpha_2.S\theta_2 & S\alpha_2.S\theta_2 & a2.C\theta_2 \\ S\theta_2 & C\alpha_2.C\theta_2 & -S\alpha_2.C\theta_2 & a2.S\theta_2 \\ 0 & S\alpha_2 & C\alpha_2 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

$$\mathbf{T^3_2} = \begin{bmatrix} C\theta_3 & -C\alpha_3.S\theta_3 & S\alpha_3.S\theta_3 & a3.C\theta_3 \\ S\theta_3 & C\alpha_3.C\theta_3 & -S\alpha_3.C\theta_3 & a3.S\theta_3 \\ 0 & S\alpha_3 & C\alpha_3 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

La matrice de transformation homogène du repère R_0 jusqu'au repère R_4 s'obtient par la multiplication successive des matrices du passage précèdent :

$$\mathbf{T^3_0 = T^1_0\, T^2_1\, T^3_2 =} \begin{bmatrix} C_{123} & -C_1.S_{23} & S_1 & C_1.(a_3C_{23} + a_2C_2) \\ S_1C_{23} & -S_1.S_{23} & -C_1 & S_1.(a_3C_{23} + a_2C_2) \\ S_{23} & C_{23} & 0 & a_3S_{23} + a_2S_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

3.5.3 Le modèle géométrique inverse (MGI)

Nous avons constaté que le MGD d'un robot permettait de calculer les coordonnées opérationnelles donnant la situation de l'organe terminal en fonction des coordonnées articulaires.

Le modèle géométrique inverse consiste à calculer les coordonnées articulaires correspondant à une situation donnée de l'organe terminal [35].

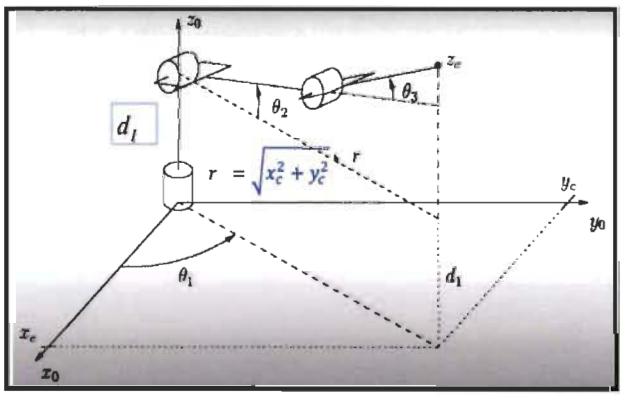

Figure 3-16 Placement des angles de rotation des joints du bras de Robot selon le modèle Denavit-Hartenberg.

La figure suivante présente comment calculer θ_1 par la projection sur les deux axes (x, y) :

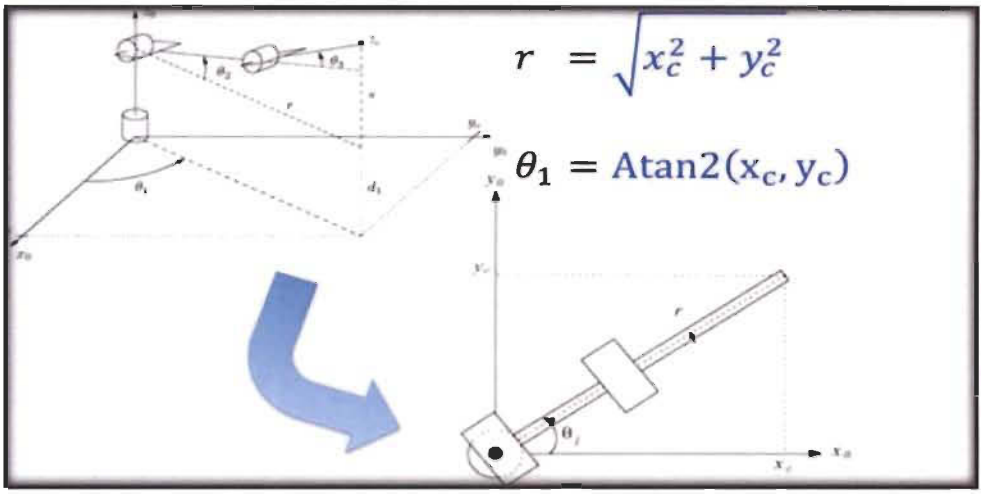

Figure 3-17 Placement de l'angle de rotation θ_1 selon le modèle Denavit-Hartenberg

Calculer θ_1

$$r = \sqrt{x^2 + y^2} \quad (3.7)$$

$$\Rightarrow \quad \theta_1 = \tan^{-1}\left(\frac{x}{y}\right) \quad (3.8)$$

$$\theta 1 = \text{Atan2}(x, y) \quad (3.9)$$

La figure suivante présente les formules utilisées pour faciliter l'obtention des équations des angles θ_2, θ_3:

Loi des sinus:

$$\frac{Sin\ A}{La} = \frac{Sin\ B}{Lb} = \frac{Sin\ C}{Lc} \quad (3.10)$$

Loi de cosinus :

$$Lc^2 = Lb^2 + La^2 - 2Lb \times La \cos(C) \quad (3.11)$$

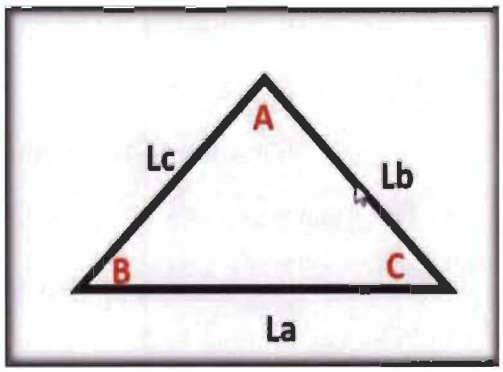

Figure 3-18 la loi des cosinus et sinus.

La figure suivante présente comment calculer θ_2, θ_3 par la projection sur les deux axes (x,z) on utilisant les formules précédentes

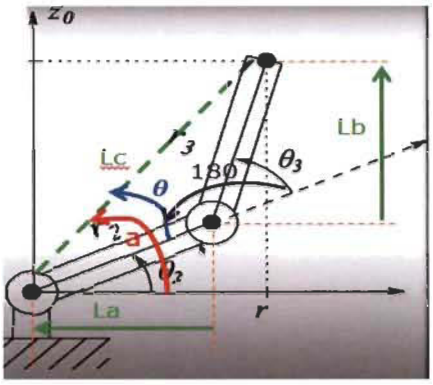

Figure 3-19 Calcul des θ_2, θ_3.

Loi des sinus et cosinus:

On cherche $\theta 3$ avec la loi de cosinus :

$$Lc^2 = Lb^2 + La^2 - 2Lb \times La \cos(C) \quad (3.12)$$

$$X^2 + Z^2 = Lb^2 + La^2 - 2Lb \times La \cos(180 - \theta 3) \quad (3.13)$$

$$\theta 3 = \arccos((x^2 + z^2 - La^2 - Lb^2)/2\ La\ Lb \quad (3.14)$$

On cherche θ avec la loi de sinus :

$$\theta_2 = a - \theta \quad (3.15)$$

$$a = \arctan\left(\frac{z}{x}\right) \quad (3.16)$$

$$\frac{Sin\ A}{La} = \frac{Sin\ B}{Lb} = \frac{Sin\ C}{Lc} \quad (3.17)$$

$$\frac{Sin\ \theta}{Lb} = \frac{Sin\ (180 - \theta 3)}{\sqrt{x^2+z^2}} \quad (3.18)$$

$$\frac{Sin\ \theta}{Lb} = \frac{Sin\ (\theta 3)}{\sqrt{x^2+z^2}} \quad (3.19)$$

$$\theta = \arcsin\left(Lb \times \frac{Sin\ (\theta 3)}{\sqrt{x^2+z^2}}\right) \quad (3.20)$$

$$\theta_2 = a - \theta \quad (3.21)$$

$$\theta 2 = \arctan\left(\frac{z}{x}\right) - \arcsin\left(Lb \times \frac{Sin\ (\theta_3)}{\sqrt{x^2+z^2}}\right) \quad (3.22)$$

Donc les coordonnées articulaires correspondant à une situation donnée de l'organe terminal de notre bras manipulateur sont :

$$\theta_1 = \tan^{-1}\left(\frac{x}{y}\right) \quad (3.23)$$

$$\theta_2 = \tan^{-1}\left(\frac{z}{x}\right) - \sin^{-1}\left(\frac{a_3 \times \sin\theta_3}{\sqrt{x^2+z^2}}\right) \quad (3.24)$$

$$\theta_3 = \cos^{-1}\left(\frac{x^2+z^2{a_2}^2{a_3}^2}{2\,a_2\,a_3}\right) \qquad (3.25)$$

3.5.4 Modélisation cinématique directe (MCD)

Le MCD d'un robot manipulateur décrit les vitesses des coordonnées opérationnelles en fonction des vitesses articulaires. Il est noté :

$$\dot{X}=J(q)\dot{q} \quad [35] \quad (3.25)$$

Avec, J(q) : Matrice Jacobienne.

Nous pouvons obtenir la matrice Jacobienne par une méthode de calcul direct, fondée sur la relation entre les vecteurs des vitesses de translation et de rotation V_n et ω_n du repère R_n, et les vitesses articulaires $\dot{q}$:

$$\begin{bmatrix} Vn \\ \omega n \end{bmatrix} = Jn\dot{q} \quad (3.26)$$

3.5.5 Modèle cinématique directe du robot étudié

Nous commençons par le calcul des vitesses angulaires :

$$\omega_0^0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad (3.27)$$

$$\omega_0^1 = \omega_0^0 + \theta_1^0 Z_0^0 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \qquad (3.28)$$

$$\omega_1^1 = \omega_0^0 + R_1^0 \omega_0^1 = \begin{bmatrix} c_1 & s_1 & 0 \\ 0 & 0 & 1 \\ s_1 & -c_1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ \dot{\theta}_1 \\ 0 \end{bmatrix} \qquad (3.29)$$

$$\omega_1^2 = \omega_1^1 + \theta_2^0 \, Z_1^1 = \begin{bmatrix} 0 \\ \dot{\theta}_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{3.30}$$

$$\omega_2^2 = R_2^1 \, \omega_1^2 = \begin{bmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 s_2 \\ \dot{\theta}_1 c_2 \\ \dot{\theta}_2 \end{bmatrix} \tag{3.31}$$

$$\omega_2^3 = \omega_2^2 + \theta_3^0 \, Z_2^2 = \begin{bmatrix} \dot{\theta}_1 s_2 \\ \dot{\theta}_1 c_2 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 s_2 \\ \dot{\theta}_1 c_2 \\ \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} \tag{3.32}$$

$$\omega_3^3 = R_3^2 \, \omega_2^3 = \begin{bmatrix} c_3 & s_3 & 0 \\ -s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 s_2 \\ \dot{\theta}_1 c_2 \\ \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 (c_3 s_2 + c_2 s_3) \\ \dot{\theta}_1 (c_2 c_3 - s_2 s_3) \\ \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 s_2 \\ \dot{\theta}_1 c_2 \\ \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} \tag{3.33}$$

$$\omega_0^3 = R_0^3 \, \omega_3^3 = \begin{bmatrix} c_1 c_{23} & -c_1 s_{23} & s_1 \\ s_1 c_{23} & -s_1 s_{23} & -c_1 \\ s_{23} & c_{23} & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 s_{23} \\ \dot{\theta}_1 c_{23} \\ \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} 0 & s_1 & s_1 \\ 0 & -c_2 & -c_2 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{3.34}$$

Ensuite, on poursuit par les vitesses linéaires :

$$V_0^0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.35}$$

$$\omega_0^1 = \omega_0^0 + \theta_1^0 \, Z_0^0 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \tag{3.36}$$

$$V_0^1 = V_0^0 + \omega_0^1 \, P_0^1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ d_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.37}$$

$$V_1^1 = V_0^1 \, R_1^0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.38}$$

$$V_1^2 = V_1^1 + \omega_1^2 P_1^2 = \begin{bmatrix} 0 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \times \begin{bmatrix} L_2 c_2 \\ L_2 s_2 \\ 0 \end{bmatrix} = \begin{bmatrix} -\dot{\theta}_2 L_2 s_2 \\ \dot{\theta}_1 L_2 c_2 \\ -\dot{\theta}_1 L_2 c_2 \end{bmatrix} \tag{3.39}$$

$$V_2^2 = R_2^1 V_1^2 = \begin{bmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\dot{\theta}_2 L_2 s_2 \\ \dot{\theta}_2 L_2 c_2 \\ -\dot{\theta}_1 L_2 c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ L_2 \dot{\theta}_2 \\ -\dot{\theta}_2 L_2 c_2 \end{bmatrix} \tag{3.40}$$

$$V_2^3 = V_2^2 + \omega_2^3 P_2^3 = \begin{bmatrix} 0 \\ L_2 \dot{\theta}_2 \\ -\dot{\theta}_2 L_2 c_2 \end{bmatrix} + \begin{bmatrix} \dot{\theta}_1 s_2 \\ \dot{\theta}_1 c_2 \\ \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} \times \begin{bmatrix} L_3 c_3 \\ L_3 s_3 \\ 0 \end{bmatrix} \tag{3.41}$$

$$V_3^3 = R_3^2 V_2^3 = \begin{bmatrix} c_3 & s_2 & 0 \\ -s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -(\dot{\theta}_2 + \dot{\theta}_3) L_3 s_3 \\ L_2 \dot{\theta}_2 + L_3 c_3 (\dot{\theta}_2 + \dot{\theta}_3) \\ -\dot{\theta}_1 L_2 c_{23} \end{bmatrix} = \begin{bmatrix} \dot{\theta}_2 L_2 s_3 \\ \dot{\theta}_2 L_2 c_3 + L_3 (\dot{\theta}_2 + \dot{\theta}_3) \\ -\dot{\theta}_1 L_3 c_{23} \end{bmatrix} \tag{3.42}$$

$$V_0^3 = R_0^3 V_3^3 = \begin{bmatrix} c_1 c_{23} & -c_1 s_{23} & s_1 \\ s_3 c_{23} & -s_1 s_{23} & -c_1 \\ s_{23} & c_{23} & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_2 L_2 s_3 \\ \dot{\theta}_2 L_2 c_3 + L_3 (\dot{\theta}_2 + \dot{\theta}_3) \\ -\dot{\theta}_1 L_3 c_{23} \end{bmatrix} \tag{3.43}$$

$$= \begin{bmatrix} -L_3 c_{23} s_1 & L_2 c_1 (s_3 c_{23} - c_3 s_{23}) - L_3 c_1 s_{23} & -L_3 c_1 s_{23} \\ L_3 c_{23} c_1 & L_2 s_1 (s_3 c_{23} - c_3 s_{23}) - L_3 s_1 s_{23} & -L_3 s_1 s_{23} \\ 0 & L_2 (s_3 s_{23} - c_3 c_{23}) - L_3 c_{23} & L_3 c_{23} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{3.44}$$

Le modèle cinématique dans R_3 est donnée par :

$$\begin{bmatrix} V_3^3 \\ \omega_3^3 \end{bmatrix} = \begin{bmatrix} 0 & L_2 s_3 & 0 \\ 0 & L_3 + L_2 c_3 & L_3 \\ -L_3 c_{23} & 0 & 0 \\ s_{23} & 0 & 0 \\ c_{23} & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{3.45}$$

Le modèle cinématique finale dans R_0 est donnée par :

$$\begin{bmatrix} V_3^3 \\ \omega_3^3 \end{bmatrix} = \begin{bmatrix} -L_3c_{23}s_1 & L_2c_1(s_3c_{23} - c_3s_{23}) - L_3c_1s_{23} & -L_3c_1s_{23} \\ 0 & L_2s_1(s_3c_{23} - c_3s_{23}) - L_3s_1s_{23} & -L_3s_1s_{23} \\ 0 & L_2(s_3s_{23} - c_3c_{23}) - L_3c_{23} & L_3c_{23} \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \quad (3.46)$$

3.5.6 Modèle cinématique inverse (MCI)

L'objectif du MCI est de calculer, à partir d'une configuration q donnée, les vitesses articulaires $\dot{q}$ qui assurent au repère terminal une vitesse opérationnelle $\dot{X}$ imposée.

Pour obtenir le modèle cinématique inverse, nous inversons le modèle de cinématique direct en résolvant un système d'équations linéaires. La mise en œuvre peut être faite de façon analytique ou numérique. Les solutions analytiques réduisent le nombre d'opérations de façon remarquable par rapport aux solutions numériques. Mais, il faut traiter les cas singuliers distinctement. Les solutions numériques sont plus générales et traitent tous les cas de la même manière [35].

3.5.6.1 Modèle cinématique inverse du robot étudié

Nous pouvons déterminer la solution au système de façon itérative en utilisant un Jacobien, en se servant de la matrice $J_{0(3x3)}$ réduite, que nous obtenions du MCD en supprimant les lignes 3, 4 et 6 ;

$$J_{0(3x3)} = \begin{bmatrix} -L_3c_{23}s_1 & L_2c_1(s_3c_{23} - c_3s_{23}) - L_3c_1s_{23} & -L_3c_1s_{23} \\ L_3c_{23}c_1 & L_2s_1(s_3c_{23} - c_3s_{23}) - L_3s_1s_{23} & -L_3s_1s_{23} \\ 0 & -c_1 & -c_1 \end{bmatrix} \quad (3.47)$$

Son MCI est obtenu en calculant $J_0^{-1}{}_{(3x3)}$ réduite avec le nouveau vecteur $\dot{X}r = [vx\ vy\ wy]^T$:

$$\dot{\theta} = J_0^{-1}{}_{(3x3)}\dot{X}r \qquad (3.48)$$

$$\begin{bmatrix}\dot{\theta_1}\\ \dot{\theta_2}\\ \dot{\theta_3}\end{bmatrix} = \begin{bmatrix}-L_3c_{23}s_1 & L_2c_1(s_3c_{23} - c_3s_{23}) - L_3c_1s_{23} & -L_3c_1s_{23}\\ L_3c_{23}c_1 & L_2s_1(s_3c_{23} - c_3s_{23}) - L_3s_1s_{23} & -L_3s_1s_{23}\\ 0 & -c_1 & -c_1\end{bmatrix}^{-1}\dot{X}r \qquad (3.49)$$

Le déterminant vaut : $L_2L_3c_{23}c_1(s_3c_{23} - c_3s_{23})$

Ce qui nous donne :

$$\begin{bmatrix}\dot{\theta_1}\\ \dot{\theta_2}\\ \dot{\theta_3}\end{bmatrix} = \frac{1}{L_2L_3c_{23}c_1(s_3c_{23} - c_3s_{23})} \times \qquad (3.50)$$

$$\begin{bmatrix}-L_2s_1c_1(s_3c_{23} - c_3s_{23}) & L_2{c_1}^2(s_3c_{23} - c_3s_{23}) & 0\\ L_3c_{23}{c_1}^2 & L_3s_1c_1c_{23} & -{L_3}^2c_{23}s_{23}\\ -L_3c_{23}{c_1}^2 & L_3c_{23}(-L_3s_{23} + L_2(s_3c_{23} - c_3s_{23})) & L_3c_{23}(L_3s_{23} - L_2(s_3c_{23} - c_3s_{23}))\end{bmatrix}\dot{X}r$$

3.6 Conclusion

Dans ce chapitre, nous avons commencé par décrire les différents organes constituant le bras manipulateur étudié. Par la suite, nous avons donné une représentation aussi simpliste que possible de son modèle géométrique et son modèle cinématique.

Nous débuterons dans ce qui suit, la partie concernant la commande du robot. Nous généralement recherchons toujours le modèle le plus simple qui permet d'expliquer, de manière satisfaisante, le comportement du processus dans son domaine d'application et celui offrant plus de facilité d'utilisation pour ses usagers.

Chapitre 4 - Commande du Bras Manipulateur

4.1 Introduction

Ce chapitre a pour objet l'application des différentes commandes citées précédemment sur le robot manipulateur. Nous introduisons la modélisation d'un servomoteur du bras sous Python dans un asservissement dans un premier temps avec la commande PID, et dans un deuxième temps, avec la commande par Logique Floue avec la présentation des résultats de simulations obtenus pour chaque type de commande.

4.2 Matériel utilisé

4.2.1 Raspberry Pi 4 B

Le Raspberry Pi 4 B est un nano-ordinateur pouvant se connecter à un moniteur, à un ensemble clavier/souris et disposant d'interfaces Wi-Fi, Bluetooth et Ethernet.

Il fonctionne depuis une carte micro-SD et utilise un système d'exploitation basé sur Linux ou Windows 10 IoT. Il est fourni sans boîtier, alimentation, clavier, écran et souris dans le but de diminuer le coût et de favoriser l'utilisation du matériel de récupération.

Cette nouvelle version de la carte Raspberry Pi 4 B est basée sur [37] un Processeur ARM Cortex-A72 64 bits quatre cœurs à 1,5 GHz, 1 GB de mémoire RAM (il existe également en version 2 et 4 GB de RAM), interfaces Wi-Fi, Bluetooth, et ports USB 2.0, USB 3.0, Ethernet Gigabit, HDMI, micro-SD, et multiple GPIO.

Les interfaces Ethernet et Bluetooth ont été améliorées par rapport aux versions précédentes et supportent maintenant l'Ethernet Gigabit ainsi que le Bluetooth 5.0. Cette carte

est basée sur un processeur ARM et permet l'exécution du système d'exploitation GNU/Linux/Windows 10 IoT et des logiciels compatibles.

Le Raspberry Pi peut effectuer des tâches d'un PC de bureau (feuilles de calcul, traitement de texte, jeux). Il peut également diffuser des vidéos en haute définition grâce à son circuit BroadcomVideoCore VI (permet le décodage des flux vidéo 4k H.265 et 1080p H.264). Cette nouvelle version supporte également le 4k en natif via ses sorties micro-HDMI.

La Raspberry Pi 4 B nécessite une carte SD munie d'un OS, une alimentation, un clavier USB, une souris USB, un boîtier et des câbles (non inclus). Pour préparer une carte SD bootable, il faut disposer d'un PC avec lecteur de carte SD.

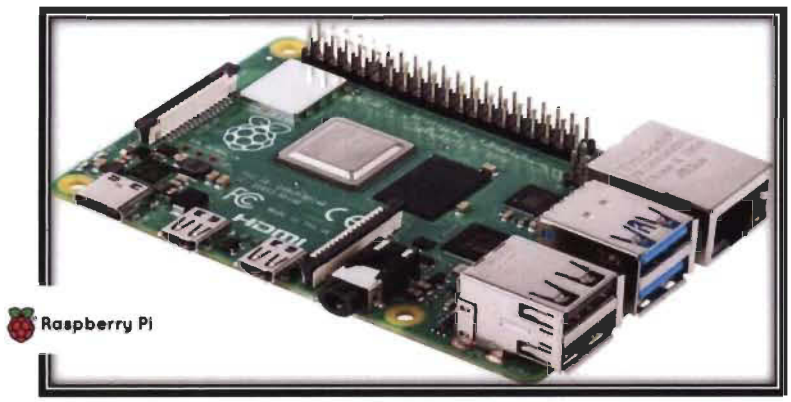

Figure 4-1 Carte Raspberry Pi 4 B.

4.2.1.1 Caractéristiques de la Carte Raspberry Pi 4 B

Caractéristiques Raspberry Pi 4 B :

- Alimentation : 5 Vcc/maxi 3 A* via prise USB Type C, Intensité maximale si toutes les fonctions sont utilisées.
- CPU : ARM Cortex-A72 quatre cœurs 1,5 GHz

- Wi-Fi : Dual-band 2,4 et 5 GHz, 802.11b/g/n/ac
- Bluetooth: Bluetooth 5 compatible BLE (Broadcom BCM43438)
- Mémoire : 1 GB LPDDR4
- Circuit vidéo : VideoCore VI à 500 MHz
- Les ports :2 ports USB 2.0 et 2 ports USB 3.0 et Port Ethernet Gigabit RJ45
- Bus: SPI, I2C, série
- Sorties : 2 x micro-HDMI (4K @ 60 fps maxi), Jack 3,5 mm (partagé avec audio)
- Dimensions : 88 x 58 x 17 mm
- Version : Raspberry Pi 4 B - 1 GB
- Poids : 46 g
- Les Interfaces : CSI pour caméra, DSI pour écran
- Sorties audios : 2 x micro-HDMI avec gestion du 5.1 et Jack 3,5 mm en stéréo (partagé avec vidéo)

4.2.2 LSS - Carte adaptateur Lynxmotion Smart Servo

4.2.2.1 Description

La carte adaptateur Lynxmotion Smart Servo (LSS) est une carte électronique qui permet une connexion et un contrôle faciles des servomoteurs « intelligents » Lynxmotion. La carte comprend une variété de caractéristiques et de fonctions en tant que carte de distribution d'alimentation centrale via six connecteurs LSS. L'adaptateur LSS est idéalement conçu pour être alimenté par une batterie LiPo via le connecteur XT60 mâle embarqué, ou avec un

adaptateur mural équipé d'un connecteur XT60 femelle. Il existe de nombreuses façons de s'interfacer avec cette carte, comme décrit ci-dessous.[21]

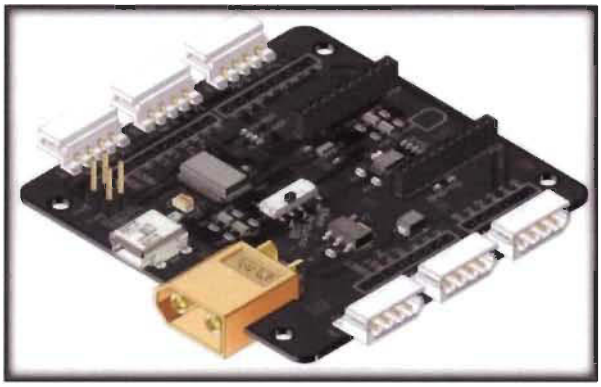

Figure 4-2 LSS - Carte adaptateur Lynxmotion Smart Servo.

4.2.2.2 Fonctionnalités

- Contrôle des actionneurs LSS sur le même bus via 6 connecteurs.
- Prise Bee intégrée (XBee, modules Bluetooth Bee ou WiFi Bee).
- Peut fonctionner comme une carte d'exploration USB XBee.
- Compatible avec le blindage Arduino en utilisant des en-têtes d'empilage.
- Méthode de contrôle sélectionnable : USB, Arduino ou XBee.
- Compatibilité des trous de montage du Raspberry Pi B + / A + / 3.
- Connecteur XT60 pour l'entrée d'alimentation.
- Sélection automatique de l'alimentation logique (USB / externe).

4.2.2.3 Caractéristiques

- Courant continu par connecteur : 3A (max).
- USB vers série intégré (puce de port COM virtuel FTDI).
- Régulateur 5V intégré.
- Régulateur 3.3V intégré.

- Mini connecteur USB.
- Dimensions : 64 x 64 x 15 mm.
- Diamètre du trou de montage : 3,1 mm.

4.3 Logiciels utilisés : L'environnement de programmation en Python

Python est un environnement de développement IDLE « Integrated DeveLopmentEnvironment » est un environnement de développement intégré spécialement pour Python qui nous permet d'exécuter directement du code Python. Il s'agit d'un langage Open Source et gratuit, téléchargeable sur le site officiel dans la rubrique téléchargement. Python Launcher permet lui d'exécuter du code Python créé dans des fichiers séparés .py en double cliquant simplement sur ces fichiers [38].

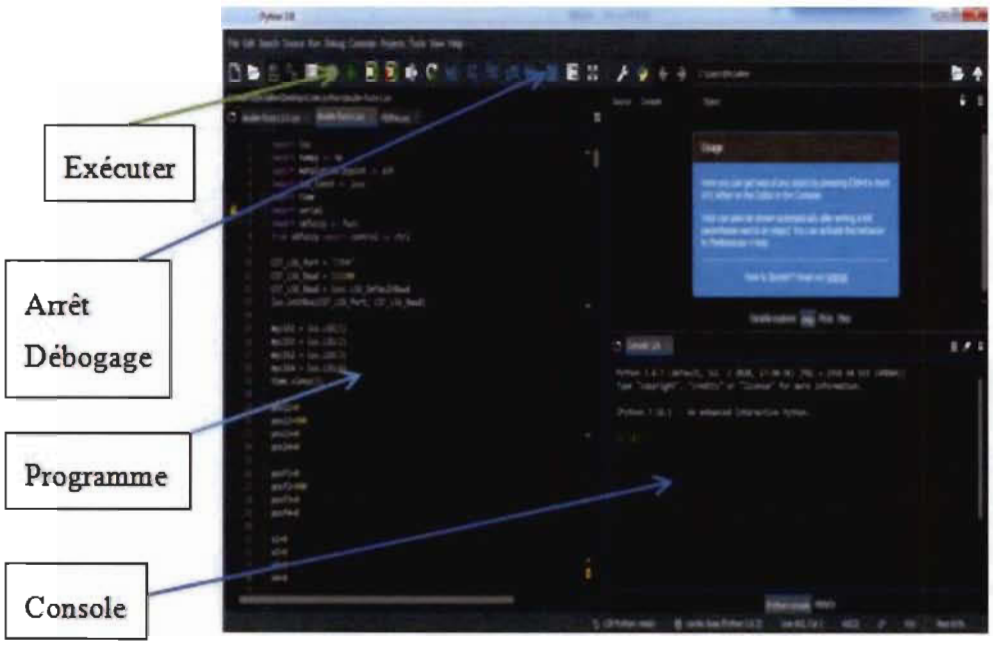

Figure 4-3 Environnement de développement en Python.

4.4 Commande du bras manipulateur

L'objectif est d'étudier et proposer la commande du bras robotisé pour améliorer la finesse et la précision de son mouvement.

Nous cherchons à faire atteindre une certaine valeur à une variable du système. Dans le cas d'un servomoteur, il s'agit principalement d'une vitesse ou d'une position.

– Les effets des perturbations doivent être minimisés, voire effacés, et ce le plus vite possible (régulation).

– Les changements de consigne doivent être suivis rapidement et avec une bonne précision, si possible sans dépassement (poursuite).

Néanmoins, ces critères de performance sont souvent antagoniques, et nous verrons que le réglage de l'asservissement en détermine un compromis.

4.4.1 Régulateur PID

Régulateur PID (Proportionnel, Intégral, Dérivé) est une méthode d'auto régulation (en boucle fermée), qui cherche à réduire l'erreur entre la consigne et la mesure [39].

E = Consigne – Mesure

Le régulateur PID sert à atteindre la valeur souhaitée pour une des variables du système (vitesse, position...).

- Régulation : minimiser rapidement les perturbations.

- Poursuite : s'adapter rapidement aux nouvelles consignes. Ceci s'appelle l'asservissement.

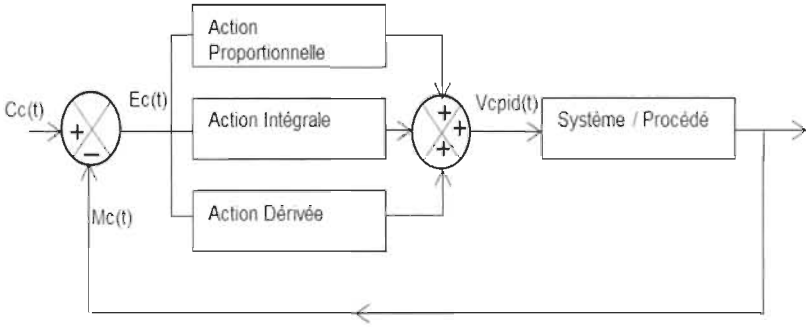

Figure 4-4 Schéma du régulateur PID [39].

4.4.1.1 Asservissements en régulation de la position et influence des coefficients sur le servomoteur du bras robotisé

Tel que mentionné précédemment, le régulateur PID est un régulateur, en boucle fermée (Figure 4-5), qui a pour rôle de diminuer l'erreur entre la consigne et la mesure et d'atteindre la position souhaitée.

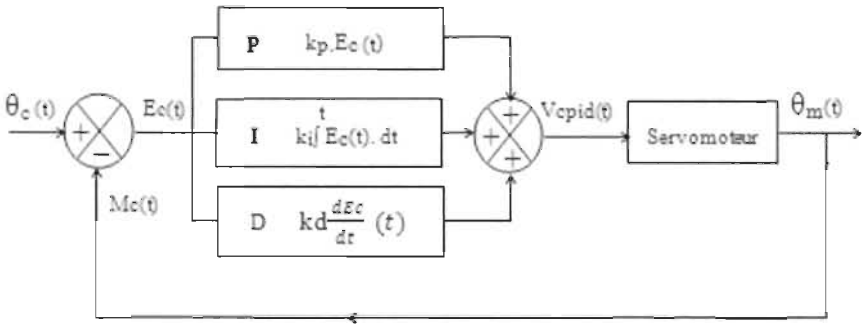

Figure 4-5 Schéma-bloc de l'implantation du PID en régulation de position.

4.4.1.1.1 L'action proportionnelle sur la position

L'asservissement de type proportionnel (P) est le plus simple qui soit. Il s'agit d'appliquer une correction proportionnelle à l'erreur corrigeant de manière instantanée tout écart de la grandeur à régler :

$$P_C(t) = k_p . E_C(t) \quad [39]$$

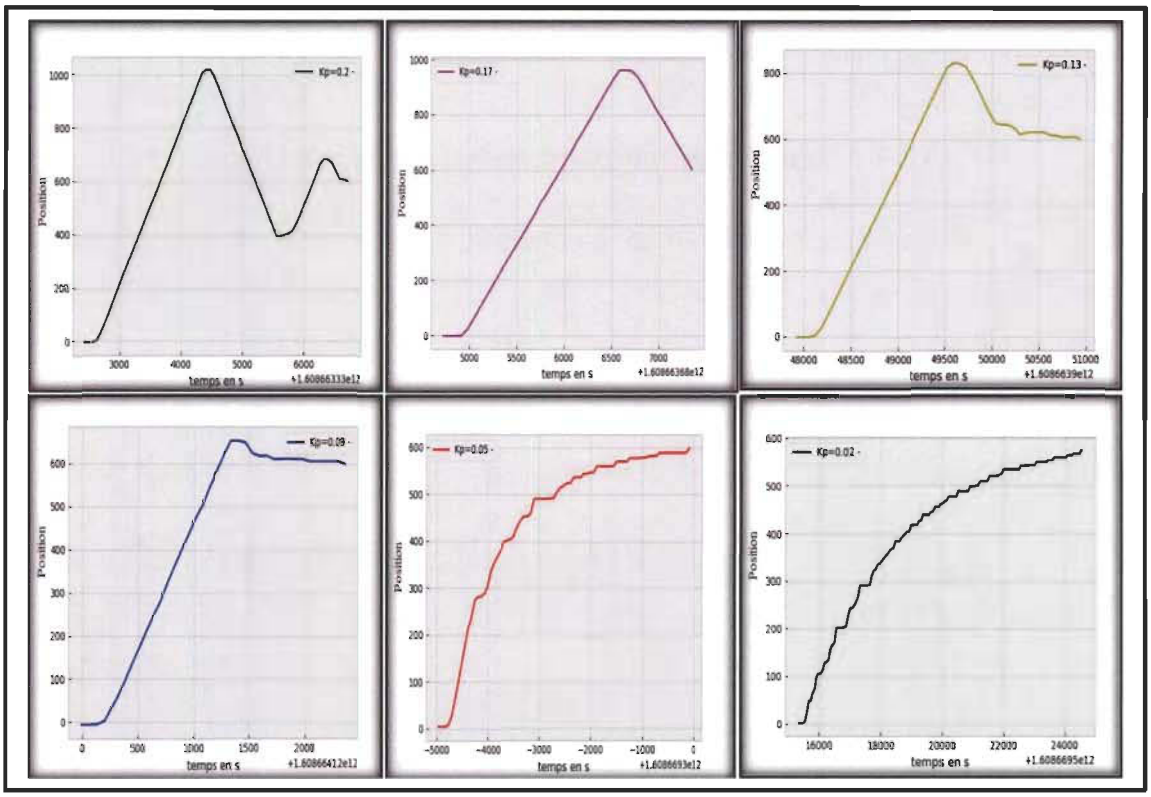

Figure 4-6 Réponse d'un servomoteur du bras dans un asservissement proportionnel, passage de la position zéro à la position 600 = 60°.

Alors, d'après les signaux de la figure 4.6, le rôle du proportionnel est d'amplifier virtuellement l'erreur pour que le système réagisse plus vivement, comme si l'erreur était plus grande qu'elle ne l'est en réalité, il permet de vaincre les grandes inerties du système et diminue le temps de montée en donnant de la puissance au servomoteur (plus l'erreur est

grande, plus nous donnons de puissance au servomoteur). Lorsque nous augmentons Kp, le système réagit plus vite et l'erreur statique s'en trouve améliorée, mais en contrepartie, le système perd en stabilité. Le dépassement se fait de plus en plus grand, et le système peut même diverger dans le cas d'un Kp démesuré. Un ajustement du gain est donc requis pour permettre une réponse acceptable. Toutefois, l'utilisation de méthodes classiques, basées sur le modèle ou sa fonction de transfert, est difficile à appliquer car les caractéristiques du système changent en fonction de la charge et de la position des différentes articulations.

4.4.1.1.2 L'action proportionnelle intégrale PI sur la position

L'erreur est intégrée sur un intervalle de temps, puis multipliée par une constante Ki.

$$Ic(t)= ki\int^{t} Ec(t).dt \quad [39]$$

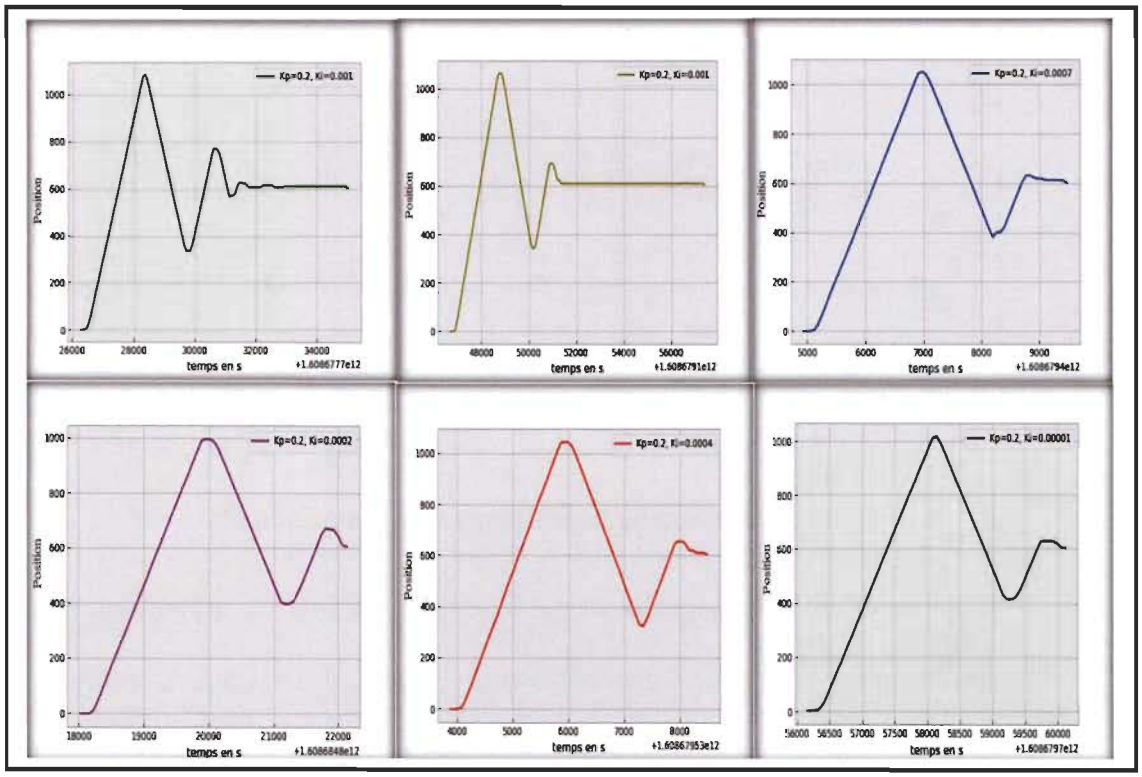

Figure 4-7 Réponse d'un servomoteur du bras dans un asservissement Kp et Ki, passage de la position zéro à la position 600 = 60°.

Le terme intégral complète l'action proportionnelle puisqu'il permet de compenser l'erreur statique et d'augmenter la précision en régime permanent. L'idée est d'intégrer l'erreur depuis le début et d'ajouter cette erreur à la consigne : lorsque nous nous rapprochons de la valeur demandée, l'erreur devient de plus en plus faible. L'intégrale agissant comme un filtre sur le signal intégré, permet de diminuer l'impact des perturbations (bruit, parasites), et il en résulte alors un système plus stable. Malheureusement, le terme intégral sur notre système ici d'après la figure 4.7 est mal adapté pour les conditions du système. Notons que plus le terme intégral est important plus il entraînera de dépassement de la consigne, une stabilisation plus lente, voire même des oscillations divergentes.

4.4.1.1.3 L'action PID sur la position

La dérivée dépend des variations de l'erreur $Ec(t)$:

$$D_c(t) = K_d \frac{dEc(t)}{dt} \quad [39]$$

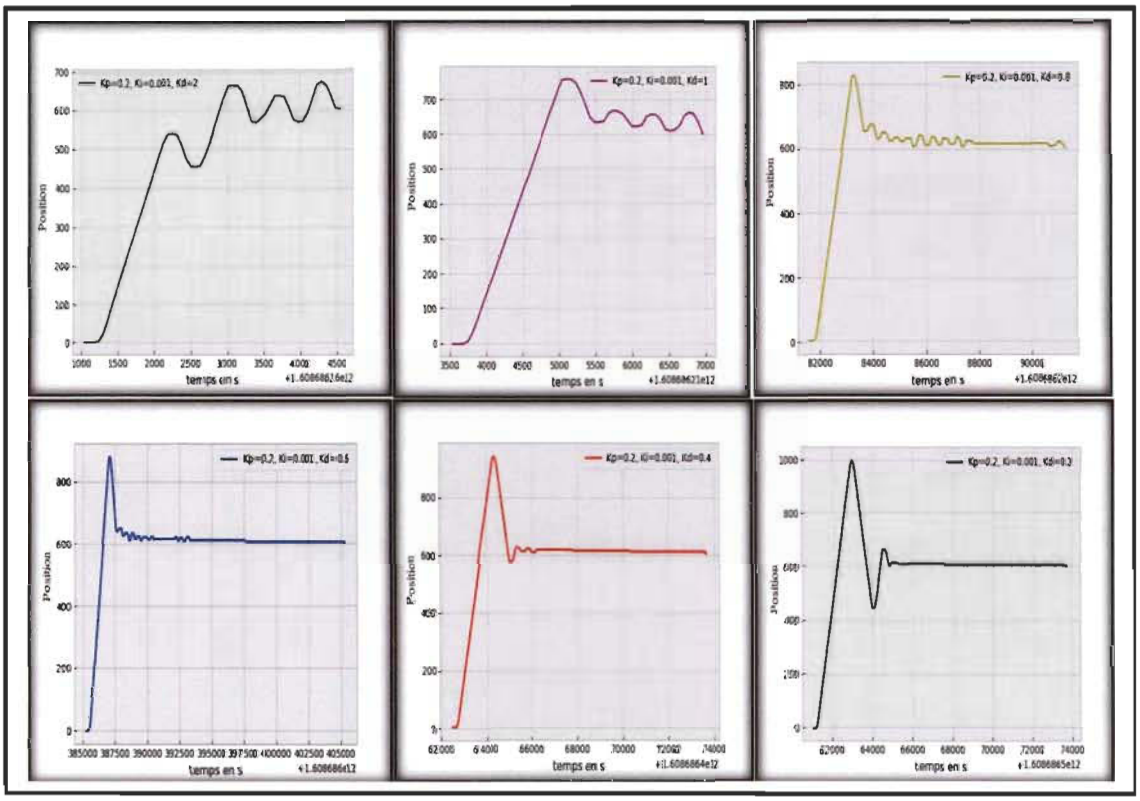

Figure 4-8 Réponse d'un servomoteur du bras dans un asservissement PID, passage de la position zéro à la position 600 = 60°.

Comme nous l'avons vu précédemment, l'introduction que la composante intégrale entraine également un risque de dépassement de la consigne. En calculant la variation de l'erreur à chaque itération, la partie dérivée permet de limiter ce phénomène en ralentissant la commande lorsque la valeur de la sortie s'approche de la consigne, elle freine alors le système.

4.4.1.1.4 Réglage des coefficients PID sur la position

Le réglage d'un PID consiste à trouver les meilleurs coefficients Kp, Ki et Kd dans le but d'obtenir une réponse adéquate du procédé et de la régulation. L'objectif est d'obtenir une réponse robuste, rapide et précise tout en limitant les dépassements.

En absence d'un modèle du système, nous avons utilisé une approche manuelle (empirique) d'ajustement des gains du correcteur. On a augmenté Kp à volonté pour choisir une valeur relativement grande de Kp, puis nous avons ajusté Ki et Kd pour éliminer l'erreur statique, limiter le dépassement et atteindre la consigne le plus vite possible.

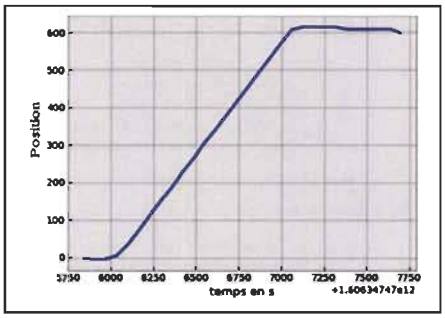

Figure 4-9 Gain ajusté P Kp=0.9 en position 600 = 60.

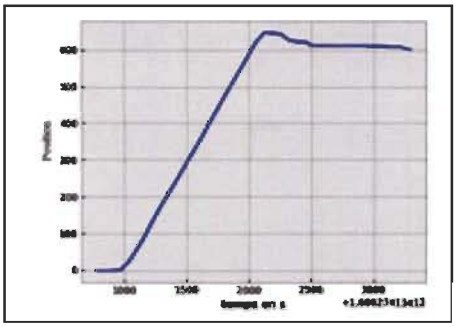

Figure 4-10 Gains ajustés PI Kp=0.9, Ki=10^{-5}en position 600 = 60.

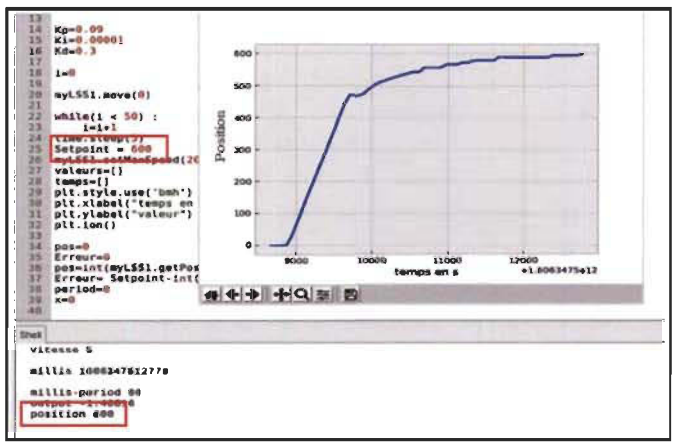

Figure 4-11 Gains ajustés pour PID : Kp=0.9, Ki=10^{-5}, Kd=0.3, en position 600 = 60°.

4.4.1.2 Asservissements en régulation de vitesse et influence des coefficients sur le servomoteur du bras robotisé

De la même manière que pour l'asservissement de position (angle), le régulateur PID peut être employé pour diminuer l'erreur entre la consigne et la mesure et ainsi atteindre une vitesse souhaitée tel que montré dans la Figure 4-12.

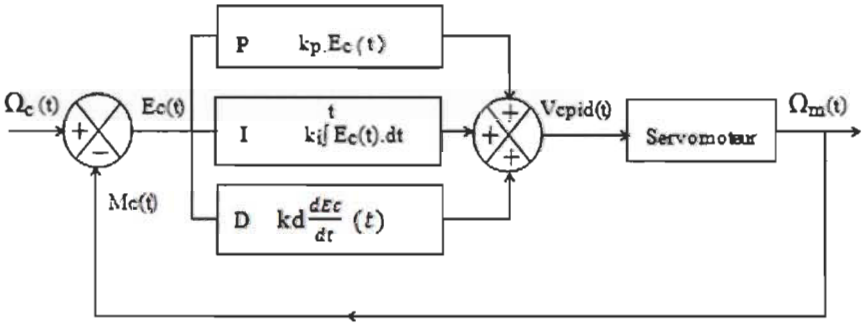

Figure 4-12 Schéma-bloc de l'implantation du PID en régulation de vitesse.

4.4.1.2.1 Réglage des coefficients PID sur la vitesse

Comme nous avons fait sur le réglage d'un PID sur la position, c'est le même principe d'ajustement les coefficients sur la vitesse consiste à trouver les meilleurs coefficients Kp, Ki et Kd dans le but d'obtenir une réponse adéquate du procédé et de la régulation.

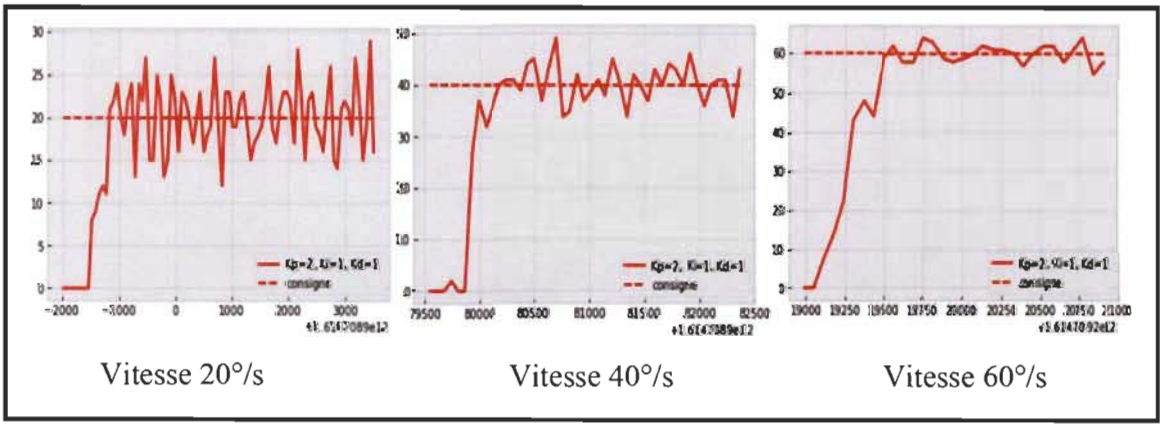

Figure 4-13 Gains ajustés pour PID : Kp=2, Ki=1, Kd=1, trois vitesses différentes.

Nous remarquons d'après la Figure 4.13 que plus nous ralentissons ou diminuons la vitesse, plus le signal de mesure de vitesse est bruité.

Toutefois, visuellement la rotation du servomoteur n'est pas saccadée comme le suggère la courbe.

4.4.2 Double asservissement position-vitesse d'un servomoteur avec PID

L'objectif d'un double asservissement à la fois en position et en vitesse, est de maîtriser la vitesse de déplacement dans un asservissement de position et de permettre de définir une courbe de vitesse au cours d'un déplacement indépendamment de la distance à parcourir. En effet, un simple asservissement PID de position modifie la tension à appliquer selon la distance du déplacement. Or il est souvent préférable de se déplacer à une vitesse globalement constante quelle que soit la distance à parcourir, en accélérant progressivement jusqu'à un palier, puis en décélérant juste avant d'atteindre la position finale afin d'éviter de la dépasser (ce que fait naturellement un conducteur de véhicule). Typiquement, nous implémentons une commande trapézoïdale, c'est-à-dire que nous appliquons un seuil à la fois sur la vitesse et l'accélération [40] :

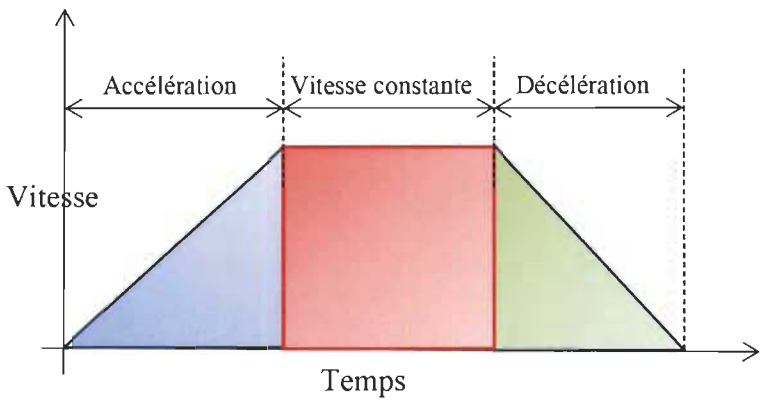

Figure 4-14 Modèle trapézoïdale de la vitesse.

4.4.2.1 Modélisation du double asservissement PID

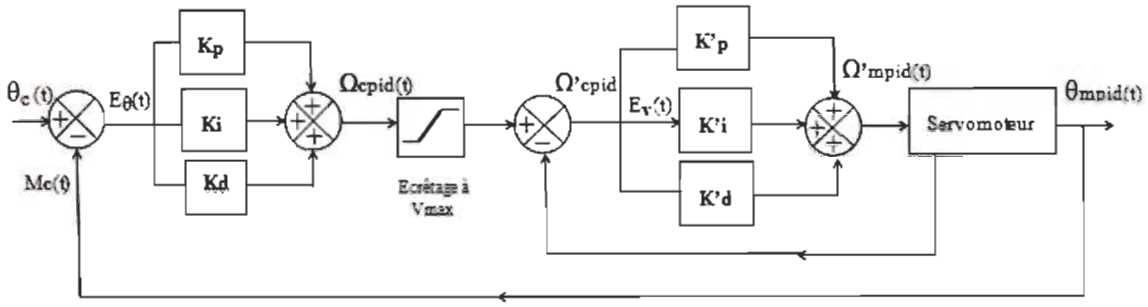

Figure 4-15 Schéma-bloc de la double implémentation de PID tenant compte des écrêtages [40].

Le principe est simple : un asservissement PID de position modifie la tension à appliquer selon la distance du déplacement: l'évolution de la vitesse se fait selon 3 phases, une phase d'accélération, une de vitesse constante et enfin une phase de décélération. Ainsi, plus nous nous rapprochons de la position de référence, plus la vitesse diminue afin d'atteindre une précision d'autant plus grande que la position de référence est proche.

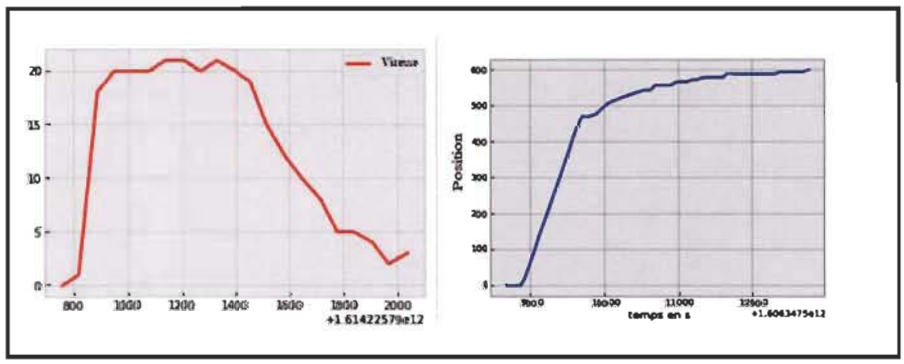

Figure 4-16 Résultat du double asservissement PID.

Le résultat obtenu par un double asservissement ne possède pas beaucoup d'ondulation au niveau de la courbe de vitesse (ondulation faible). Nous remarquons que le signal de sortie

MLI (PWM) au cours de la vitesse constante est maximal, ce qui signifie aussi que la vitesse est maximale et suit de près la valeur limite prédéfinie.

4.4.3 Régulation par la Logique Floue

Le terme d'ensemble flou apparaît pour la première fois en 1965 lorsque le professeur Lotfi A. Zadeh, de l'université de Berkeley aux USA, publie un article intitulé « Ensembles flous » (Fuzzy sets) [41].

Un régulateur flou est un système à base de connaissance particulière composé de quatre modules principaux à savoir : la base de règle, la fuzzification, le moteur d'inférence et la défuzzification comme il est montré par la Figure 4.17.

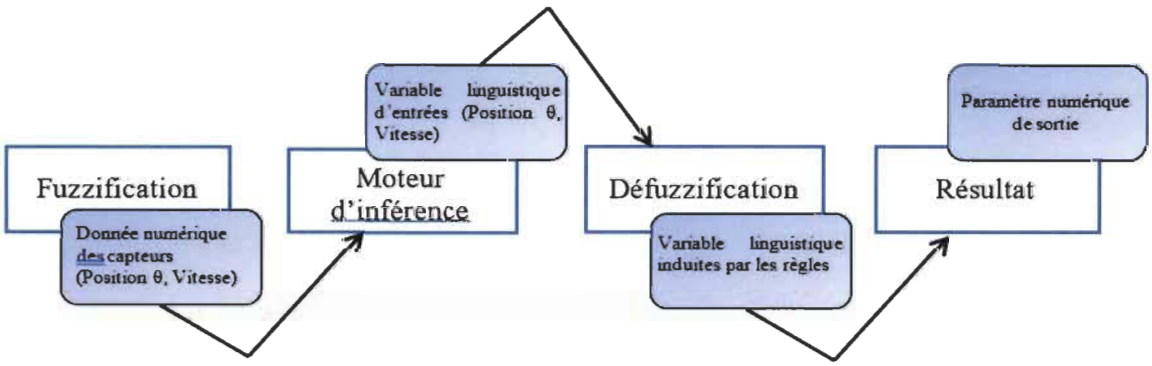

Figure 4-17 Schéma représentatif du fonctionnement d'un système par Logique Floue [41].

4.4.3.1 Régulation par la Logique Floue sur la position

4.4.3.1.1 La fuzzification de position

Conversion des valeurs d'entrée (grandeurs physiques) en grandeurs floues réunies dans le vecteur x, cette partie consiste à traduire les données numériques quantitatives provenant d'un capteur, alors dans notre cas l'entrée est la position du servomoteur, en variables

linguistiques qualitatives grâce à une fonction d'appartenance utilisée. Alors, l'entrée est partitionnée dans notre cas en 8 variables linguistiques sur un intervalle de position [-1800, 1800], représentant une variation d'angle entre -180° et 180°, par les fonctions d'appartenance représentées sur la figure [41].

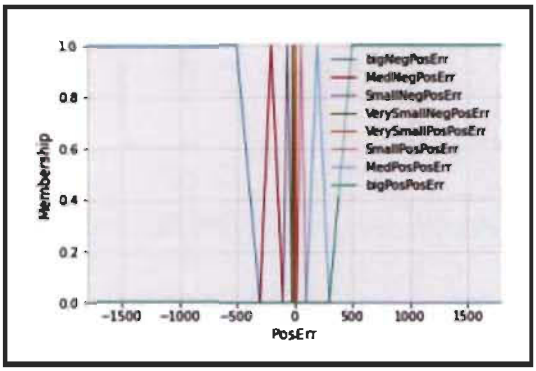

Figure 4-18 Variables linguistiques (θ, $\mu(\theta)$) pour décrire la position.

4.4.3.1.2 Inférence floue de la position

Le contrôleur par Logique Floue utilise une forme de quantification d'informations imprécises (ensembles flous d'entrée) à générer par un schéma d'inférence, qui est basé sur une base de connaissances de la force de contrôle à appliquer sur le système. Avec la base des règles, la prise des décisions pour chaque règle activée donne un sous-ensemble flou de sortie, alors les règles de base pour la position du mouvement dans le servomoteur s'expriment comme suit :

Tableau 4-1 Les règles de base pour la position du mouvement dans le servomoteur

Les règles	Erreur de Position	Résultat PWM
Règle 1	Grande Positive	Grande Positive
Règle 2	Moyenne Positive	Moyenne Positive
Règle 3	Petite Positive	Petite Positive
Règle 4	Très petite Positive	Très petite Positive
Règle 5	Grande Négative	Grande Négative

Règle 6	Moyenne Negative	Moyenne Negative
Règle 7	Petite Negative	Petite Negative
Règle 8	Très petite Negative	Très petite Negative

4.4.3.1.3 La défuzzification

Le résultat de l'inférence en utilisant une des méthodes d'implication floue ne peut être utilisée directement. Une transformation doit être prévue à la sortie du bloc d'inférence pour convertir les sous-ensembles flous de sortie en valeurs déterminées, cette transformation étant connue par le terme défuzzification (concrétisation) [41]. La figure suivante représente l'influence de la logique floue sur la position :

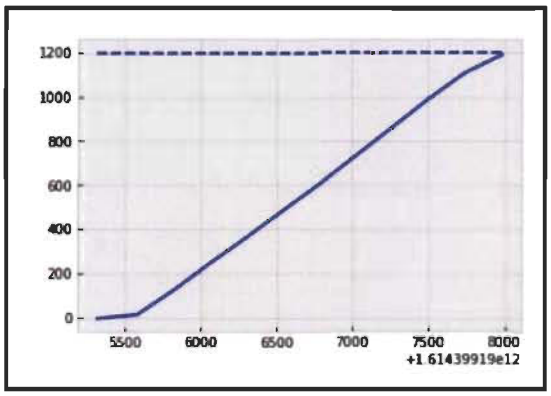

Figure 4-19 Réponse d'un servomoteur du bras dans un asservissement avec logique floue. Passage de la position zéro à la position 1200 = 120°.

4.4.3.2 Régulation de vitesse par la Logique Floue

4.4.3.2.1 La fuzzification de la vitesse

L'entrée maintenant est la vitesse angulaire du servomoteur, nous traduisons ces données numériques quantitatives en 3 variables linguistiques sur un intervalle de vitesse [-60, 60] par les fonctions d'appartenance représentées sur la figure 4-20 [41].

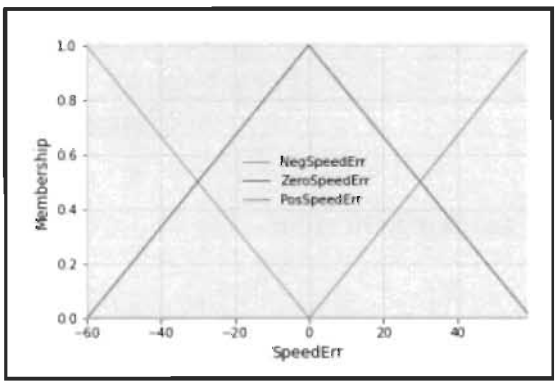

Figure 4-20 Variables linguistiques (Ω, $\mu(\Omega)$) pour décrire la Vitesse.

4.4.3.2.2 Inférence Floue de la Vitesse

Le schéma d'inférence généré par le contrôleur par logique floue de la vitesse angulaire du servomoteur permet la prise des décisions pour chaque règle activée. Celui-ci donne le sous-ensemble flou de sortie suivant.

Tableau 4-2 les règles de base pour la vitesse du mouvement dans le servomoteur

Les règles	Erreur de vitesse	Résultat PWM
Règle 1	Negative	Negative
Règle 2	Zero	Zero
Règle 3	Positive	Positive

4.4.3.2.3 La défuzzification

L'étape finale, appelée défuzzification (ou concrétisation), consiste à utiliser la transformation à la sortie du bloc d'inférence et à réaliser la conversion des sous-ensembles flous de sortie en valeurs déterminées [41]. La figure suivante représente la réponse de la logique floue pour trois exemples de commande de vitesse.

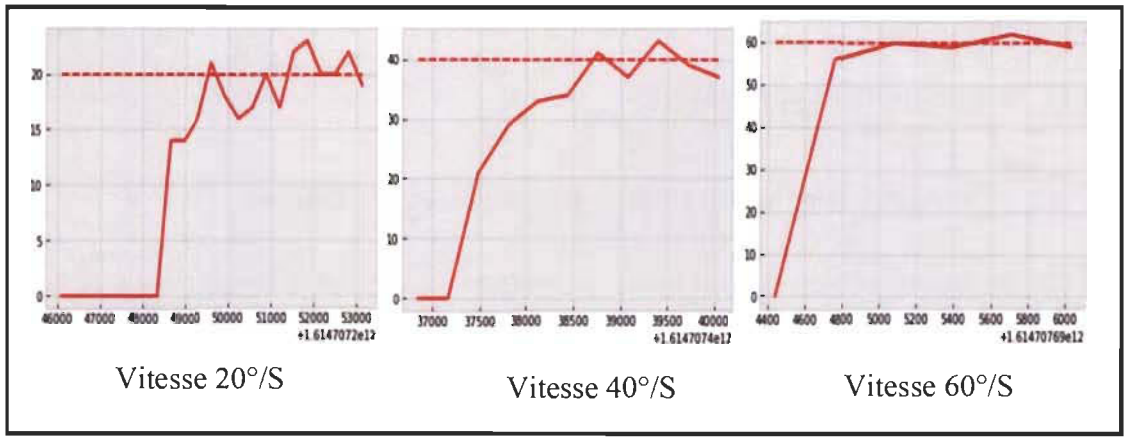

Figure 4-21 Réponse de la commande de vitesse utilisant la Logique Floue.

Nous remarquons d'après le Figure 4.23 que plus nous ralentissons ou diminuons la vitesse, plus le signal de mesure de vitesse est bruité. Ce phénomène a été observé aussi pour la commande de vitesse avec PID.

Dans la section suivante, nous appliquons le double asservissement par Logique Floue d'un servomoteur, passant par la méthode de freinage en douceur avant la position souhaitée pour minimiser l'erreur de la position.

4.4.4 Double asservissement par Logique Floue d'un servomoteur

L'objectif d'un double asservissement du servomoteur, à la fois en position et en vitesse, est de maîtriser la vitesse de déplacement dans un asservissement de position pour permettre de définir une courbe de vitesse au cours d'un déplacement indépendamment de la distance à parcourir. En effet, l'asservissement par Logique Floue sur les deux variables (position et vitesse), facilite le freinage à appliquer (sur la tension) selon la distance du déplacement pour ralentir avant d'arriver à la position souhaitée. Or il est souvent préférable de se déplacer à une vitesse globalement constante quelle que soit la distance à parcourir, en décélérant juste

avant d'atteindre la position finale afin d'éviter de la dépasser. Typiquement, on implémente une commande trapézoïdale [40].

4.4.4.1.1 La fuzzification de la position et de la vitesse

Les entrées sont maintenant la position et la vitesse angulaire du servomoteur. Nous traduisons ces données numériques quantitatives chacun en 3 variables linguistiques triangulaires sur un intervalle de position [-1800, 1800] et la vitesse [-60, 60] par les fonctions d'appartenance représentées sur les deux figures suivantes [41].

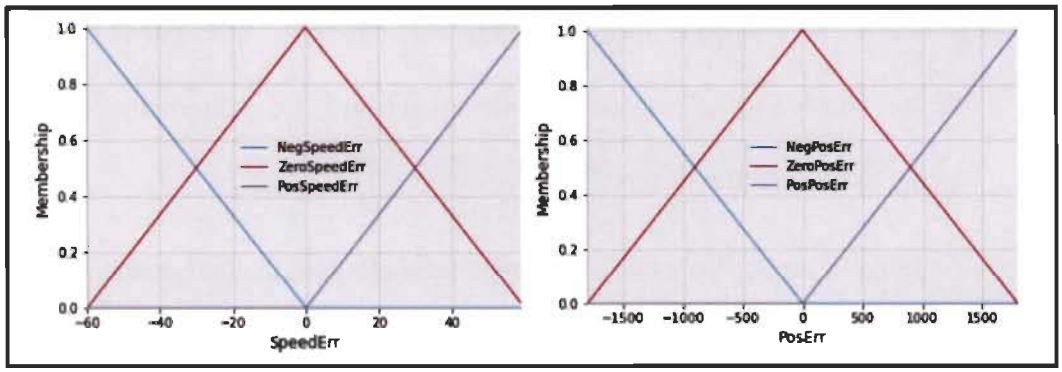

Figure 4-22 Variables linguistiques pour décrire la position (θ, $\mu(\theta)$) et la vitesse ($\boldsymbol{\Omega}$, $\mu(\boldsymbol{\Omega})$).

4.4.4.1.2 Inférence Floue de la position et de la vitesse

Le schéma d'inférence généré par le contrôleur flou de la position et de la vitesse angulaire du servomoteur permet la prise des décisions pour chaque règle activée et donne le sous-ensemble flou de sortie suivant.

Tableau 4-3 les règles de base pour la position et la vitesse du mouvement dans le servomoteur

	Les entrées		La sortie
Les règles	Erreur de Position	Erreur de Vitesse	Résultat PWM
Règle 1	Négative	Négative	Grande Négative
Règle 2	Négative	Zéro	Négative

Règle 3	Négative	Positive	Zéro
Règle 4	Zéro	Négative	Négative
Règle 5	Zéro	Zéro	Zéro
Règle 6	Zéro	Positive	Positive
Règle 7	Positive	Négative	Zéro
Règle 8	Positive	Zéro	Positive
Règle 9	Positive	Positive	Grande Positive

4.4.4.1.1 La défuzzification

Après, dans le processus de défuzzification (ou concrétisation), il s'agit d'utiliser la transformation à la sortie du bloc d'inférence et convertir les sous-ensembles flous de sortie en valeurs déterminées [41]. La figure suivante représente la réponse de commande de position et vitesse parla Logique Floue.

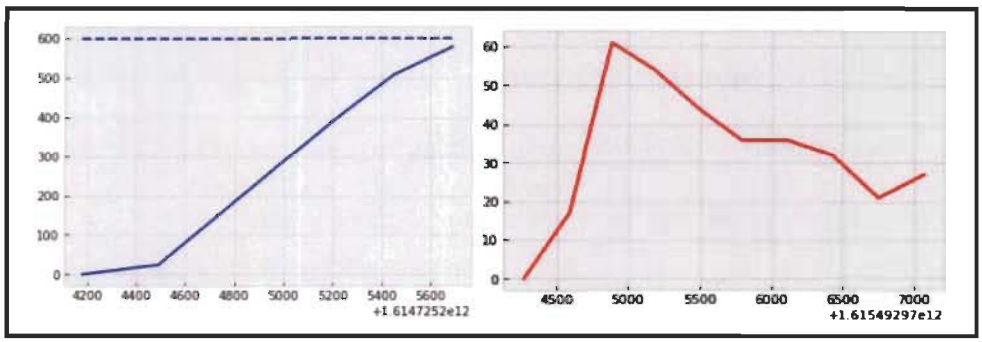

Figure 4-23 Résultat du double asservissement par la Logique Floue sur la position (bleu) et sur la vitesse (rouge).

4.5 Conclusion

Dans ce chapitre, nous avons abordé les différents aspects (matériel et logiciel) relatifs à la réalisation du projet, notamment les différents composants du bras robot. Nous avons étudié également la commande d'un servomoteur du bras robot utilisant des correcteurs de type PID et la commande par Logique Floue. Nous avons présenté des exemples des résultats expérimentaux obtenus pour chaque type de commande.

Chapitre 5 - Interprétations et Comparaison

5.1 Introduction

Dans un but comparatif, les différentes approches de commande étudiées et évaluées auparavant sont testées sur le prototype de servomoteur d'un bras manipulateur Lynxmotion. La procédure de comparaison consiste à vérifier les performances des lois de commande sous des conditions similaires d'opération (positions initiale et finale). Notons que bien que dans la simulation ceci est possible dans l'étude expérimentale deux conditions identiques de test ne seront jamais réalisables.

5.1.1 Commande de position

Les figures 5.1 et 5.2 montrent les résultats de tests des deux méthodes pour une commande de position en imposant une variation de 0° à 60°.

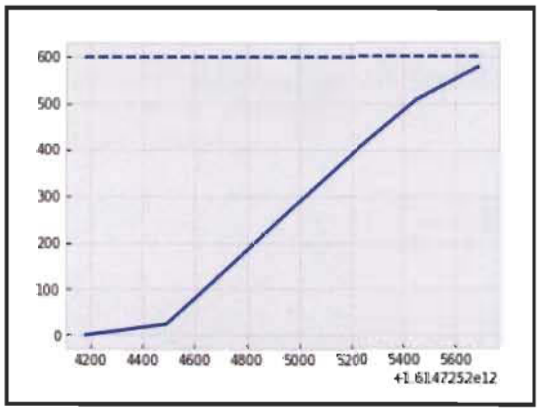

Figure 5-1 Logique Floue sur la position, passage de 0° à 60°.

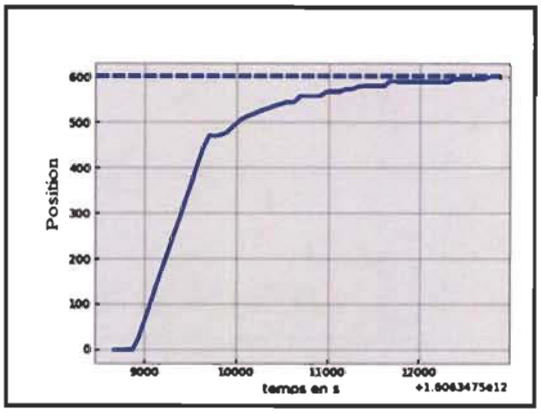

Figure 5-2 PID sur la position, passage de 0° à 60°.

Nous constatons que la rapidité du système (temps de réponse, temps de monté) dans la réponse obtenue par la commande par Logique Floue est meilleure que celle obtenue par la commande avec PID. Dans la commande par Logique Floue, la réduction de l'erreur et la stabilisation se font plus rapidement par contre avec la commande PID l'atténuation de l'erreur se fait lentement et présente une certaine oscillation (typique dans les correcteurs de type PID). Les deux régulateurs permettent d'éliminer l'erreur et le dépassement, donc peuvent être employés pour obtenir une bonne précision sans mouvements brusques ou saccadés.

5.1.2 Commande de vitesse

Les figures 5.3 et 5.4 montrent les résultats de tests des deux méthodes pour une commande de vitesse en imposant des variations de 0 à 20°/s, de 0 à 40°/s et de 0 à 60°/s.

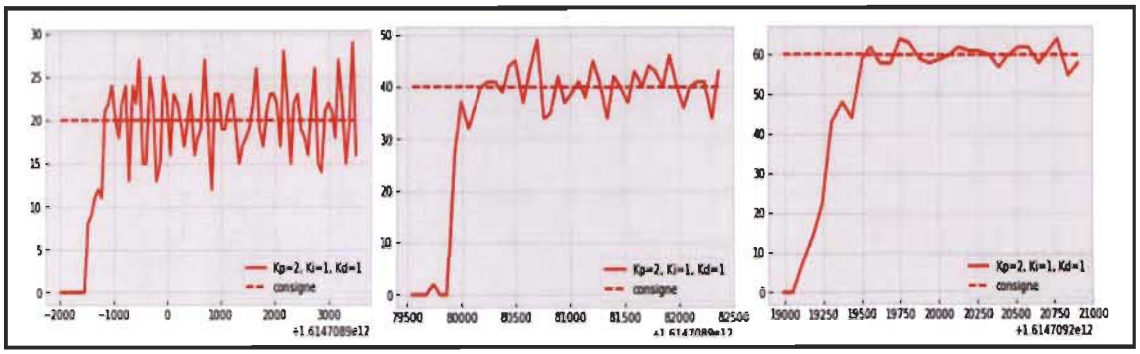

Figure 5-3 Résultats avec correcteur PID pour trois différentes consignes de vitesse angulaire 20°/s, 40°/s, 60°/s.

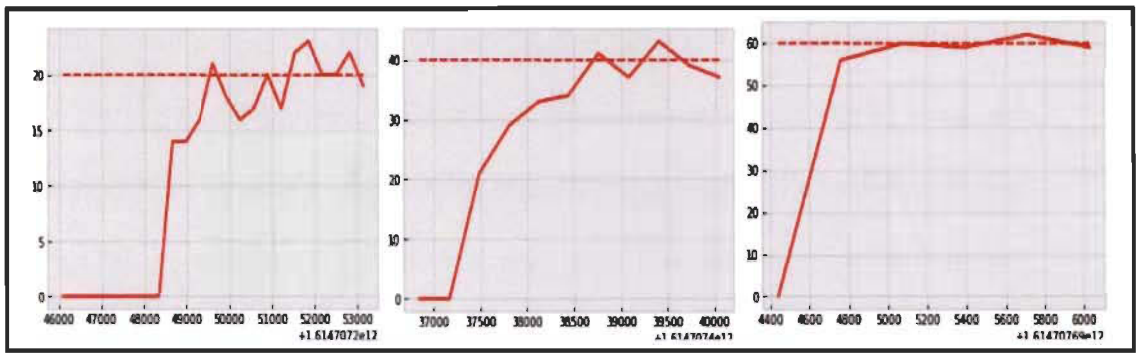

Figure 5-4 Résultats avec la Logique Floue pour trois différentes consignes de vitesse angulaire 20°/s, 40°/s, 60°/s.

Nous remarquons que les résultats obtenus avec la Logique Floue (d'après la figure 5.4) présentent moins de perturbation dans la vitesse du servomoteur par rapport aux résultats obtenus avec la commande PID (Figure 5-3). En revanche, le temps de réponse dans les faibles vitesses des mouvements de la commande PID plus rapide par rapport à la commande de vitesse par Logique Floue 20°/s, 40°/s. Mais dans les grandes vitesses 60°/s le temps de réponse de la vitesse de mouvement de la commande par logique Floue est plus rapide.

5.1.3 Double asservissement position-vitesse

Les figures 5.5 et 5.6 montrent les résultats de tests des deux méthodes pour une commande avec double asservissement position-vitesse en imposant des variations de 0 à 60° pour la commande PID et la commande par Logique Floue.

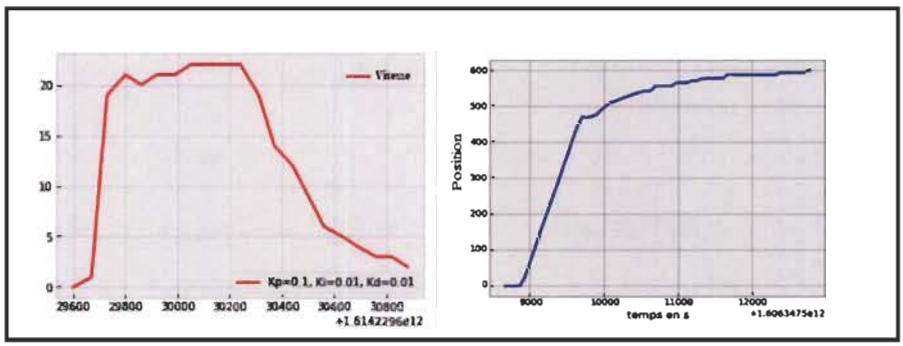

Figure 5-5 Résultats pour le double asservissement position-vitesse avec PID.

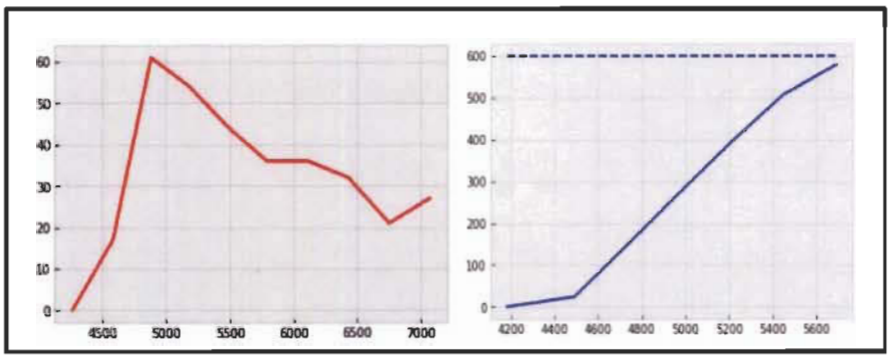

Figure 5-6 Résultats pour le double asservissement position-vitesse avec Logique Floue.

Nous constatons que la commande par la Logique Floue donne des meilleurs résultats à savoir: un bon suivi de la position et vitesses articulaires, ainsi qu'une meilleure convergence asymptotique de l'erreur. La commande avec PID peut être aussi un bon choix. Toutefois, la réponse risque d'être affectée si la charge mécanique du servomoteur varie. Dans ce cas, nous serons obligés de rajuster les paramètres (gains) du correcteur.

Selon les résultats obtenus, nous pouvons affirmer que la commande par la Logique Floue, en plus de donner des bonnes performances et assurer une convergence asymptotique de l'erreur, son ajustement n'est pas basé sur les caractéristiques ou paramètres du système. De sorte qu'elle peut rester fonctionnelle même si la charge du servomoteur change, ce qui est souvent le cas dans une application de robotique. La commande par la Logique Floue garantit à la fois l'atteinte du point de consigne et l'atténuation des oscillations.

La commande avec correcteur PID prend juste une seule vitesse constante qui est inférieure à la vitesse maximale du servomoteur, et si les changements de paramètres ne sont pas trop importants, elle peut rester fonctionnelle.

5.2 Conclusion

Dans ce chapitre, nous avons fait une comparaison des résultats expérimentaux entre le correcteur PID et la commande par la Logique Floue pour réguler la position et la vitesse d'une articulation (servomoteur) d'un bras robot.

Nous remarquons que la commande par la Logique Floue est très efficace, très performante (rapidité, stabilité et précision) par rapport au correcteur PID. Elle n'est pas affectée par les retards et les oscillations souvent rencontrées dans les correcteurs de type PID.

Chapitre 6 - Application de contrôle du robot sous Logiciel LABVIEW

6.1 Introduction

Un système numérique de contrôle-commande est un système de contrôle d'un procédé industriel doté d'une interface homme-machine pour la supervision et d'un réseau de communication numérique. Dans le but de commander notre bras manipulateur, nous avons proposé une application développée sous le logiciel Labview et l'environnement de développement IDLE Python. Celle-ci permet la commande manuelle et automatique des angles des organes du robot.

6.2 Présentation du logiciel Labview

Abréviation de LabVIEW (Laboratory Virtual Instrument Engineering Workbench) est le cœur d'une plate-forme de conception de système de mesure et de contrôle.

LabVIEW offre une approche de programmation graphique qui aide à visualiser tous les aspects de différentes applications, y compris la configuration matérielle, les données de mesure et le débogage.

Une des différences fondamentales de LabVIEW est que ce langage suit un modèle de flux de données, et non de flux d'instructions. Cela signifie que pour un langage textuel, ce sont les instructions qui ont la priorité, alors qu'avec LabVIEW, ce sont les données. Une fonction s'exécutera donc uniquement si elle dispose à ses entrées de toutes les données dont elle a besoin. Lorsqu'un langage classique est ainsi séquentiel, LabVIEW est naturellement

prédisposé au parallélisme. Ce qui augmente encore sa puissance et la rapidité d'exécution du code [42].

6.3 Types de commande du bras manipulateur

Afin de faciliter la communication entre l'homme et la machine, nous avons créé une interface permettant à l'utilisateur de commander le bras. Nous allons utiliser la méthode de programmation par auto-apprentissage [43].

Pour commander les quatre servomoteurs du bras manipulateur, nous avons réalisé deux modes de commande : une commande pour configurer la trajectoire du bras et une commande de marche automatique, la commande par des potentiomètres pour varier les angles des organes du robot, programmés directement via l'application LabVIEW.

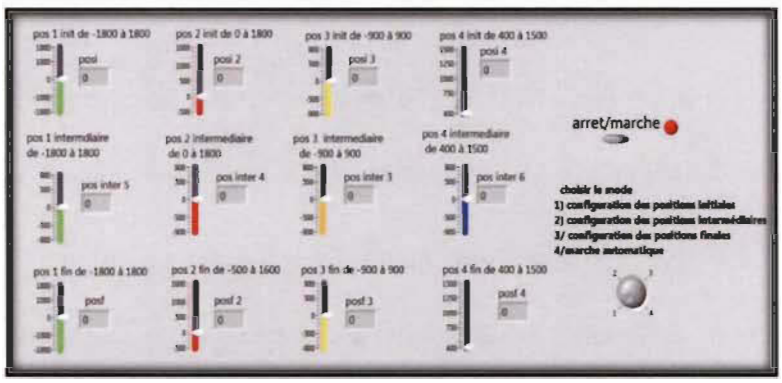

Figure 6-1 Interface de commande.

Partie 1 :

Cette partie consiste à :

- Un interrupteur (switch) qui permet d'activer ou désactiver l'application;

- Un sélecteur (switch) de 4 positions qui permet de donner l'accès de configurer la trajectoire du bras par : une case (1) accès pour configurer les positions initiales des servomoteurs, (2) accès pour les positions intermédiaire, (3) accès pour les positions finales, accès (4) marche automatique toutes les positions des servomoteurs.

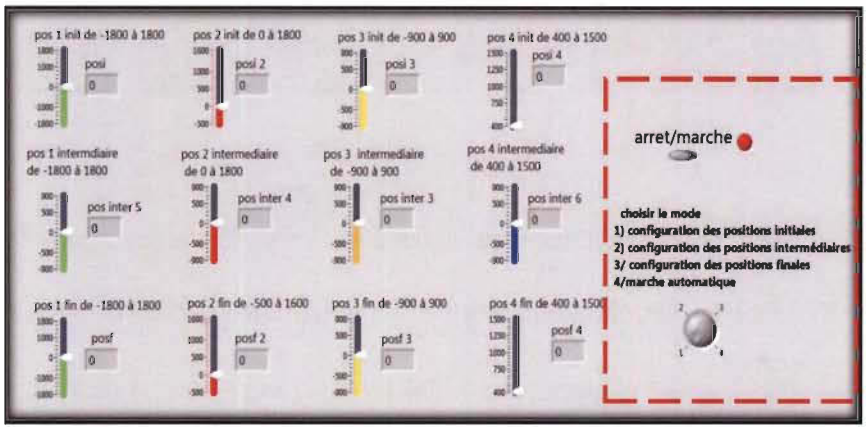

Figure 6-2 Interface de commande, rectangle pointillé : configurer le mode et marche automatique.

Partie 2 :

La commande par angles préprogrammés est un programme qui va nous permettre de suivre directement des trajectoires désirées. Nous pouvons ainsi, définir les différentes positions à l'aide de variation des potentiomètres (Figure 6-4), qui eux désignent des angles pour les servomoteurs, ce qui nous fournit une séquence de mouvement (par les positions des articulations) et nous définit une trajectoire à réaliser lors de la mise en marche automatique.

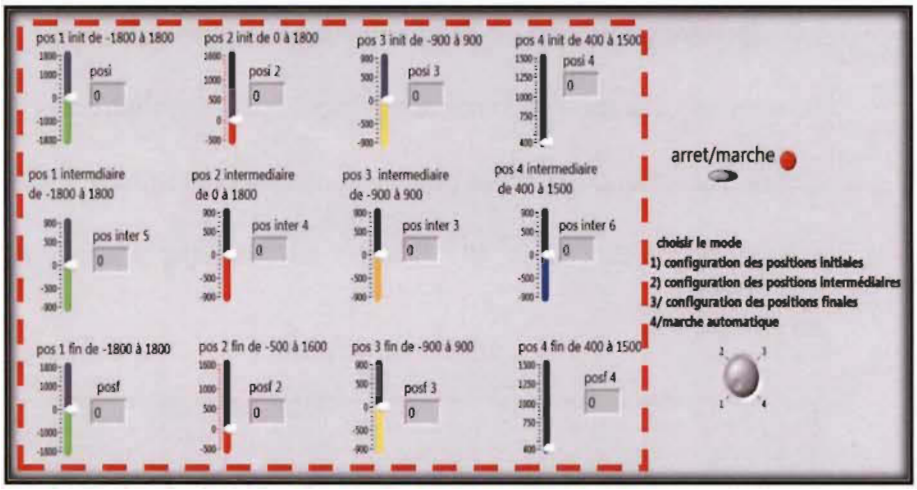

Figure 6-3 La commande par angles avec les potentiomètres.

Partie 3 : La séquence étant définie par les angles préprogrammés, dès que nous mettons le système en marche automatique, l'application exécute le programme sous Python pour exécuter la séquence de mouvement (prendre, déplacer et déposer des objets) avec le bras robot selon les positions sauvegardées. Dans ce sous-programme, nous illustrons une séquence répétitive pour réaliser le mouvement préprogrammé en faisant appel au programme Python (Figure 6-5).

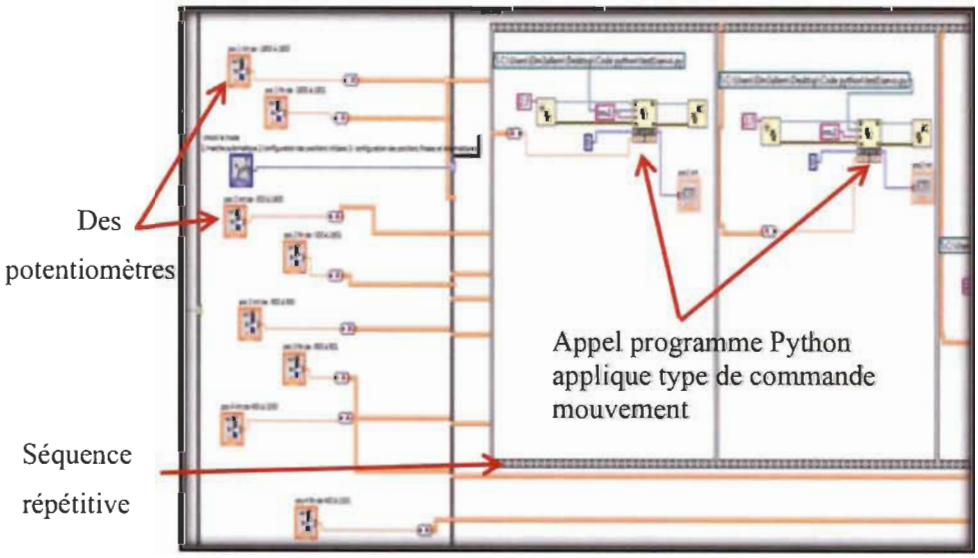

Figure 6-4 Diagramme bloc de l'interface LabVIEW avec Python.

6.4 Conclusion

Dans ce chapitre, nous avons proposé une application sous Labview et Python pour la commande du robot. Cette application permet essentiellement de :

- Configurer manuellement, à l'aide de potentiomètres, les positions souhaitées pour définir la trajectoire de mouvement à réaliser par le bras robot.
- Exécuter, de manière automatique et répétitive, les séquences de mouvements souhaitées.

Chapitre 7 - Conclusion Générale

L'objectif de ce mémoire était essentiellement d'étudier les lois de commande afin d'obtenir de meilleures performances et plus de robustesse pour générer un mouvement doux dans les bras robotisés à faible coût. Ce mémoire présente, donc de manière résumée, le travail qui concerne l'étude et la réalisation d'un bras manipulateur à trois degrés de liberté à faible coût, avec l'hypothèse d'un environnement de travail sans obstacles. Nous avons présenté une exploration générale du domaine de la robotique, les différents organes constituant le bras manipulateur, et la modélisation et l'élaboration des modèles géométriques, cinématiques, qui ont pour rôle respectivement de générer des trajectoires de références. Enfin, nous nous sommes intéressés à l'étude expérimentale de la commande de position et vitesse du robot en vue de vérifier les performances en termes de douceur ou finesse des mouvements.

Des résultats expérimentaux obtenus lors de l'étude pratique et l'implémentation des techniques de commande par correcteur PID et par Logique Floue nous ont permis de déterminer certains avantages et inconvénients de ces méthodes par rapport à l'objectif d'améliorer la douceur ou finesse des mouvements.

En termes généraux, la commande par la Logique Floue, dans le contexte de cette étude et sous les contraintes associées au matériel, offre des meilleures performances en termes de vitesse et atténuation de l'erreur soit par une commande simple de position ou soit par une commande considérant simultanément la vitesse et la position du servomoteur.

Afin de pouvoir réaliser les tests et contrôler le mouvement du bras robotique avec les régulateurs étudiés, une application d'interface utilisateur a été développée sous le logiciel LabVIEW.

Le prototype réalisé peut servir de base pour des travaux futurs qui seront destinés à l'amélioration du robot et de la fonctionnalité de ses diverses parties.

Les perspectives et recommandations suivantes permettront d'aller plus loin dans cette recherche dans des travaux futurs :

- Utiliser un matériau plus solide pour la réalisation du bras robot;
- Conduire l'étude sur différents types et qualités de servomoteurs;
- Faire une étude plus détaillée sur la modélisation dynamique directe et inverse, utilisant une commande via les observateurs robustes en présence des perturbations;
- Utiliser des servomoteurs offrant un couple moteur plus important, dans le but de soulever des objets plus lourds.

Bibliographie

[1] Bréan, Simon La science-fiction en France: théorie et histoire d'une littérature, PUPS, coll.« Lettres française » (2012).

[2] Liu, Lili. L'ergothérapie à l'ère de la quatrième révolution industrielle. Canadian Journal of Occupational Therapy. *85*(4), E1-E14 (2018).

[3] C. Bohn and D. P. Atherton, an analysis package comparing PID anti-windup strategies, IEEE Control Syst.Mag. 15, 34–40 (1995).

[4] Jingqing, Han. Acta Automatica Sinica. Nonlinear PID controller. *20*(4), 487-490 (1994).

[5] Xu, Yangming, Hollerbach, John M, & Ma, Donghai. A nonlinear PD controller for force and contact transient control. IEEE Control Systems Magazine. *15*(1), 15-21 (1995).

[6] Armstrong, Brian, McPherson, Joseph, & Li, Yonggang. *A Lyapunov stability proof for nonlinear-stiffness PD control.* Paper presented at the Proceedings of IEEE International Conference on Robotics and Automation. (1996).

[7] « Robotics », dans English Oxford Living Dictionaries. [En ligne]. Disponible: https://en.oxforddictionaries.com/definition/robotics

[8] Čapek, Karel. *RUR (Rossum's universal robots)*: Penguin. (2004).

[9] Huyssen, Andreas. After the great divide: Modernism, mass culture, postmodernism. The vamp and the machine: Fritz Lang's Metropolis. 65-81 (1986).

[10] Bessette, Juliette. Les Cahiers de l'École du Louvre. Recherches en histoire de l'art, histoire des civilisations, archéologie, anthropologie et muséologie. John McHale, l'Amérique passée à la machine. (13) (2019).

[11] Noh, Sun Young, & Jeong, Kyungmin. *Design Concepts of Emergency Response Robot Platform K-R2D2.* Paper presented at the Trans. of the Korean Nuclear Society, Autumn Meeting. (2016).

[12] Hurlbut, Shane, Snow, Ben, Gibson, Charlie, Alzmann, Christian. Engineering, & Technology. Terminator treatment. *4*(19), 45-47 (2009).

[13] Engelberger, Joseph F. *Robotics in practice: management and applications of industrial robots*: Springer Science & Business Media. (2012).

[14] Official website « IFR INTERNATIONAL FEDERATION OF ROBOTICS »

[15] Germain, Michel A, & Livernaux, Philippe. Bulletin de l'Académie Vétérinaire de France. Utilisation du robot Da Vinci-S® pour les sutures vasculaires et nerveuses chez l'animal: rat et porc. (2010).

[16] Raibert, Marc, Blankespoor, Kevin, Nelson, Gabriel, & Playter, Rob. Bigdog, the rough-terrain quadruped robot. IFAC Proceedings Volumes. *41*(2), 10822-10825. (2008).

[17] Karastoyanov, D, & Karastanev, S. Reuse of Industrial Robots. IFAC-PapersOnLine. *51*(30), 44-47 (2018).

[18] Omron Adept Technologie, Inc. Plat-Forme LD. In. États-Unis d'Amérique. (2018).

[19] Salameen, Laith, Estatieh, Abdelkarim, Darbisi, Salman, Tutunji, Tarek A, & Rawashdeh, Nathir A. *Interfacing Computing Platforms for Dynamic Control and Identification of an Industrial KUKA Robot Arm.* Paper presented at the 2020 21st International Conference on Research and Education in Mechatronics (REM). (2020).

[20] Boimond, Jean-Louis. « ROBOTIQUE » Cours, ISTIA, Université Angers, 2017.

[21] Lynxmotion, Official website. (2018). Retrieved from https://www.robotshop.com/info/wiki/lynxmotion/view/servo-erector-set-robots-kits/

[22] Barbulescu, Vlad, Marica, Ioan, Gheorghe, Viorel, Nistor, Mircea, & Patrascu, Monica. *Encoder-based path tracking with adaptive cascaded control for a three omni-wheel robot.* Paper presented at the 2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet). (2017).

[23] Boukli Hacene Lotfi Fazil « Commande et supervision d'un ensemble de robots via internet », Mémoire en vue de l'obtention de Magister, Université Des Sciences Et De La Technologie D'oran, 26 avril 2012.

[24] Ni, Zhichen, Liu, Juan, & Chu, Zhaoqi. *Multifunctional Trajectory Control System of Robot Based on PC.* Paper presented at the 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE). (2020).

[25] Luqman, Hafiz Muhammad, & Zaffar, Mubeen. *Chess Brain and Autonomous Chess Playing Robotic System.* Paper presented at the 2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC). (2016).

[26] Fisette P., Buyse H., Samin J.C. « Introduction à la robotique » Cours, 19 février 2004

[27] Craig, John J. « Introduction to robotics: mechanics and control », Pearson Education International, Livre, 2009.

[28] Gangloff Jacques « Cours de Robotique », Cours, ENSPS 3A Master ISTI, 2018.

[29] Angelles Jorge « Fundamentals of robotic mechanical systems », Springer, Livre, 2002.

[30] Emmanuel Godoy & Coll, Régulation industrielle. Dunod, Paris 2007.

[31] Zestedesavoir, Official website. (2021). Retrieved from; https://zestedesavoir.com/tutoriels/686/arduino-premiers-pas-en-informatique-embarquee/747_le-mouvement-grace-aux-moteurs/3438_un-moteur-qui-a-de-la-tete-le-servomoteur/#1-10705_principe-du-servomoteur.

[32] Gautron Laurent « PHYSIQUE TOUT LE COURS EN FICHES », Dunod, Livre,

[33] Pasquier Claude « Mécanique », Cours, Polytech Paris-Sud,2012.

[34] Lafond Roger « Analyse dynamique du mouvement », Cours,2018.

[35] Jacques GangloffCours de robotique de manipulation de Télécom Physique Strasbourg / master IRIV

[36] W.Khalil et E. Dombre, *Modélisation identification et commande des robots*, 2eéd. Paris : Hermès Sciences, 1999.

[37] Gotronic, Official website, Retrieved from;
https://www.gotronic.fr/art-carte-raspberry-pi-4-b-1-gb-30752.htm#:~:text=Cette%20carte%20est%20bas%C3%A9e%20sur,traitement%20de%20texte%2C%20jeux).

[38] Python, Official website. (2021). Retrieved from;
https://www.python.org/downloads/

[39] M. Camus, E. Deguine et D. Ross, *Régulation par PID,* Paris : TELECHOM Tech, 2010

[40] Martínez, José Román García, Reséndiz, Juvenal Rodríguez, Prado, Miguel Ángel Martínez, & Miguel, Edson Eduardo Cruz. *Assessment of jerk performance s-curve and trapezoidal velocity profiles.* Paper presented at the 2017 XIII International Engineering Congress (CONIIN). (2017).

[41] Zadeh, Lotfi Asker, Klir, George J, & Yuan, Bo. *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers* (Vol. 6): World Scientific. (1996).

[42] Cottet, Francis, Pinard, Michel et Desruelle, Luc, *LabVIEW Programmation et applications,* 3e éd. Paris: DUNOD, 2015.

[43] Igus, Official website. (2021). Retrieved from;
https://www.igus.fr/info/robot-software

[44] J.Graham, « CES 2017: Cars, robots are expected to star" dans USA TODAY, 2017. [Enligne]. Disponible : https://www.usatoday.com/story/tech/2016/12/30/cars-robotsexpected-star-ces2017/95890122/

[45] R. Gourdeau, ELE4203 — Robotique : Modélisation des Robots Manipulateurs, Département de génie ´électrique, Ecole Polytechnique de Montréal, 4 novembre 2010.

[46] Soria-Olivas, Emilio, Martín-Guerrero, José David, Camps-Valls, Gustavo, Serrano-López, Antonio J, Calpe-Maravilla, Javier, & Gómez-Chova, Luis %J IEEE Transactions on Neural Networks. (2003). A low-complexity fuzzy activation function for artificial neural networks. *14*(6), 1576-1579.

[47] R. W. Erickson, Fundamentals of Power Electronics, 2nd ed. Secaucus,NJ: Kluwer, 2000.

[48] Fuzzy logic Toolbox user guide.Natick, MA: Mathworks, 1997.

[49] Ghoul, Abdallah. Chapitre I Modélisation des robots manipulateurs, 2016.

Annexe

Annexe 1 : Code PID Position pour un seul servomoteur.

```
import lss
import numpy as np
import matplotlib.pyplot as plt
import lss_const as lssc
import time
import serial

CST_LSS_Port ="COM4"
CST_LSS_Baud =115200
CST_LSS_Baud = lssc.LSS_DefaultBaud
lss.initBus(CST_LSS_Port, CST_LSS_Baud)
myLSS1 = lss.LSS( )
#myLSS1.reset() #reset le servo moteur
time.sleep(5)# il prend 5 seconde apres le reset pour redémarrer

Kp=0.09
Ki=0.00001
Kd=0.3

i=0

myLSS1.move(0)

while(i <50):
    i=i+1
time.sleep(5)
Setpoint =1200
#myLSS1.setMaxSpeed(60)
position=[]
temps=[]
consigne=[]
plt.style.use('bmh')
plt.xlabel("temps en s")
plt.ylabel("Position")
plt.ion()

pos=0
Erreur=
pos=int(myLSS1.getPosition())
Erreur= Setpoint-int(myLSS1.getPosition())
period=0
x=

while(Setpoint-pos<-5or Setpoint-pos> ):
    pos =int(myLSS1.getPosition())
    position.append(pos)
    consigne.append(Setpoint)
    ErreurI= Setpoint-int(myLSS1.getPosition())
print("erreur",ErreurI,'\n')
    x=x+Erreur
    millis =int((time.time()*1000))
    temps.append(millis)
```

```
print("pos=",int(myLSS1.getPosition()))
print("speed=",int(myLSS1.getSpeed()))
print("voltage=",int(myLSS1.getVoltage()))
print("Curent = "+str(myLSS1.getCurrent()))
print("millis",millis)
print("            ")

print("millis-period", millis-period )
    Output=Kp*ErreurI+Ki*x+Kd*(ErreurI-Erreur)
print("output",Output)
    Erreur=ErreurI
    period=millis


    myLSS1.moveRelative(round(Output))

print("position",str(myLSS1.getPosition()),'\n')
plt.plot(temps,position,'b')# trace la courbe
plt.plot(temps,consigne,"b--")
plt.legend()
plt.draw()
plt.show()
plt.ioff()# on quitte le mode interactif pour rendre la main a
l'utilisateur sur la courbe
plt.show()
lss.closeBus()
```

Annexe 2 : Code PID Vitesse pour un seul servomoteur.

```
import lss
import numpy as np
import matplotlib.pyplot as plt
import lss_const as lssc
import time
import serial

CST_LSS_Port ="COM4"
CST_LSS_Baud =115200
CST_LSS_Baud = lssc.LSS_DefaultBaud
lss.initBus(CST_LSS_Port, CST_LSS_Baud)
myLSS1 = lss.LSS(1)
#myLSS1.reset() #reset le servo moteur
time.sleep(5)# il prend 5 seconde apres le reset pour redémarrer

#partie de la courbe

plt.style.use('bmh')
#plt.xlabel("temps en s")
#plt.ylabel("valeur")
#plt.ion()
#les valeurs du régulateur
Kp=2
Ki=1
Kd=1
#leer étape position 0
myLSS1.move(0)
#initialisation de la variable previous position à 0 pour mémoriser la
position précédente
prev_pos=0
#bouble while toujours vraie
while1:
    valeurs=[]#on y stoque les valeurs pour dessiner les courbes
    temps=[]
    vitesse=[]
    consigne=[]
print("write down the next position:")#1 angle en degré x 10 qu'on veut
atteindre
    Setpoint =int(input())#lecture de la valeur écrite par l'utilisateur
print("write the speed")#Speed degré par seconde
    set_speed =int(input())#lecture de la valeur écrite par l'utilisateur
    pv_speed=int(myLSS1.getSpeed())# la vitesse du servo
    pos =int(myLSS1.getPosition())#la position du servo
    e_speed_sum=0#somme des erreurs
    e_speed_pre=0#mémorisation de l'erreur précedent
    possomme=0#les degrés parcourus par le servo
print("setpoint",Setpoint,"prev_pos",prev_pos,"possomme",possomme)
while(Setpoint-pos<-6or Setpoint-pos>6)and(possomme<abs(Setpoint-
prev_pos)):
#condition d'arret (erreur entre -30 et 30 degrés)
#les degres parcours < la différence entre la derniere position et la
nouvelle
```

```
        pv_speed=int(myLSS1.getSpeed())#la vitesse du servo
        pos =int(myLSS1.getPosition())#la position du servo
        e_speed = set_speed - pv_speed #calcul de l'erreur de vitesse
        pwm_pulse = e_speed*Kp + e_speed_sum*Ki + e_speed - e_speed_pre
*Kd #la formule du régulateur PID
        e_speed_pre = e_speed   #mémorisation de l'erreur précedent
        e_speed_sum += e_speed  #somme des erreurs

if e_speed_sum >700:
          e_speed_sum =700#limitation d'intervalle la somme ne dépasse
pas 300

if e_speed_sum <-700:
          e_speed_sum =-700#limitation d'intervalle la somme ne dépasse
pas -300

        Erreur= Setpoint-int(myLSS1.getPosition())#calcul de l'erreur de
position

if(pwm_pulse <700and pwm_pulse >-700):#affectation de la pwm au moteur si
elle est dans
            myLSS1.moveRDM(int(pwm_pulse))#un intervalle acceptable de
300 et 300


elif pwm_pulse>700:#limitation d'intervalle pwm ne dépasse pas 300
            myLSS1.moveRDM(700)
            pwm_pulse=700

elif pwm_pulse<-700:#limitation d'intervalle pwm ne dépasse pas -300
            myLSS1.moveRDM(-700)
            pwm_pulse=-700

        myLSS1.moveRDM(int(pwm_pulse))


        valeurs.append(pos)#ajout des valeurs pour dessiner la courbe
        vitesse.append(pv_speed)#ajout des valeurs pour dessiner la
courbe
        consigne.append(set_speed)
        millis =int((time.time()*1000))
        temps.append(millis)#pour le traçage de la courbe
        possomme= possomme +abs(pos-int(myLSS1.getPosition()))
#calcule de la somme des degres parcourus à chaque itération de la boucle
while


print("position",str(myLSS1.getPosition()),'\n')
    prev_pos=Setpoint #memorisation de la derniere posirion dans la
variable prev_pos
#traçage de la courbe
    plt.plot(temps,vitesse,'r', label="Kp=2, Ki=1, Kd=1")# trace la
courbe
    plt.plot(temps,consigne,"r--",label="consigne")
    plt.legend()
```

```
    myLSS1.hold()
    plt.ioff()# on quitte le mode interactif pour rendre la main a
l'utilisateur sur la courbe
    plt.show()
    lss.closeBus()
```

Annexe 3 : Code double PID sur la position et la vitesse (trapézoïdale)pour un seul servomoteur

```
import lss
import numpy as np
import matplotlib.pyplot as plt
import lss_const as lssc
import time
import serial

CST_LSS_Port ="COM4"
CST_LSS_Baud =115200
CST_LSS_Baud = lssc.LSS_DefaultBaud
lss.initBus(CST_LSS_Port, CST_LSS_Baud)
myLSS1 = lss.LSS(2)
#myLSS1.reset() #reset le servo moteur
#time.sleep(5) # il prend 5 seconde apres le reset pour redémarrer
myLSS1.setAngularAcceleration(1)#l'accéleraton MAX
myLSS1.setAngularDeceleration(1)#la décéleration MAX
#myLSS1.setMax(10)#la Vitesse MAX

#pour le traçage de la courbe
plt.style.use('bmh')
#plt.xlabel("temps en s")
#plt.ylabel("valeur")
plt.ion()

#une fonction qui prends des position dans un intervalle in-min,in-max et
fait
#la régle de trois pour les calculer dans l'intervalle out-in,out-max
defmapnbr( x, in_min,  in_max, out_min, out_max):
return(x - in_min)*(out_max - out_min)/(in_max - in_min)+ out_min;
'''
#les position du regulateur
Kp=0.1
Ki=0.01
Kd=0.01'''
Kp=0.1
Ki=0.01
Kd=0.01
#1er étape position 0
myLSS1.move(0)
time.sleep(5)
#initialisation de la variable previous position à 0 pour mémoriser la
position précédente
prec_pos=0
#bouble while toujours vraie

while1:
    position=[]
    voltage=[]#on y stoque les position pour dessiner les courbes
    temps=[]
    Vitesse=[]
```

```
    courant=[]
    temperature=[]
    consigne=[]
    erreur_position=[]

print("write down the next position:")#l angle en degré x 10 qu'on veut
atteindre
    Setpoint =int(input())#lecture de la valeur écrite par l'utilisateur

print("\r\nQuerying LSS...")
    pos =int(myLSS1.getPosition())
    rpm=int(myLSS1.getSpeedRPM())
    curr =int(myLSS1.getCurrent())
    volt =int(myLSS1.getVoltage())
    temp =int(myLSS1.getTemperature())

    pwm_pulse=0
    e_pos_sum=0#somme des erreurs
    e_pos_pre=0#mémorisation de l'erreur précedent
    possomme=0#les degrés parcourus par le servo ,c'est une condition de
sécurité
    prec_pos=0#initialisation de la position précedente à 0
print("setpoint",Setpoint,"prec_pos",prec_pos,"possomme",possomme)
    V=0
    prec_pwm=0
while(Setpoint-pos<-3or Setpoint-pos> )and(possomme<abs(Setpoint-
prec_pos)):
#condition d'arret (erreur entre -2 et 2 degrés)
#les degres parcours < la différence entre la derniere position et la
nouvelle

        rpm=int(myLSS1.getSpeedRPM())#la Vitesse du servo
        pos =int(myLSS1.getPosition())#la position du servo
        e_pos = Setpoint -  pos #calcul de l'erreur de position
        pwm_pulse = e_pos*Kp + e_pos_sum*Ki + e_pos - e_pos_pre *Kd #la
formule du régulateur PID
        e_pos_pre = e_pos  #mémorisation de l'erreur précedent
        e_pos_sum += e_pos #somme des erreurs
print("pwm before map",pwm_pulse)#affichage de la sortie du PID
# if pwm_pulse >500:
#  pwm_pulse = 500 #limitation d'intervalle pwm ne dépasse pas 300

# if pwm_pulse<-500:
# pwm_pulse = -500  #limitation d'intervalle pwm ne dépasse pas -300

if(pwm_pulse >0):
         pwm=mapnbr( pwm_pulse,5,700,180,700)#calcul des valeur de sortie
du PID dans
#l'intervalle de fonctionnement du servo moteur
elif(pwm_pulse <0):
            pwm=mapnbr( pwm_pulse,-5,-700,-180,-700)#de meme pour les
position negatives

print("pwm",pwm)#affichage du résultat de la conversion

if e_pos_sum >700:
```

```
            e_pos_sum =700#limitation d'intervalle de la somme du PID (ne
dépasse pas 300)

if e_pos_sum <-700:
            e_pos_sum =-700#limitation d'intervalle de la somme du PID (ne
dépasse pas -300)

        Erreur= Setpoint-int(myLSS1.getPosition())#calcul de l'erreur de
position

#encrétage de la Vitesse

if pwm>700:#limitation d'intervalle pwm ne dépasse pas 300
            pwm=700

if pwm<-700:#limitation d'intervalle pwm ne dépasse pas -300
            pwm=-700

#encrétage de l'accéleration

if pwm-prec_pwm>700:
            pwm = prec_pwm+700
if pwm-prec_pwm<-700:
            pwm = prec_pwm-700



        myLSS1.moveRDM(int(pwm))#la commande de mouvement du servo moteur

        prec_pwm=pwm #memorisation de la dernière commande pwm


print("\r\n---- Telemetry ----")
print("Position  (1/10 deg) = "+str(pos));
print("Vitesse          (rpm) = "+str(rpm));
print("Curent           (mA) = "+str(curr));
print("Voltage          (mV) = "+str(volt));
print("Temperature (1/10 C) = "+str(temp));
#print("pos=",int(myLSS1.getPosition()))
#print("Vitesse",int(myLSS.getSpeedRPM()))
#print("Curent           (mA) = " + str(curr))
#print("Temperature (1/10 C) = " + str(temp))


        voltage.append(volt)#insérer dans la courbe
        position.append(pos)#ajout des position pour dessiner la courbe
#Vitesse.append(rpm)
        courant.append(curr)
        temperature.append(temp)
        consigne.append(Setpoint)
        Vitesse.append(rpm)
        millis =int((time.time()*1000))
        temps.append(millis)#pour le traçage de la courbe
#err_pos = int (abs(prec_pos - Setpoint))
        erreur_position.append(e_pos)
        possomme= possomme +abs(pos-int(myLSS1.getPosition()))
```

```
#calcule de la somme des degres parcourus à chaque itération de la boucle
while

    myLSS1.hold()
print("position",str(myLSS1.getPosition()),'\n')
    prec_pos=Setpoint #memorisation de la derniere posirion dans la
variable prec_pos
#traçage de la courbe
    plt.figure("Voltage")
    plt.plot(temps,voltage,"k", label="ligne -")
    plt.figure("position")
    plt.plot(temps,position,"*-", label="Kp=0.1, Ki=0.01, Kd=0.01")
    plt.plot(temps,consigne,"b--",label="consigne --")
    plt.legend()
    plt.figure("Vitesse")
    plt.plot(temps,Vitesse,"r", label="Kp=0.1, Ki=0.01, Kd=0.01")# trace
la courbe    #plt.subplot(3,1,3)
    plt.legend()
    plt.figure("courant")
    plt.plot(temps,courant,"g:", label="ligne :")
    plt.figure("température")
    plt.plot(temps,temperature,"o--",label="ligne --")
    plt.legend()
    plt.figure("erreur_position")
    plt.plot(temps,erreur_position,"k", label="ligne -")
    myLSS1.hold()
    plt.ioff()# on quitte le mode interactif pour rendre la main a
l'utilisateur sur la courbe
    plt.show()
lss.closeBus()
```

Annexe 4 : Code Fuzzy_logic sur la Position

```
import lss
import numpy as np
import matplotlib.pyplot as plt
import lss_const as lssc
import time
import serial
import skfuzzy as fuzz
from skfuzzy import control as ctrl

CST_LSS_Port ="COM4"
CST_LSS_Baud =115200
CST_LSS_Baud = lssc.LSS_DefaultBaud
lss.initBus(CST_LSS_Port, CST_LSS_Baud)
myLSS1 = lss.LSS(1)
myLSS1.move(0)
#attente 3 secondes
time.sleep(3)
#les entrées

Setpoint=600

#pour le traçage de la courbe
plt.style.use('bmh')
plt.xlabel("temps en ")
plt.ylabel("valeur")

#fonction pour le freinage
def test(speed):
    i=0
if(speed >=10and speed <=17):
       i=5#la marge d'erreur en degrés que le servo doit s'arreter avant
les atteindre
if(speed >17and speed <=23):
       i=8
if(speed >23and speed <=27):
       i=10
if(speed >27and speed <=33):
       i=13
return(i)

#on y stoque les valeurs pour dessiner les courbes
Vitesse=[]
position=[]
temps=[]
correction=[]
voltage=[]#on y stoque les position pour dessiner les courbes

courant=[]

consigne=[]
x1=0
#entrée erreur de vitesse entre -80 et 80
```

```
PosErr= ctrl.Antecedent(np.arange(-1800,1800,1),'PosErr')
#sortie rectification erreur PWM entre -50 et 50
RDMrectif=ctrl.Consequent(np.arange(-300,300,1),'RDMrectif')

#la définition des intervalles de l'entrée
PosErr['bigNegPosErr']=fuzz.trapmf(PosErr.universe,[-1800,-1800,-500,-
300])
PosErr['MedNegPosErr']=fuzz.trimf(PosErr.universe,[-300,-200,-100])
PosErr['SmallNegPosErr']=fuzz.trimf(PosErr.universe,[-100,-60,-20])
PosErr['VerySmallNegPosErr']=fuzz.trimf(PosErr.universe,[-20,-10,0])

PosErr['VerySmallPosPosErr']=fuzz.trimf(PosErr.universe,[0,10,20])
PosErr['SmallPosPosErr']=fuzz.trimf(PosErr.universe,[20,60,100])
PosErr['MedPosPosErr']=fuzz.trimf(PosErr.universe,[100,200,300])
PosErr['bigPosPosErr']=fuzz.trapmf(PosErr.universe,[300,500,1800,1800])

#la définition des intervalles de la sortie

RDMrectif['bigNegRDMrectif']=fuzz.trapmf(RDMrectif.universe,[-300,-300,-
285,-270])
RDMrectif['MedNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-270,-235,-
200])
RDMrectif['SmallNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-200,-190,-
180])
RDMrectif['VerySmallNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-180,-
175,-170])

RDMrectif['VerySmallPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[170,175
,180])
RDMrectif['SmallPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[180,190,200
])
RDMrectif['MedPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[200,235,270])
RDMrectif['bigPosRDMrectif']=fuzz.trapmf(RDMrectif.universe,[270,285,300,
300])

#dessiner les courbes des définitions
PosErr.view()
RDMrectif.view()
#les lois à appliquer entre la sortie et l'entrée
rule1 = ctrl.Rule(PosErr['bigNegPosErr'],RDMrectif['bigNegRDMrectif'])
rule2 = ctrl.Rule(PosErr['MedNegPosErr'],RDMrectif['MedNegRDMrectif'])
rule3 =
ctrl.Rule(PosErr['SmallNegPosErr'],RDMrectif['SmallNegRDMrectif'])
rule4 =
ctrl.Rule(PosErr['VerySmallNegPosErr'],RDMrectif['VerySmallNegRDMrectif']
)
rule5 =
ctrl.Rule(PosErr['VerySmallPosPosErr'],RDMrectif['VerySmallPosRDMrectif']
)
rule6 =
ctrl.Rule(PosErr['SmallPosPosErr'],RDMrectif['SmallPosRDMrectif'])
rule7 = ctrl.Rule(PosErr['MedPosPosErr'],RDMrectif['MedPosRDMrectif'])
rule8 = ctrl.Rule(PosErr['bigPosPosErr'],RDMrectif['bigPosRDMrectif'])

#faire entrer les lois dans le systeme fuzzy
```

```
perf_ctrl=ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,r
ule8])
perf=ctrl.ControlSystemSimulation(perf_ctrl)


pos =int(myLSS1.getPosition())#la position du servo
pwm_pulse=0
possomme=0
#prev_pos=0

while(Setpoint-pos<-7or Setpoint-pos>7)and(possomme<abs(Setpoint)):
#ajout des valeurs pour dessiner la courbe
   position.append(int(myLSS1.getPosition()))
   Vitesse.append(int(myLSS1.getSpeed()))
   curr =int(myLSS1.getCurrent())
   millis =int((time.time()*1000))
   temps.append(millis)
   correction.append(pwm_pulse)
   consigne.append(Setpoint)
   courant.append(curr)

#l'entrée au fuzzy l'erreur de position
   perf.input['PosErr']=(Setpoint-int(myLSS1.getPosition()))

print("pos1=",int(myLSS1.getPosition()))

if(abs(Setpoint-
int(myLSS1.getPosition()))<=test(abs(int(myLSS1.getSpeed())))):
       myLSS1.moveRDM(0)
       myLSS1.hold()
print("out if1",int(myLSS1.getPosition()),"
",test(int(myLSS1.getSpeed())))
break


#fuzzy commence le calcul
   perf.compute()


print("pos0=",int(myLSS1.getPosition()))
if(abs(Setpoint-
int(myLSS1.getPosition()))<=test(abs(int(myLSS1.getSpeed())))):
       myLSS1.moveRDM(0)
       myLSS1.hold()
print("out if0",int(myLSS1.getPosition()),"
",test(int(myLSS1.getSpeed())))
break

#la somme des réctifications fuzzy
   pwm_pulse=int(perf.output['RDMrecti '])


if pwm_pulse>700:#limitation d'intervalle pwm ne dépasse pas 300
            myLSS1.moveRDM(700)
            pwm_pulse=700
```

```
elif pwm_pulse<-700:#limitation d'intervalle pwm ne dépasse pas -300
            myLSS1.moveRDM(-700)
            pwm_pulse=-700

#affectation de la sortie au servo moteur
   myLSS1.moveRDM(int(pwm_pulse))

#condition d'arret ,atteinte de la position
print("pos3=",int(myLSS1.getPosition()))
if(abs(Setpoint-
int(myLSS1.getPosition()))<=test(abs(int(myLSS1.getSpeed()))):
       myLSS1.moveRDM(0)
       myLSS1.hold()
print("out if3",int(myLSS1.getPosition()),"
",test(int(myLSS1.getSpeed())))
break

   possomme= possomme +abs(pos-int(myLSS1.getPosition()))
#calcule de la somme des degres parcourus à chaque itération de la boucle
while
   pos =int(myLSS1.getPosition())#la position du servo



myLSS1.hold()
possomme= possomme +abs(pos-int(myLSS1.getPosition()))
print("position finale=",int(myLSS1.getPosition()))
print("Currant          (mA) = "+str(curr));
x1=i

plt.figure("position")
plt.plot(temps,position,'b')
plt.plot(temps,consigne,"b--")
#plt.legend()
plt.figure("Vitesse")
plt.plot(temps,Vitesse,' ')# trace la courbe    #plt.subplot(3,1,3)
#plt.legend()
plt.figure("courant")
plt.plot(temps,courant,'g')
#plt.legend()
plt.figure("correction")
plt.plot(temps,correction,'k')
#plt.legend()
plt.show()
lss.closeBus()
```

Annexe 5 : Code Fuzzy_logic sur la Vitesse

```
import lss
import numpy as np
import matplotlib.pyplot as plt
import lss_const as lssc
import time
import serial
import skfuzzy as fuzz
from skfuzzy import control as ctrl

CST_LSS_Port ="COM4"
CST_LSS_Baud =115200
CST_LSS_Baud = lssc.LSS_DefaultBaud
lss.initBus(CST_LSS_Port, CST_LSS_Baud)
myLSS1 = lss.LSS(1)
myLSS1.move(0)
#attente 3 secondes
time.sleep(3)
#les entrées

SetSpeed=60

Setpoint=1000

#pour le traçage de la courbe
plt.style.use('bmh')
plt.xlabel("temps en s")
plt.ylabel("valeur")

def test(speed):
    i=0
if(speed >=10and speed <=17):
        i=5
if(speed >17and speed <=23):
        i=7
if(speed >23and speed <=27):
        i=10
if(speed >27and speed <=33):
        i=13
return(i)

#on y stoque les valeurs pour dessiner les courbes
Vitesse=[]
position=[]
temps=[]
consigne=[]
correction=[]
#entrée erreur de vitesse entre -80 et 80
SpeedErr= ctrl.Antecedent(np.arange(-60,60,1),'SpeedErr')
#sortie rectification erreur PWM entre -50 et 50
RDMrectif=ctrl.Consequent(np.arange(-700,700,1),'RDMrectif')
#la définition des intervalles de l'entrée
SpeedErr['NegSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,-60,0])
```

```
SpeedErr['ZeroSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,0,60])
SpeedErr['PosSpeedErr']=fuzz.trimf(SpeedErr.universe,[0,60,60])

#la définition des intervalles de la sortie
RDMrectif['NegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-700,0])
RDMrectif['ZeroRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,0,700])
RDMrectif['PosRDMrectif']=fuzz.trimf(RDMrectif.universe,[0,700,700])

#dessiner les courbes des définitions
SpeedErr.view()
RDMrectif.view()
#les lois à appliquer entre la sortie et l'entrée
rule1 = ctrl.Rule(SpeedErr['NegSpeedErr'],RDMrectif['NegRDMrectif'])
rule2 = ctrl.Rule(SpeedErr['ZeroSpeedErr'],RDMrectif['ZeroRDMrectif'])
rule3 = ctrl.Rule(SpeedErr['PosSpeedErr'],RDMrectif['PosRDMrectif'])

#faire entrer les lois dans le systeme fuzzy
perf_ctrl=ctrl.ControlSystem([rule1,rule2,rule3])
perf=ctrl.ControlSystemSimulation(perf_ctrl)


pos =int(myLSS1.getPosition())#la position du servo
pwm_pulse=0
possomme=0
#prev_pos=0

while(Setpoint-pos<-3or Setpoint-pos>3)and(possomme<abs(Setpoint)):


#l'entrée au fuzzy l'erreur de vitesse
   perf.input['SpeedErr']=(SetSpeed-int(myLSS1.getSpeed()))

print("pos1=",int(myLSS1.getPosition()))
if(Setpoint-int(myLSS1.getPosition())<=test(int(myLSS1.getSpeed()))):
       myLSS1.moveRDM( )
       myLSS1.hold()
print("out if1",int(myLSS1.getPosition()),"  i
",test(int(myLSS1.getSpeed())))
break

#fuzzy commence le calcul
   perf.compute()
print("pos2=",int(myLSS1.getPosition()))
if(Setpoint-int(myLSS1.getPosition())<=test(int(myLSS1.getSpeed()))):
       myLSS1.moveRDM( )
       myLSS1.hold()
print("out if2",int(myLSS1.getPosition()),"
",test(int(myLSS1.getSpeed())))
break


#ajout des valeurs pour dessiner la courbe
   position.append(int(myLSS1.getPosition()))
   Vitesse.append(int(myLSS1.getSpeed()))
   millis =int((time.time()*1000))
   temps.append(millis)
```

```
    consigne.append(SetSpeed)
    correction.append(int(perf.output['RDMrectif']))

print("pos0=",int(myLSS1.getPosition()))
if(Setpoint-int(myLSS1.getPosition())<=test(int(myLSS1.getSpeed()))):
        myLSS1.moveRDM(0)
        myLSS1.hold()
print("out if0",int(myLSS1.getPosition()),"
",test(int(myLSS1.getSpeed())))
break

#la somme des rectifications fuzzy
    pwm_pulse=pwm_pulse+int(perf.output['RDMrectif ])

if(pwm_pulse <700and pwm_pulse >-700):#affectation de la pwm au moteur si
elle est dans
            myLSS1.moveRDM(int(pwm_pulse))#un intervalle acceptable de -
300 et 300


elif pwm_pulse>700:#limitation d'intervalle pwm ne dépasse pas 300
            myLSS1.moveRDM(700)
            pwm_pulse=700

elif pwm_pulse<-700:#limitation d'intervalle pwm ne dépasse pas -300
            myLSS1.moveRDM(-700)
            pwm_pulse=-700

#affectation de la sortie au servo moteur
    myLSS1.moveRDM(int(pwm_pulse))

#condition d'arret ,atteinte de la position
print("pos3=",int(myLSS1.getPosition()))
if(Setpoint-int(myLSS1.getPosition())<=test(int(myLSS1.getSpeed()))):
        myLSS1.moveRDM(0)
        myLSS1.hold()
print("out if3",int(myLSS1.getPosition()),"
",test(int(myLSS1.getSpeed())))
break

    possomme= possomme +abs(pos-int(myLSS1.getPosition()))
#calcule de la somme des degres parcourus à chaque itération de la boucle
while
    pos =int(myLSS1.getPosition())#la position du servo

print("pos4=",int(myLSS1.getPosition()))
if(Setpoint-int(myLSS1.getPosition())<=test(int(myLSS1.getSpeed()))):
        myLSS1.moveRDM(0)
        myLSS1.hold()
print("out if4",int(myLSS1.getPosition()),"
",test(int(myLSS1.getSpeed())))
break


myLSS1.hold()
possomme= possomme +abs(pos-int(myLSS1.getPosition()))
```

```
print("pos=",int(myLSS1.getPosition()))

plt.figure("position")
plt.plot(temps,position,'b')
plt.figure("Vitesse")
plt.plot(temps,Vitesse,'r')
plt.plot(temps,consigne,"r--")
plt.figure("correction")
plt.plot(temps,correction,'k')
plt.show()
lss.closeBus()
```

Annexe 6 : Code double Fuzzy_logic sur la position et la vitesse(trapézoïdale) pour un seul servomoteur

```
import lss
import numpy as np
import matplotlib.pyplot as plt
import lss_const as lssc
import time
import serial
import skfuzzy as fuzz
from skfuzzy import control as ctrl

CST_LSS_Port ="COM4"
CST_LSS_Baud =115200
CST_LSS_Baud = lssc.LSS_DefaultBaud
lss.initBus(CST_LSS_Port, CST_LSS_Baud)
myLSS1 = lss.LSS(1)
myLSS1.move(0)
#attente 3 secondes
time.sleep(3)
#les entrées
#-500 =>1600
Setpoint=1500

#pour le traçage de la courbe
plt.style.use('bmh')
plt.xlabel("temps en s")
plt.ylabel("valeur")

#fonction pour le freinage
def test2(speed):
    i=0
if(speed >=20and speed <=30):
        i=20#la marge d'erreur en degrés que le servo doit s'arreter avant
les atteindre
if(speed >10and speed <=20):
        i=10
if(speed >40and speed <=50):
        i=30
if(speed >50and speed <=60):
        i=40
return(i)

#on y stoque les valeurs pour dessiner les courbes
Vitesse=[]
position=[]
temps=[]
correction=[]
voltage=[]#on y stoque les position pour dessiner les courbes

courant=[]

consigne=[]
```

```
#x1=0
#entrée erreur de position entre -1800 et 1800
PosErr= ctrl.Antecedent(np.arange(-1800,1800,1),'PosErr')

#entrée erreur de vitesse entre -80 et 80
SpeedErr= ctrl.Antecedent(np.arange(-60,60,1),'SpeedErr')

#sortie rectification erreur PWM entre -50 et 50
RDMrectif=ctrl.Consequent(np.arange(-700,700,1),'RDMrectif')

#la définition des intervalles de l'entrée Position

PosErr['NegPosErr']=fuzz.trimf(PosErr.universe,[-1800,-1800,0])
PosErr['ZeroPosErr']=fuzz.trimf(PosErr.universe,[-1800,0,1800])
PosErr['PosPosErr']=fuzz.trimf(PosErr.universe,[0,1800,1800])

#la définition des intervalles de l'entrée Speed

SpeedErr['NegSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,-60,0])
SpeedErr['ZeroSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,0,60])
SpeedErr['PosSpeedErr']=fuzz.trimf(SpeedErr.universe,[0,60,60])

#la définition des intervalles de la sortie

RDMrectif['bigNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-700,-
500])
RDMrectif['NegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-500,0])
RDMrectif['ZeroRDMrectif']=fuzz.trimf(RDMrectif.universe,[-500,0,500])
RDMrectif['PosRDMrectif']=fuzz.trimf(RDMrectif.universe,[0,500,700])
RDMrectif['bigPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[500,700,700])


#dessiner les courbes des définitions
PosErr.view()
SpeedErr.view()
RDMrectif.view()

#les lois à appliquer entre la sortie et l'entrée
rule1 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['bigNegRDMrectif'])
rule2 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['NegRDMrectif'])
rule3 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['ZeroRDMrectif'])
rule4 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['NegRDMrectif'])
rule5 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['ZeroRDMrectif'])
rule6 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['PosRDMrectif'])
rule7 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['ZeroRDMrectif'])
rule8 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['PosRDMrectif'])
rule9 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['bigPosRDMrectif'])
```

```
#faire entrer les lois dans le systeme fuzzy
perf_ctrl=ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,r
ule8,rule9])
perf=ctrl.ControlSystemSimulation(perf_ctrl)


pos =int(myLSS1.getPosition())#la position du servo
pwm_pulse=0
possomme=0
prec_pwm=0
#prev_pos=0

if(Setpoint-int(myLSS1.getPosition())<=0):
    SetSpeed=-30

if(Setpoint-int(myLSS1.getPosition())>0):
    SetSpeed=30

while(Setpoint-pos<-5or Setpoint-pos>5):
if(Setpoint-int(myLSS1.getPosition())<=0):
        SetSpeed=-10

if(Setpoint-int(myLSS1.getPosition())>0):
        SetSpeed=10
#ajout des valeurs pour dessiner la courbe
   position.append(int(myLSS1.getPosition()))
   Vitesse.append(int(myLSS1.getSpeed()))
   curr =int(myLSS1.getCurrent())
   millis =int((time.time()*1000))
   temps.append(millis)
   correction.append(pwm_pulse)
   consigne.append(Setpoint)
   courant.append(curr)

#l'entrée au fuzzy l'erreur de vitesse
   perf.input['SpeedErr']=(SetSpeed-int(myLSS1.getSpeed()))
#l'entrée au fuzzy l'erreur de position
   perf.input['PosErr']=(Setpoint-int(myLSS1.getPosition()))

print("vitesse",int(myLSS1.getSpeed()))

if(abs(Setpoint-
int(myLSS1.getPosition()))<=test2(abs(int(myLSS1.getSpeed())))):
       myLSS1.moveRDM(0)
       myLSS1.hold()
print("out if1",int(myLSS1.getPosition()),"
",test2(int(myLSS1.getSpeed())))
break


#fuzzy commence le calcul
   perf.compute()
```

```
print("pos2=",int(myLSS1.getPosition()))
if(abs(Setpoint-
int(myLSS1.getPosition()))<=test2(abs(int(myLSS1.getSpeed()))):
        myLSS1.moveRDM(0)
        myLSS1.hold()
print("out if2",int(myLSS1.getPosition()),"
",test2(int(myLSS1.getSpeed())))
break

#la somme des rectifications fuzzy
   pwm_pulse=pwm_pulse+int(perf.output['RDMrectif'])
print("pwm1=",pwm_pulse)
print("speed=",int(myLSS1.getSpeed()))
if pwm_pulse>500:#limitation d'intervalle pwm ne dépasse pas 300
            myLSS1.moveRDM(700)
            pwm_pulse=700

elif pwm_pulse<-700:#limitation d'intervalle pwm ne dépasse pas -300
            myLSS1.moveRDM(-700)
            pwm_pulse=-700

#affectation de la sortie au servo moteur
   myLSS1.moveRDM(int(pwm_pulse))
print("pwm2=",pwm_pulse)
#condition d'arret ,atteinte de la position
print("pos3=",int(myLSS1.getPosition()))
if(abs(Setpoint-
int(myLSS1.getPosition()))<=test2(abs(int(myLSS1.getSpeed()))):
        myLSS1.moveRDM(0)
        myLSS1.hold()
print("out if3",int(myLSS1.getPosition()),"
",test2(int(myLSS1.getSpeed())))
break

   possomme= possomme +abs(pos-int(myLSS1.getPosition()))
#calcule de la somme des degres parcourus à chaque itération de la boucle
while
   pos =int(myLSS1.getPosition())#la position du servo

myLSS1.hold()
#possomme= possomme +abs(pos-int(myLSS1.getPosition()))
print("position finale=",int(myLSS1.getPosition()))
print("Currant          (mA) = "+str(curr));
#x1=1

plt.figure("position")
plt.plot(temps,position,'b')
plt.plot(temps,consigne,"b--")
#plt.legend()
plt.figure("Vitesse")
plt.plot(temps,Vitesse,'r')# trace la courbe    #plt.subplot(3,1,3)
#plt.legend()
plt.figure("courant")
plt.plot(temps,courant,'g')
#plt.legend()
```

```
plt.figure("correction")
plt.plot(temps,correction,'k')
#plt.legend()
plt.show()
lss.closeBus()
```

Annexe 7 : Code double Fuzzy_logic sur la position et la vitesse(trapézoïdale) pour tous les servomoteurs du bras

```
import lss
import numpy as np
import matplotlib.pyplot as plt
import lss_const as lssc
import time
import serial
import skfuzzy as fuzz
from skfuzzy import control as ctrl

CST_LSS_Port ="COM4"
CST_LSS_Baud =115200
CST_LSS_Baud = lssc.LSS_DefaultBaud
lss.initBus(CST_LSS_Port, CST_LSS_Baud)

myLSS1 = lss.LSS(1)
myLSS2 = lss.LSS(2)
myLSS3 = lss.LSS(3)
myLSS4 = lss.LSS(4)

def test2(speed):
    i=0
if(speed >=20and speed <=30):
        i=20
if(speed >10and speed <=20):
        i=20
if(speed >40and speed <=50):
        i=30
if(speed >50and speed <=60):
        i=40
return(i)


def test(speed):
    i=0
if(speed >=10and speed <=17):
        i=5
if(speed >17and speed <=23):
        i=8
if(speed >23and speed <=27):
        i=10
if(speed >27and speed <=33):
        i=13
return(i)


    Setpoint=1000

    PosErr= ctrl.Antecedent(np.arange(-3600,3600,1),'PosErr')

    SpeedErr= ctrl.Antecedent(np.arange(-60,60,1),'SpeedErr')
```

```
    RDMrectif=ctrl.Consequent(np.arange(-700,700,1),'RDMrectif')



    PosErr['NegPosErr']=fuzz.trimf(PosErr.universe,[-3600,-3600,0])
    PosErr['ZeroPosErr']=fuzz.trimf(PosErr.universe,[-3600,0,3600])
    PosErr['PosPosErr']=fuzz.trimf(PosErr.universe,[0,3600,3600])



    SpeedErr['NegSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,-60,0])
    SpeedErr['ZeroSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,0,60])
    SpeedErr['PosSpeedErr']=fuzz.trimf(SpeedErr.universe,[0,60,60])


    RDMrectif['bigNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-
700,-500])
    RDMrectif['NegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-
500,0])
    RDMrectif['ZeroRDMrectif']=fuzz.trimf(RDMrectif.universe,[-
500,0,500])
    RDMrectif['PosRDMrectif']=fuzz.trimf(RDMrectif.universe,[0,500,700])

RDMrectif['bigPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[500,700,700])


    rule1 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['bigNegRDMrectif'])
    rule2 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['NegRDMrectif'])
    rule3 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule4 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['NegRDMrectif'])
    rule5 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule6 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['PosRDMrectif'])
    rule7 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule8 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['PosRDMrectif'])
    rule9 =
ctrl.Rule(PosErr['PosPosErr']&SpeedErr['PosSpeedErr'],RDMrectif['bigPosRD
Mrectif'])


perf_ctrl=ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,r
ule8,rule9])
    perf=ctrl.ControlSystemSimulation(perf_ctrl)


    pos =int(myLSS1.getPosition())
    pwm_pulse=0
```

```
    possomme=0
    prec_pwm=0


if(Setpoint-int(myLSS1.getPosition())<=0):
        SetSpeed=-30

if(Setpoint-int(myLSS1.getPosition())>0):
        SetSpeed=30

while(Setpoint-pos<-5or Setpoint-pos>5):
if(Setpoint-int(myLSS1.getPosition())<=0):
            SetSpeed=-10

if(Setpoint-int(myLSS1.getPosition())>0):
            SetSpeed=10



       perf.input['SpeedErr']=(SetSpeed-int(myLSS1.getSpeed()))

       perf.input['PosErr']=(Setpoint-int(myLSS1.getPosition()))

print("vitesse",int(myLSS1.getSpeed()))

if(abs(Setpoint-
int(myLSS1.getPosition()))<=test2(abs(int(myLSS1.getSpeed()))):
           myLSS1.moveRDM(0)
           myLSS1.hold()
print("out if1",int(myLSS1.getPosition())," i
",test2(int(myLSS1.getSpeed())))
break



       perf.compute()


print("pos2=",int(myLSS1.getPosition()))
if(abs(Setpoint-
int(myLSS1.getPosition()))<=test2(abs(int(myLSS1.getSpeed()))):
           myLSS1.moveRDM(0)
           myLSS1.hold()
print("out if2",int(myLSS1.getPosition())," i
",test2(int(myLSS1.getSpeed())))
break


       pwm_pulse=pwm_pulse+int(perf.output['RDMrectif'])
print("pwm1=",pwm_pulse)
print("speed=",int(myLSS1.getSpeed()))
if pwm_pulse>700:
                myLSS1.moveRDM(700)
                pwm_pulse=700
```

```
elif pwm_pulse<-700:
                myLSS1.moveRDM(-700)
                pwm_pulse=-700



        myLSS1.moveRDM(int(pwm_pulse))
print("pwm2=",pwm_pulse)

print("pos3=",int(myLSS1.getPosition()))
if(abs(Setpoint-
int(myLSS1.getPosition()))<=test2(abs(int(myLSS1.getSpeed()))):
            myLSS1.moveRDM(0)
            myLSS1.hold()
print("out if3",int(myLSS1.getPosition()),"  1
",test2(int(myLSS1.getSpeed())))
break

        possomme= possomme +abs(pos-int(myLSS1.getPosition()))

        pos =int(myLSS1.getPosition())



    myLSS1.hold()

print("position finale=",int(myLSS1.getPosition()))
    lss.closeBus()
return(pos)


    Setpoint=1000

    PosErr= ctrl.Antecedent(np.arange(-1800,1800,1),'PosErr')


    SpeedErr= ctrl.Antecedent(np.arange(-60,60,1),'SpeedErr')


    RDMrectif=ctrl.Consequent(np.arange(-700,700,1),'RDMrectif')



    PosErr['NegPosErr']=fuzz.trimf(PosErr.universe,[-1800,-1800,0])
    PosErr['ZeroPosErr']=fuzz.trimf(PosErr.universe,[-1800,0,1800])
    PosErr['PosPosErr']=fuzz.trimf(PosErr.universe,[0,1800,1800])



    SpeedErr['NegSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,-60,0])
    SpeedErr['ZeroSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,0,60])
    SpeedErr['PosSpeedErr']=fuzz.trimf(SpeedErr.universe,[0,60,60])


    RDMrectif['bigNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-
700,-500])
```

```
    RDMrectif['NegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-
500,0])
    RDMrectif['ZeroRDMrectif']=fuzz.trimf(RDMrectif.universe,[-
500,0,500])
    RDMrectif['PosRDMrectif']=fuzz.trimf(RDMrectif.universe,[0,500,700])

RDMrectif['bigPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[500,700,700])


    rule1 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['bigNegRDMrectif'])
    rule2 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['NegRDMrectif'])
    rule3 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule4 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['NegRDMrectif'])
    rule5 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule6 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['PosRDMrectif'])
    rule7 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule8 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['PosRDMrectif'])
    rule9 =
ctrl.Rule(PosErr['PosPosErr']&SpeedErr['PosSpeedErr'],RDMrectif['bigPosRD
Mrectif'])


perf_ctrl=ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,r
ule8,rule9])
    perf=ctrl.ControlSystemSimulation(perf_ctrl)


    pos =int(myLSS2.getPosition())
    pwm_pulse=0
    possomme=0
    prec_pwm=0


if(Setpoint-int(myLSS2.getPosition())<=0):
        SetSpeed=-30

if(Setpoint-int(myLSS2.getPosition())>0):
        SetSpeed=30

while(Setpoint-pos<-20or Setpoint-pos>20):
if(Setpoint-int(myLSS2.getPosition())<=0):
            SetSpeed=-10

if(Setpoint-int(myLSS2.getPosition())>0):
            SetSpeed=10

       perf.input['SpeedErr']=(SetSpeed-int(myLSS2.getSpeed()))
```

```
        perf.input['PosErr']=(Setpoint-int(myLSS2.getPosition()))

print("vitesse",int(myLSS2.getSpeed()))

if(abs(Setpoint-
int(myLSS2.getPosition()))<=test2(abs(int(myLSS2.getSpeed()))):
            myLSS2.moveRDM(0)
            myLSS2.hold()
print("out if1",int(myLSS2.getPosition())," i
",test2(int(myLSS2.getSpeed())))
break


        perf.compute()


print("pos2=",int(myLSS2.getPosition()))
if(abs(Setpoint-
int(myLSS2.getPosition()))<=test2(abs(int(myLSS2.getSpeed()))):
            myLSS2.moveRDM(0)
            myLSS2.hold()
print("out if2",int(myLSS2.getPosition())," i
",test2(int(myLSS2.getSpeed())))
break


        pwm_pulse=pwm_pulse+int(perf.output['RDMrectif'])
print("pwm1=",pwm_pulse)
print("speed=",int(myLSS2.getSpeed()))
if pwm_pulse>700:
                myLSS2.moveRDM(700)
                pwm_pulse=700

elif pwm_pulse<-700:
                myLSS2.moveRDM(-700)
                pwm_pulse=-700



        myLSS2.moveRDM(int(pwm_pulse))
print("pwm2=",pwm_pulse)

print("pos3=",int(myLSS2.getPosition()))
if(abs(Setpoint-
int(myLSS2.getPosition()))<=test2(abs(int(myLSS2.getSpeed()))):
            myLSS2.moveRDM(0)
            myLSS2.hold()
print("out if3",int(myLSS2.getPosition())," i
",test2(int(myLSS2.getSpeed())))
break

        possomme= possomme +abs(pos-int(myLSS2.getPosition()))

        pos =int(myLSS2.getPosition())
```

```
    myLSS2.hold()


    lss.closeBus()
return(pos)


    Setpoint=900

    PosErr= ctrl.Antecedent(np.arange(-900,900,1),'PosErr')


    SpeedErr= ctrl.Antecedent(np.arange(-60,60,1),'SpeedErr')


    RDMrectif=ctrl.Consequent(np.arange(-700,700,1),'RDMrectif')



    PosErr['NegPosErr']=fuzz.trimf(PosErr.universe,[-900,-900,0])
    PosErr['ZeroPosErr']=fuzz.trimf(PosErr.universe,[-900,0,900])
    PosErr['PosPosErr']=fuzz.trimf(PosErr.universe,[0,900,900])



    SpeedErr['NegSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,-60,0])
    SpeedErr['ZeroSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,0,60])
    SpeedErr['PosSpeedErr']=fuzz.trimf(SpeedErr.universe,[0,60,60])



    RDMrectif['bigNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-
700,-500])
    RDMrectif['NegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-700,-
500,0])
    RDMrectif['ZeroRDMrectif']=fuzz.trimf(RDMrectif.universe,[-
500,0,500])
    RDMrectif['PosRDMrectif']=fuzz.trimf(RDMrectif.universe,[0,500,700])

RDMrectif['bigPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[500,700,700])


    rule1 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['bigNegRDMrectif'])
    rule2 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['NegRDMrectif'])
    rule3 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule4 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['NegRDMrectif'])
    rule5 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule6 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['PosRDMrectif'])
```

```
    rule7 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule8 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['PosRDMrectif'])
    rule9 =
ctrl.Rule(PosErr['PosPosErr']&SpeedErr['PosSpeedErr'],RDMrectif['bigPosRD
Mrectif'])


perf_ctrl=ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,r
ule8,rule9])
    perf=ctrl.ControlSystemSimulation(perf_ctrl)


    pos =int(myLSS3.getPosition())
    pwm_pulse=0
    possomme=0
    prec_pwm=0


if(Setpoint-int(myLSS3.getPosition())<=0):
        SetSpeed=-30

if(Setpoint-int(myLSS3.getPosition())>0):
        SetSpeed=30

while(Setpoint-pos<-20or Setpoint-pos>20):
if(Setpoint-int(myLSS3.getPosition())<=0):
            SetSpeed=-10

if(Setpoint-int(myLSS3.getPosition())>0):
            SetSpeed=10



       perf.input['SpeedErr']=(SetSpeed-int(myLSS3.getSpeed()))

       perf.input['PosErr']=(Setpoint-int(myLSS3.getPosition()))

print("vitesse",int(myLSS3.getSpeed()))

if(abs(Setpoint-
int(myLSS3.getPosition()))<=test2(abs(int(myLSS3.getSpeed())))):
           myLSS3.moveRDM(0)
           myLSS3.hold()
print("out if1",int(myLSS3.getPosition()),"  i
",test2(int(myLSS3.getSpeed())))
break



       perf.compute()


print("pos2=",int(myLSS3.getPosition()))
```

```
if(abs(Setpoint-
int(myLSS3.getPosition()))<=test2(abs(int(myLSS3.getSpeed()))):
            myLSS3.moveRDM(0)
            myLSS3.hold()
print("out if2",int(myLSS3.getPosition()),"  i
",test2(int(myLSS3.getSpeed())))
break


        pwm_pulse=pwm_pulse+int(perf.output['RDMrectif'])
print("pwm1=",pwm_pulse)
print("speed=",int(myLSS3.getSpeed()))
if pwm_pulse>700:
                    myLSS3.moveRDM(700)
                    pwm_pulse=700

elif pwm_pulse<-700:
                    myLSS3.moveRDM(-700)
                    pwm_pulse=-700


        myLSS3.moveRDM(int(pwm_pulse))
print("pwm2=",pwm_pulse)

print("pos3=",int(myLSS3.getPosition()))
if(abs(Setpoint-
int(myLSS3.getPosition()))<=test2(abs(int(myLSS3.getSpeed()))):
            myLSS3.moveRDM(0)
            myLSS3.hold()
print("out if3",int(myLSS3.getPosition()),"  i
",test2(int(myLSS3.getSpeed())))
break

        possomme= possomme +abs(pos-int(myLSS3.getPosition()))

        pos =int(myLSS3.getPosition())



    myLSS3.hold()

return(pos)
    lss.closeBus()



    Setpoint=1000

    PosErr= ctrl.Antecedent(np.arange(-100,1500,1),'PosErr')


    SpeedErr= ctrl.Antecedent(np.arange(-60,60,1),'SpeedErr')


    RDMrectif=ctrl.Consequent(np.arange(-500,500,1),'RDMrectif')
```

```
    PosErr['NegPosErr']=fuzz.trimf(PosErr.universe,[-100,-100,0])
    PosErr['ZeroPosErr']=fuzz.trimf(PosErr.universe,[-100,0,1500])
    PosErr['PosPosErr']=fuzz.trimf(PosErr.universe,[0,1500,1500])



    SpeedErr['NegSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,-60,0])
    SpeedErr['ZeroSpeedErr']=fuzz.trimf(SpeedErr.universe,[-60,0,60])
    SpeedErr['PosSpeedErr']=fuzz.trimf(SpeedErr.universe,[0,60,60])


    RDMrectif['bigNegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-500,-
500,-300])
    RDMrectif['NegRDMrectif']=fuzz.trimf(RDMrectif.universe,[-500,-
300,0])
    RDMrectif['ZeroRDMrectif']=fuzz.trimf(RDMrectif.universe,[-
300,0,300])
    RDMrectif['PosRDMrectif']=fuzz.trimf(RDMrectif.universe,[0,300,500])

RDMrectif['bigPosRDMrectif']=fuzz.trimf(RDMrectif.universe,[300,500,500])


    rule1 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['bigNegRDMrectif'])
    rule2 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['NegRDMrectif'])
    rule3 = ctrl.Rule(PosErr['NegPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule4 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['NegRDMrectif'])
    rule5 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule6 = ctrl.Rule(PosErr['ZeroPosErr']&
SpeedErr['PosSpeedErr'],RDMrectif['PosRDMrectif'])
    rule7 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['NegSpeedErr'],RDMrectif['ZeroRDMrectif'])
    rule8 = ctrl.Rule(PosErr['PosPosErr']&
SpeedErr['ZeroSpeedErr'],RDMrectif['PosRDMrectif'])
    rule9 =
ctrl.Rule(PosErr['PosPosErr']&SpeedErr['PosSpeedErr'],RDMrectif['bigPosRD
Mrectif'])


perf_ctrl=ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,r
ule8,rule9])
    perf=ctrl.ControlSystemSimulation(perf_ctrl)


    pos =int(myLSS4.getPosition())
    pwm_pulse=0
    possomme=0
    prec_pwm=0
```

```
if(Setpoint-int(myLSS4.getPosition())<=0):
        SetSpeed=-30

if(Setpoint-int(myLSS4.getPosition())>0):
        SetSpeed=30

while(Setpoint-pos<-20or Setpoint-pos>20):
if(Setpoint-int(myLSS4.getPosition())<=0):
            SetSpeed=-10

if(Setpoint-int(myLSS4.getPosition())>0):
            SetSpeed=10

       perf.input['SpeedErr']=(SetSpeed-int(myLSS4.getSpeed()))

       perf.input['PosErr']=(Setpoint-int(myLSS4.getPosition()))

print("vitesse",int(myLSS4.getSpeed()))

if(abs(Setpoint-
int(myLSS4.getPosition()))<=test2(abs(int(myLSS4.getSpeed()))):
           myLSS4.moveRDM(0)
           myLSS4.hold()
print("out if1",int(myLSS4.getPosition())," i
",test2(int(myLSS4.getSpeed())))
break



       perf.compute()


print("pos2=",int(myLSS4.getPosition()))
if(abs(Setpoint-
int(myLSS4.getPosition()))<=test2(abs(int(myLSS4.getSpeed()))):
           myLSS4.moveRDM(0)
           myLSS4.hold()
print("out if2",int(myLSS4.getPosition())," i
",test2(int(myLSS4.getSpeed())))
break


       pwm_pulse=pwm_pulse+int(perf.output['RDMrectif'])
print("pwm1=",pwm_pulse)
print("speed=",int(myLSS4.getSpeed()))
if pwm_pulse>500:
                myLSS4.moveRDM(500)
                pwm_pulse=500

elif pwm_pulse<-500:
                myLSS4.moveRDM(-500)
                pwm_pulse=-500



       myLSS4.moveRDM(int(pwm_pulse))
```

```
print("pwm2=",pwm_pulse)

print("pos3=",int(myLSS4.getPosition()))
if(abs(Setpoint-
int(myLSS4.getPosition()))<=test2(abs(int(myLSS4.getSpeed()))):
            myLSS4.moveRDM(0)
            myLSS4.hold()
print("out if3",int(myLSS4.getPosition()),"  i
",test2(int(myLSS4.getSpeed())))
break

        possomme= possomme +abs(pos-int(myLSS4.getPosition()))

        pos =int(myLSS4.getPosition())



    myLSS4.hold()


    lss.closeBus()
return(pos)
```