

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement supérieur et de la recherche
scientifique.



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

ECOLE NATIONALE POLYTECHNIQUE
Département de Génie Electrique

PROJET DE FIN D'ETUDES

En vue d'obtention du diplôme
d'ingénieur d'Etat en Automatique
Promotion 2002

SUJET

Développement d'un Logiciel d'Animation et
de Commande pour Bras Manipulateurs

Proposé et dirigé par :

Mr M.TADJINE
Mr O. STIHI

Etudié par :

K. DJABELLA
B. LASSAMI

E.N.P 10, Avenue Hassen Badi, 16200 El-Harrach, Alger .

Remerciements

Le travail présenté dans ce mémoire a été mené en Laboratoire de commande des processus du département de Génie Electrique de l'Ecole Nationale Polytechnique. Il a été dirigé par monsieur M. TADJINE et monsieur O. STIHI à qui nous témoignons notre gratitude pour le savoir qu'ils nous ont prodigués.

Nous remercions monsieur D. Boukhetala, monsieur L. Abdelloual et monsieur M. S. Khelladi pour tous leurs conseils et indications qui nous ont été d'une très grande utilité.

Nous tenons également à remercier monsieur M. DJABELLA pour son aide indispensable dans la phase finale de rédaction de ce manuscrit.

Nos remerciements vont aujourd'hui à tous ceux qui ont participé à notre formation du primaire à l'université.

Enfin, nous remercions tout le personnel de la bibliothèque et tous ceux qui nous ont aidé de près ou de loin pour la réalisation de ce projet.

Karima DJABELLA & Bilel LASSAMI.

Dédicace

Je dédie ce travail avant tout, à mes parents bien aimés.
Pour l'amour qu'ils me portent, pour les sacrifices consentis à ma réussite.
Qu'ils trouvent ici le témoignage de ma reconnaissance la plus dévouée.
Que dieu leur préserve bonne santé et leur accorde longue vie.

*

A mes sœurs ; Farida, Radia, Amel, Salima, Sihem.

*

A mes grands-parents et à la mémoire de mon oncle Louenes.

*

A Mhena, qui m'a tellement encouragé durant cette année, ainsi qu'à ma nièce Imene que j'adore.

*

A mes amies et à tous ceux qui m'ont aidé, j'exprime ma profonde gratitude.

Karima DJABELLA.

Dédicace

Je dédie ce travail avant tout, à la mémoire de mon père que je n'oublierai jamais, à ma mère qui m'a tout donné et à qui je dois tout.

*

A mes frères Adel, Khaled, Nasereddine, Youcef ainsi qu'à ma sœur Amel.

*

A tout les membre de ma famille ; oncles, tantes, cousins et cousines sans oublier mon grand-père et mes grands-mères.

*

Enfin, je dédie ce travail à mes deux cousins Tasnime et Amine.

Bilel LASSAMI.

Sommaire

Introduction générale

01

Chapitre1 : Classification et Architecture des Systèmes Robotique

1.1 Introduction	03
1.2 Etude des chaînes cinématiques	03
1.2.1 Notion de structure mécanique	03
1.2.2 Classification des chaînes cinématiques	04
1.2.3 Classes des robots manipulateurs	05
1.2.3.1 Les manipulateurs sériels	05
1.2.3.2 Les manipulateurs parallèles	06
Comparaison entre les structures sérielles et parallèles	06
1.2.3.3 Les manipulateurs hybrides	07
1.3 Classification structurelle des robots manipulateurs	08
1.3.1 Robots cartésiens	09
1.3.2 Robots cylindriques	10
1.3.3 Robots à coordonnées sphériques	11
1.3.4 Robots articulés	12
1.4 Définitions et terminologies	13
1.5 Conclusion	18

Chapitre2 : Modélisation des Robots Manipulateurs

2.1 Introduction	19
2.2 Modélisation géométrique	20
2.2.1 Coordonnées homogènes d'un vecteur	20
2.2.2 Paramètres de Denavit-Hartenberg	21
2.2.3 Modèles géométriques directs	23
2.2.3.1 Modèle géométrique direct du robot Puma 560	23
2.2.3.2 Modèle géométrique direct du robot Scara	28
2.2.3.3 Modèle géométrique direct du robot hybride	30
2.2.4 Modèles géométriques inverses	33
2.2.4.1 Modèle géométrique inverse du robot Puma 560	34
2.2.4.2 Modèle géométrique inverse du robot Scara	36
2.2.4.3 Modèle géométrique inverse du robot hybride	38
2.3 Modélisation dynamique	40
2.3.1 Approche d'Euler-Lagrange	40
2.3.2 Modèle dynamique général	48
2.3.2.1 Modèle dynamique du robot Puma 560	49
2.3.2.2 Modèle dynamique du robot Scara	51
2.4 Conclusion	54

Chapitre3 : Commande Classique des Robots Manipulateurs

3.1 Introduction	
3.2 Commande classique des robots manipulateurs (couple calculé)	55
3.3 Etude de l'influence des actionneurs	55
3.3.1 Moteur à courant continu	59
3.3.2 Moteur asynchrone	60
3.4 Résultats de simulations	63
3.5 Conclusion	68
	86

Chapitre4 : Présentation du Logiciel

4.1 Introduction	
4.2 Logiciels utilisés	87
4.2.1 Builder C++	87
4.2.3 Matlab	87
4.2.3 Bibliothèques utilisées	88
4.2.3.1 La bibliothèque graphique OpenGL	89
4.2.3.2 La librairie Glut	89
4.3 Organisation du programme	89
4.3.1 Les fonctions MGD	90
4.3.2 Les fonctions MGI	90
4.3.3 La génération des trajectoires	92
4.3.3 Simulation du modèle dynamique	92
4.4 Interface développée	95
4.5 Conclusion	97
	118

Chapitre5 : Interface de Commande

5.1 Introduction	
5.2 Présentation de l'interface	119
5.3 Partie Matériel	119
5.4 Partie Programmation	120
5.4.1 Calcul de la position du moteur	122
5.4.2 La fonction « Updates »	122
5.4.3 La fonction « Trajectory Updates »	123
5.4.4 Calcul des erreurs	124
5.4.5 Calcul du rapport cyclique	128
5.4.6 Interrupteur de commande	128
5.4.7 Manipulation avec le terminal ASCII	129
5.5 Conclusion	134
	134

Conclusion Générale

Introduction Générale



Introduction Générale

Le terme « robot » vient du tchèque ; il signifie : travail forcé. Le mot, sous son acceptation moderne, fut introduit par l'auteur tchèque Karel Capek dans son ouvrage R.U.R. (Rossum's Universal Robots) qui date de 1921.

Pour le sens commun, un robot est un dispositif mécanique articulé capable d'imiter certaines fonctions humaines telles que la manipulation d'objets ou la locomotion, dans le but de substituer à l'homme pour la réalisation de certaines tâches matérielles. Cette réalisation est plus ou moins autonome selon les facultés de perception de l'environnement dont est doté le robot. La robotique est l'ensemble des activités de construction et de mise en œuvre des robots.

Un robot industriel n'est qu'un appareil de « manutention » dont la mécanique est programmable ; il peut se déplacer suivant plusieurs directions et possède un outil terminal, en extrémité de bras, appelé souvent manipulateur. Cet outil est capable d'accomplir des tâches ordinairement effectuées par l'homme.

La Division Internationale de Robotique de la société des Ingénieurs de Production (RI/SME) définit le robot industriel comme « un manipulateur multifonctions reprogrammable capable de déplacer des matériaux, des pièces, des outils ou des appareils spéciaux suivant des chemins programmés en vue d'effectuer des opérations de fabrication diversifiées ». Les mots clés de cette définition sont le qualificatif « reprogrammable », l'appellation « multifonctions » (capacité d'effectuer divers types de tâches spécialisées, ou parfois une seule) ainsi que l'action de « déplacer » des objets, ce qui définit bien la tâche du manipulateur. Pour compléter la définition de la (RI/SME), nous pouvons inclure aussi comme mot clé la notion de « mouvement tridimensionnels » (ou à plusieurs degrés de liberté), celle d'« outil terminal ».

Les systèmes robotiques se composent d'au moins trois parties essentielles :

1. Le bras manipulateur, structure mécanique mobile.
2. Les actionneurs qui mettent en mouvement les articulations du bras.
3. L'organe de commande et d'exécution des programmes spécifiant les tâches.

Le but principal de ce travail, est la conception d'une interface graphique permettant d'animer, de simuler et de commander plusieurs robots (trois plus exactement), et enfin d'établir la communication à travers le port RS 232 entre un microcontrôleur et l'ordinateur pour la commande en temps réel des bras manipulateurs.

La simulation (c'est à dire la possibilité pour un opérateur de créer et d'animer sur console informatique un robot) nécessite la connaissance des concepts généraux de la robotique. Dans ce mémoire, nous proposons une interface graphique qui permet la simulation de trois robots industriels.

Notre mémoire s'articule autour de cinq chapitres :

Chapitre 1 ; représente un ensemble de généralités qui permettront de définir différentes classes de manipulateurs. Ce chapitre est aussi l'occasion de préciser le vocabulaire utilisé en robotique.

Chapitre 2 ; destiné à donner quelques techniques de modélisation des systèmes articulés rigides et permet de donner une idée sur la complexité des modèles dynamiques des systèmes articulés.

Chapitre 3 ; nous abordons la commande du Couple Calculé et nous présentons les actionneurs choisis pour actionner les articulations des robots.

Chapitre 4 ; nous présentons notre interface et expliquons les différentes tâches qu'elle offre à un opérateur. Ce chapitre contient plusieurs démonstrations des tâches fournies dans notre application.

Chapitre 5 ; nous proposons un circuit d'interfaçage qui permet la commande d'un moteur à courant continu ainsi que la possibilité d'établir la communication entre le microcontrôleur et l'ordinateur à travers une jonction RS 232.

Chapitre 1
Classification et Architecture
des Systèmes Robotiques

1.1 Introduction :

Un système mécanique est un assemblage de solides (corps) déformables ou non ; dans le cadre de notre étude les solides sont non déformables. Chaque solide est soumis à des actions mécaniques à distance ou de contact [1]. Un mécanisme est un système mécanique particulier, dont tous les corps sont soumis à des actions mécaniques de contact.

Un robot est un mécanisme dont l'étude nécessite l'emploi d'un vocabulaire emprunté, d'une part, à la mécanique et, d'autre part, à la robotique [2]. Un robot manipulateur, ou plus simplement manipulateur, est une chaîne cinématique possédant deux corps particuliers et des actionneurs. Le premier corps particulier, appelé base, est le socle du robot ; le second, appelé terminal, est le support d'un préhenseur ou d'un outil ; le rôle du manipulateur est, en effet, de transporter ou de façonner des objets. Nous ne considérerons, par la suite, que des manipulateurs ayant leur base fixe et nous pourrions, alors, présenter et utiliser la notion de degré de mobilité. De ce fait, nous supposons que le mouvement du manipulateur est engendré par M actionneurs qui commandent les M paramètres de configuration.

Les robots manipulateurs peuvent être classés en deux catégories : ceux qui ont une structure de chaîne cinématique ouverte et ceux qui ont une structure de chaîne cinématique fermée.

Dans ce chapitre, nous rappelons, en premier lieu, les classes des mécanismes, en général, et les robots manipulateurs, en particulier. Après cela, nous procédons à la définition de quelques notions dont l'étude des robots nécessite.

1.2 Etude des chaînes cinématiques :

1.2.1 Notion de structure mécanique :

Une structure mécanique n'est autre qu'un mécanisme. Ses corps sont liés, entre eux, par l'intermédiaire de liaisons qui leur permettent d'avoir des mouvements relatifs [3].

Le nombre de degré de liberté (noté généralement ddl) d'une liaison est égal au nombre minimal de paramètres variables qui déterminent la position d'un corps par rapport au corps auquel il est lié par cette liaison. Ainsi, une liaison introduit un certain nombre de contraintes c (au plus 6) qui dépend du mode d'assemblage des éléments de la liaison. Si $c=6$, alors, la liaison est rigide, et si $c=0$, alors il n'y a aucune liaison entre les deux corps. Il y a donc une relation qui lie le nombre de contraintes c au degré de liberté d'une liaison[3] :

$$h = 6 - c \quad ; \text{ où } c \text{ est appelé aussi classe de la liaison.}$$

Pour permettre la compréhension des différents mécanismes, nous allons utiliser les conventions de représentation suivantes [4]:

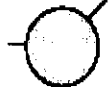
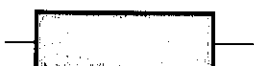
Nom	Schéma de l'articulation	
Pivot ou rotorde		non motorisée R
		motorisée <u>R</u>
Prismatique		non motorisée P
		motorisée <u>P</u>
Sphérique		non motorisée S
		motorisée <u>S</u>

Tableau 1.1 : Représentation des liaisons mécaniques selon AFNOR.

1.2.2 Classification des chaînes cinématiques :

On peut classer les mécanismes dans les deux grandes familles de chaînes cinématiques suivantes [2]:

- Les chaînes cinématiques simples dont chaque membre possède un degré de connexion (nombre de liaisons mécaniques) inférieur ou égal à deux [5].
- Les chaînes cinématiques complexes dont l'un des membres, différent de la base, possède un degré de connexion supérieur ou égal à trois [6].

Les chaînes cinématiques simples et complexes se subdivisent à leur tour en :

- Chaînes ouvertes. (voir figure 1.1.b).
- Chaînes fermées. (voir figure 1.1.a).

Une chaîne cinématique complexe fermée, où deux corps particuliers sont reliés par un ensemble de chaînes cinématiques simples et ouvertes, est appelée chaîne cinématique parallèle.

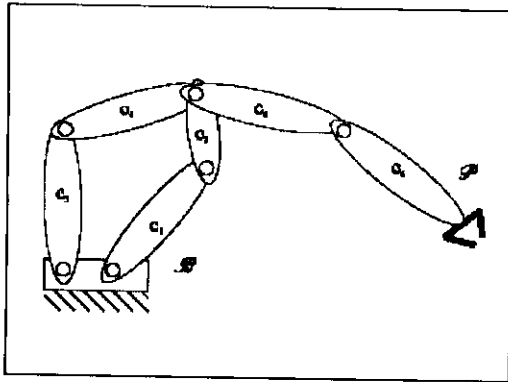


Figure 1.1.a : Manipulateur possédant une chaîne cinématique fermée.

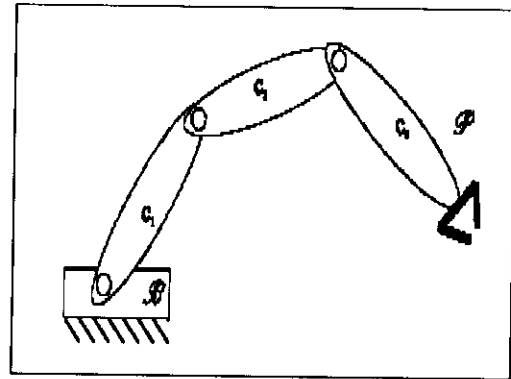


Figure 1.1.b : Chaîne cinématique ouverte.

1.2.3 Les classes des robots manipulateurs :

1.2.3.1 Les robots manipulateurs sériels :

Ce type de robot est généralement constitué d'une structure mécanique poly articulée formant entre la base du robot et l'organe terminal une seule chaîne cinématique ouverte, (Figure 1.1-a) [7]. La structure est composée de plusieurs corps rigides, liés entre eux par des articulations motorisées permettant le déplacement de deux corps consécutifs suivant un seul degré de liberté. Ce dernier peut être soit une rotation si l'articulation est rotoïde, soit une translation si l'articulation est prismatique. La modification de la position et de l'orientation de l'outil, nécessaire à la réalisation des tâches que le robot doit effectuer, est générée par la modification de la configuration de ses articulations (voir figure 1.2-a).

❖ Remarques :

- Certains robots sériels possèdent plusieurs organes terminaux, chacun étant relié à la base par une seule chaîne cinématique ouverte. Ces robots sont nommés robots sériels arborescents.
- Pour améliorer leur rigidité, certaines structures comportent une, voire plusieurs boucles cinématiques.

1.2.3.2 Les robots manipulateurs parallèles :

Ce type de robot se différencie du précédent par le fait que la base du robot et l'organe terminal ne sont plus liés par une seule chaîne cinématique mais par plusieurs (Figure 1.2- b).

Ces chaînes cinématiques sont indépendantes et reliées entre elles par un ou plusieurs segments et par un ou plusieurs actionneurs [8]. Parmi les robots parallèles, des distinctions peuvent être faites entre les robots pleinement parallèles, les robots parallèles mixtes (ou hybrides) et les robots pleinement parallèles légers [8].

❖ Comparaison entre les structures sérielles et parallèles :

Les robots sériels sont actuellement les plus utilisés dans les applications robotisées. Cela est dû aux caractéristiques avantageuses que procure leur structure morphologiquement analogue à celle du bras humain (domaine accessible relativement étendu, ...). Ces avantages font que leurs applications sont nombreuses et variées. Par contre, les inconvénients induits par leur structure sont :

- une faible rigidité, les actionneurs étant portés en série par la structure même,
- un rapport masse transportée/masse du robot relativement faible pour la plupart des robots,
- une précision peu élevée.

Les robots parallèles présentent les avantages suivants [9] :

- une rigidité plus importante et donc une meilleure précision,
- un rapport masse transportée/masse du robot plus élevé que dans le cas des robots sériels,
- une grande capacité dynamique.

Malgré les avantages cités au dessus , ces robots sont à l'heure actuelle encore peu utilisés dans l'industrie. Deux inconvénients majeurs nuisent essentiellement à leur intégration :

- un volume de travail restreint,
- une versatilité faible.

L'insertion de ces boucles permet d'augmenter la rigidité du manipulateur et d'en diminuer

❖ Remarques :

- Les structures sérielles possédant une ou plusieurs boucles cinématiques ont une meilleure rigidité et de ce fait une précision plus importante.
- Les technologies actuelles ne cessent d'améliorer les performances des nouvelles générations de robots. Par exemple, en ce qui concerne l'amélioration du rapport masse utile/masse globale, citons le robot sériel Mitsubishi PA-10 dont la masse utile est de 10kg pour une masse totale de 30 kg (ce rapport est de 2,5 kg pour 62 kg dans le cas du robot Puma 560).

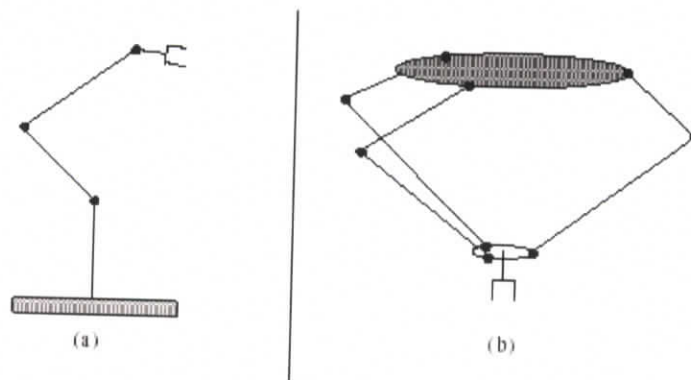


Figure 1.2 : a) Robot série, b) Robot parallèle.

1.2.3.3 Les manipulateurs hybrides :

Ce sont des manipulateurs possédant une structure sérielle dans laquelle on a inséré une ou plusieurs boucles fermées (voir figure 1.3).

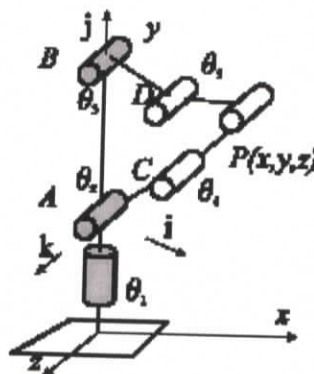


Figure 1.3 : Manipulateur hybride à trois degrés de liberté.

Nous allons maintenant passer en revue les classes de robots que nous venons de définir.

1.3.1 Robots cartésiens :

La structure d'un robot cartésien possède comme nous l'avons dit, trois degré de liberté de translation (voir figure 1.4). Autrement dit, ses déplacements s'effectuent au moyen de glissière, à la façon de la table fraiseuse. Le rapport volume utile/surface au sol d'un tel robot dépend de la rigidité de sa structure. Plus il est rigide, plus ce rapport est faible.

La mécanique du manipulateur ainsi que son logiciel s'apparentent fortement à ceux d'une machine-outil à commande numérique.

Le poignet d'un robot cartésien peut effectuer un déplacement rectiligne dans l'espace qui l'entoure par programmation de mouvements simultanés suivant chaque axe, à vitesse constante ou variable.

Supposons que l'extrémité du bras doit accomplir un cheminement rectiligne de :

$$V_x = V(x/l) \quad (1-1)$$

$$V_y = V(y/l) \quad (1-2)$$

$$V_z = V(z/l) \quad (1-3)$$

avec :

$$l = \sqrt{x^2 + y^2 + z^2} \quad (1-4)$$

x, y, z sont les composantes du mouvement l à l'effecteur. La simplicité de ces relations ne se retrouve pas dans d'autres systèmes de coordonnées que nous avons évoqués. Il faut alors développer des algorithmes complexes pour obtenir une trajectoire rectiligne. Les calculateurs de tels robots doivent en permanence transformer les coordonnées cartésiennes suivant lesquelles sont rédigés les programmes de commande, afin d'en extraire les coordonnées d'articulation correspondantes du robot.

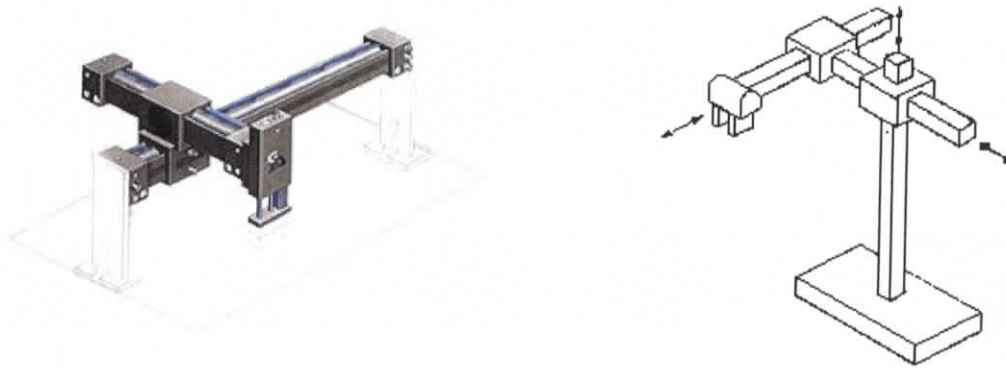


Figure 1.4 : Robot cartésien à trois axes.

Une autre caractéristique importante des robots cartésiens est leur résolution spatiale constante, quels que soient l'axe considéré et sa position dans son volume utile. Ce n'est pas le cas des autres types de robots comme nous allons les voir.

Malgré ces qualités, le robot cartésien n'est pas très répandu dans l'industrie. La raison en est qu'il manque de flexibilité. Il ne peut atteindre des objets situés au sol ou invisibles de sa base. Sa rapidité d'exécution dans le plan horizontal est généralement moindre que celle des robots qui disposent d'une base rotative.

1.3.2 Robots cylindriques :

Le bâti d'un robot cylindrique est constitué d'un bras horizontal supporté par une colonne verticale, elle-même montée sur un support rotatif. Le bras horizontal peut s'éloigner et se rapprocher de la colonne, le support de bras peut monter et descendre le long de la colonne, et l'ensemble peut effectuer des rotations par rapport au socle support. L'espace utile dans lequel travaille le robot est donc cylindrique annulaire (tore à section rectangulaire) autour de l'axe de la colonne verticale (voir figure 1.5).

La résolution d'un robot cylindrique n'est pas constante; elle dépend de la distance r entre l'axe de la colonne verticale et l'extrémité de la pince qui termine le bras horizontal. Si la résolution angulaire du bâti est α , la résolution d'arc en bout de bras est $\alpha.r$.

L'un des inconvénients des robots cylindriques par rapport aux robots cartésiens est leur mauvaise résolution qui est de l'ordre de 100 fois la résolution des robots cartésiens. Ils ont cependant pour eux l'agrément d'une vitesse appréciable de l'extrémité du bras, encore que celle-ci

soit limitée par le moment d'inertie de l'ensemble autour de l'axe de la colonne, essentiellement variable du fait de la charge manutentionnée (au bout du bras).

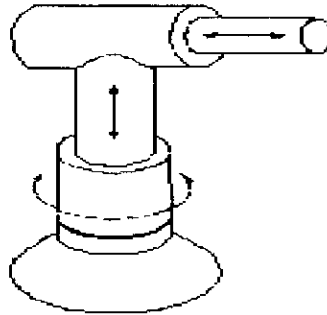


Figure 1.5 : Robot cylindrique.

Les performances dynamiques d'un robot cylindrique sont parfois difficiles à obtenir. Les efforts supportés par le moteur du bâti dépendent de la position de l'accélération des autres articulations qui opposent à ce moteur un couple résistant essentiellement variable, suivant la distance radiale de la charge manutentionnée. Le moteur qui pilote les rotations horizontales travaille donc dans de mauvaises conditions; le mouvement d'ensemble du robot y perd en souplesse et en précision.

1.3.3 Robots à coordonnées sphériques:

La cinématique de ce genre de robot qui travaille en coordonnées polaires est analogue à celle d'une tourelle de char. Le robot sphérique est constitué d'un bâti rotatif, d'une colonne verticale qui supporte un axe de rotation horizontal autour duquel peut tourner un bras à déplacement télescopique comme l'illustre la figure 1.6. Les rotations suivant les divers axes que nous venons d'énumérer sont mesurées par des codeurs à impulsions. Le volume utile dans lequel travaille le robot a la forme d'une coquille sphérique. Là encore, la résolution spatiale à l'extrémité du bras est très différente de celle que l'on peut attendre d'un robot cartésien, du fait de l'amplification en bout de bras des erreurs angulaires intervenant sur l'axes de rotation.

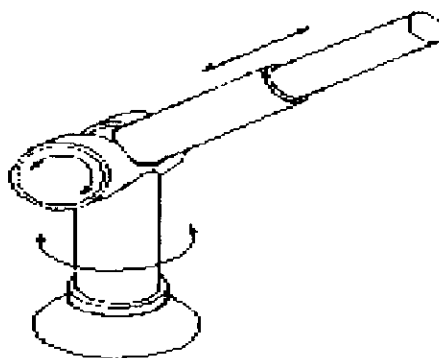


Figure 1.6 : Robot à coordonnées sphériques.

Il existe une importante disparité entre les résolutions suivant chacun des degrés de liberté d'un robot sphérique. Les mouvements de rotation d'un tel robot sont cependant plus rapides que ses mouvements de translation. L'avantage essentiel d'un robot sphérique sur ses « confrères » cartésien et cylindrique est sa flexibilité d'utilisation. Son axe horizontal lui permet d'atteindre des objets cités sur des niveaux inférieurs au plan de rotation horizontal.

1.3.4 Robots articulés:

Les robots articulés se composent d'un assemblage de trois éléments: Un bâti rotoïde horizontal qui supporte un bras à deux articulations. Cet arrangement copie tout simplement le bras humain. La pince, située en bout de bras, imite la main, attachée à l'avant bras par un poignet au robot (voir figure 1.7).

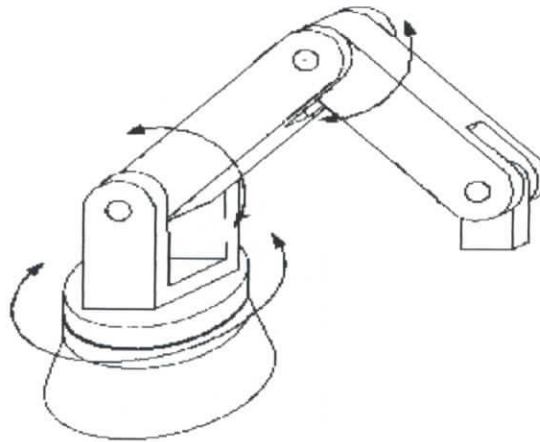


Figure 1.7 : Robot articulé.

L'épaule est souvent munie d'un degré de liberté de rotation horizontale; c'est ce que montre la figure 1.7.

Du fait que ce genre de robot possède trois axes de rotation principaux, sa résolution volumique dépend largement de sa position de travail; elle reste de toute façon mauvaise en bout de bras, puisque s'y cumulent les résolutions des trois articulations. En revanche, ce type de robot possède une excellente souplesse, ce qui en explique le succès sur les systèmes robotisés de petite et moyenne importance.

1.4 Définitions et terminologies :

Notion de degré de mobilité d'une chaîne cinématique :

Le degré de mobilité d'une chaîne cinématique est le nombre de paramètres indépendants (variables) qui déterminent la géométrie (configuration) de la chaîne. Ces paramètres sont appelés coordonnées généralisées.

Notion de configuration et de domaine de travail généralisé [2],[3] :

La configuration du manipulateur q est définie par l'ensemble des coordonnées généralisées notées $q = (q_1, q_2, \dots, q_M)^T$.

En général la plage de variation d'une coordonnée généralisée est bornée :

$$q_i \in [q_{i_{\min}}, q_{i_{\max}}] \quad (1-5)$$

q appartient donc à l'hyper parallélépipède D_q défini par :

$$D_q = \prod_{i=1}^M [q_{i_{\min}}, q_{i_{\max}}] \quad (1-6)$$

Qui constitue le domaine de travail généralisé du manipulateur et qui est inclus dans l'espace généralisé ou espace des configurations.

Notion de situation de l'organe terminal et de domaine de travail opérationnel [2],[3] :

La notion de situation est liée à la position et l'orientation de l'organe terminal du robot manipulateur. Dorénavant, par situation de l'organe terminal, nous entendons position et orientation du repère affine qui lui est lié par rapport au repère affine à la base. Cette situation est définie par m coordonnées opérationnelles notées $x = (x_1, x_2, \dots, x_m)^T$.

Dans l'espace, il est nécessaire d'utiliser 6 coordonnées indépendantes pour spécifier la situation de l'organe terminal : 3 de position et 3 d'orientation. Mais l'orientation peut être définie par plus de 3 coordonnées dépendantes [3].

X appartient au domaine de travail opérationnel D_x du manipulateur et est inclus dans l'espace opérationnel.

Les différents modèles d'un robot manipulateur :

Un robot manipulateur, quelle que soit la fonction qui lui est attribuée, doit positionner et orienter son organe terminal dans l'espace opérationnel, car le concepteur perçoit de façon naturelle, par la vision, la tâche à accomplir dans cet espace [3]. La difficulté de la commande provient du fait que les actionneurs, dont est doté le robot manipulateur, n'agissent pas directement sur la situation de l'organe terminal, mais agissent sur la configuration du robot manipulateur. D'où la nécessité de définir les passages entre les espaces opérationnels et généralisés.

Ainsi nous pouvons distinguer les modèles suivants :

- Le modèle géométrique direct, qui consiste à calculer les coordonnées opérationnelles x , en fonction des coordonnées généralisées q ; ce modèle est noté : $x = f(q)$.
- Le modèle géométrique inverse, qui consiste à calculer les coordonnées généralisées q , en fonction des coordonnées opérationnelles x ; ce modèle est noté : $q = g(x)$.
- Le modèle différentiel direct, qui consiste à calculer les différentielles des coordonnées opérationnelles dx , en fonction des différentielles des coordonnées généralisées dq ; ce modèle est noté : $dx = J dq$ (J n'est autre qu'une matrice jacobienne, on l'appelle matrice jacobienne du robot manipulateur).
- Le modèle différentiel inverse, qui consiste à calculer les différentielles des coordonnées généralisées dq , en fonction des différentielles des coordonnées opérationnelles dx ; ce modèle est noté : $dq = K dx$ (K n'est autre qu'une matrice jacobienne, on l'appelle matrice jacobienne inverse du robot manipulateur).
- Les modèles dynamiques direct et inverse, qui relient les coordonnées, vitesses et accélérations généralisées aux efforts généralisés (ces efforts généralisés sont les forces appliquées au niveau des liaisons prismatiques ou les couples au niveau des liaisons rotoides). Ces modèles tiennent compte des efforts extérieurs appliqués sur l'organe terminal.

❖ Remarques :

- Dans le cas d'un manipulateur non redondant J et K sont carrées et $K = J^{-1}$ (1-7)
- Les termes des matrices J et K peuvent être fonction soit des composantes de x, soit des composantes de q, soit de la combinaison des deux. Le calcul de l'une des deux matrices, peut être plus facile que l'autre ; cela dépend de la structure de la chaîne.

Lien entre les domaines de travail généralisé et opérationnel :

Le modèle géométrique direct permet d'écrire : $D_x = f(D_q)$. Le domaine de travail opérationnel étant, dans le cas général, impossible à visualiser, on considère souvent le domaine de travail en position D_{x_p} , défini comme l'espace physique engendré par un point de l'organe terminal, lorsque la configuration du robot évolue avec une orientation fixe de cet organe terminal, ce domaine dépend, donc, du point précédemment évoqué dont le choix est arbitraire.

En conséquence, ce domaine de travail en position D_{x_p} n'est autre que la projection du domaine de travail opérationnel D_x dans le sous-espace q qui définit la position.

Notion de degré de liberté (ddl) local et global d'un robot manipulateur [2],[3] :

Le degré de liberté local d_L , d'un robot manipulateur, est le degré local de son organe terminal, i.e. le nombre de coordonnées opérationnelles indépendantes qui permettent de fixer la situation de cet organe.

Ce nombre dépend de la configuration du robot manipulateur. Le degré de liberté (global) D, d'un robot manipulateur, est la valeur maximale de d_L pour toutes les configurations que peut prendre le robot manipulateur.

Les inégalités suivantes sont toujours vérifiées :

$$d_L \leq D \leq M \quad (1-8)$$

Ces inégalités s'expliquent par le fait que, pour pouvoir commander un organe terminal possédant D ddl, il faut au moins autant d'actionneurs.

Notion de redondance [2],[3] :

Si $D < M$, le robot manipulateur est dit redondant. On associe, alors, un ordre de redondance au robot manipulateur qui est égal à $D-M$.

Notion de configuration singulière ou singularité :

Les configurations singulières ou singularités se traduisent, physiquement, par un comportement anormal du mécanisme et, analytiquement, par la nullité du déterminant de la matrice jacobienne j ou de matrice jacobienne inverse K . Les singularités sont, relativement, bien abordées, dans le cas des chaînes cinématiques simples et ouvertes ; elles le sont beaucoup moins dans le cas des chaînes cinématiques complexes fermées. Nous exposons les principaux résultats obtenus dans le premier cas (robots sériels) et dans un cas particulier du second, à savoir les chaînes parallèles.

Singularités des chaînes cinématiques simples et ouvertes :

Les configurations singulières sont des configurations pour lesquelles, le degré de liberté local d_L est inférieur au degré de liberté global D :

$$d_L < D \quad (1-9)$$

Les configurations correspondantes sont, alors, qualifiées de singulières d'ordre $D - d_L$.

Ces singularités sont mises en évidence à partir d'une analyse différentielle basée sur le calcul du rang de la jacobienne j (C.Gosselin [5] qualifie ces singularités de type 1). En effet, nous avons :

$$dx = J.dq \quad (1-10)$$

dans le cas où : $\det(J) = 0$.

alors : $\ker(J) \neq \emptyset$.

ce qui fait que : $\exists dq \neq 0$ tel que $dx = 0$.

Ce qui veut dire, que l'on peut trouver une variation infinitésimale, des coordonnées généralisées, qui ne conduit aucune variation de la situation de la pince.

Singularités des chaînes cinématiques parallèles :

Dans le cas des chaînes cinématiques parallèles, nous dénombrons deux familles de singularités [2], [6].

- **Les singularités locales :**

Les singularités locales sont mises en évidence, à partir d'une analyse différentielle basée sur le calcul des matrices : jacobienne inverse K [2],[6],[11],[12], cinématique inverses C^{-1} [13], ou des coordonnées de plücker [14].

Il y a singularité, si les matrices K , C^{-1} , ou des coordonnées de plücker ne sont pas de rang maximal. En effet, on a, par exemple, si on utilise la matrice jacobienne inverse (ces singularités sont qualifiées, par C.Gosselin [5] de type 2) :

$$dq = K \cdot dx \tag{1-11}$$

dans le cas où K n'est pas de rang plein, alors :

$$Ker(K) \neq \emptyset.$$

$$\Rightarrow \exists dx \neq 0 \quad \text{tel que} \quad dq = 0.$$

Ce qui veut dire que la situation de l'organe terminal subit des mouvements infinitésimaux sans que les coordonnées généralisées ne varient.

- **Les singularités structurelles :**

Il s'agit de singularités qui apparaissent lors du calcul d'un modèle géométrique. Le calcul ne peut être à terme par la résolution générale. Ce phénomène se produit en général pour des mécanismes ayant des propriétés géométriques particulières. Ces singularités ont été mises en

évidence, pour la première fois, par R.Bricard en 1897 [15] lors de ses travaux sur l'octaèdre articulé. Notons que Gosselin [5] qualifie ces singularités de type 3, dans le cas d'un manipulateur non redondant.

Ces singularités se traduisent, physiquement, par une grande déformabilité, i.e. une déformabilité finie, du robot parallèle.

Quant aux chaînes cinématiques complexes fermées, plus générales, comme elles sont formées de chaînes cinématiques simples et de chaînes parallèles, nous pensons que nous allons retrouver les singularités propres aux chaînes simples et celles propres aux chaînes parallèles. Elles sont, donc, mises en évidence par les deux matrices jacobienues J et K . Mais, la difficulté est qu'il n'est pas toujours possible d'exprimer, simplement et d'une manière unique, une transformation entre les coordonnées généralisées et les coordonnées opérationnelles, quand la chaîne est complexe et fermée. Par conséquent, il est difficile de calculer les matrices jacobienues.

1.5 Conclusion :

Dans ce chapitre, nous avons rappelé les différentes notions nécessaires pour l'étude des mécanismes, en général, et des robots manipulateurs, en particulier.

Après avoir défini les mécanismes, nous les avons classés en deux familles : les mécanismes à structure de chaîne simple et les mécanismes à structure de chaîne complexe. Chaque famille se subdivise en deux : les mécanismes ouverts ou fermés. En outre, nous avons illustré les différentes structures de bras manipulateurs : cartésienne, cylindrique, sphérique et articulée.

Pour finir, nous avons précisé la définition de certains termes qui sont nécessaires, pour l'étude des robots manipulateurs.

Chapitre 2
Modélisation des Robots
Manipulateurs

2.1 Introduction :

Pour effectuer l'analyse et la synthèse dynamique, il est nécessaire de connaître les relations qui existent entre les grandeurs d'entrées et les grandeurs de sorties. L'ensemble de ces relations constitue le modèle mathématique du système. L'efficacité de ce modèle repose sur une analogie entre le comportement des objets physiques et celui des formules mathématiques.

Dans l'automatique, modéliser un système consiste à établir un ensemble de relations mathématiques qui permettent de décrire avec une précision suffisante les interactions entre ce système et son environnement extérieur. Lorsque les relations sus citées sont issues des équations de la physique, le modèle obtenu est dit modèle de connaissances, ces relations découlent des observations disponibles sur le système. On aboutit aussi au modèle de représentation, en passant par l'identification.

Dans le domaine de la robotique, l'élaboration du modèle nécessite une étude détaillée et approfondie de la structure du robot. Dans la littérature il existe plusieurs méthodes pour la modélisation des mécanismes à structures de chaînes simples ou complexes, ouvertes ou fermées [9].

Les plus utilisées sont les méthodes de Denavit-Hartenberg et Sheth-Uicker, la première est très bien adaptée pour les mécanismes à structures de chaînes simples où toutes les liaisons sont élémentaires. Mais, elle présente des difficultés lorsqu'il s'agit de mécanismes à structures de chaînes complexes ; en effet, les corps, possédant plusieurs liaisons élémentaires (rotoïdes et/ou prismatiques) en aval, doivent être dotés d'autant de repères, ce qui entraîne des lourdeurs. La deuxième méthode celle de Sheth-Uicker, vient palier les inconvénients cités précédemment, mais elle présente des redondances pour les mécanismes à structures de chaînes simples.

L'utilisation de la transformée de Denavit-Hartenberg (**D-H**) facilite la description géométrique du manipulateur, cette dernière nous permet d'aboutir au modèle cinématique et géométrique direct et inverse du robot. La même transformation offre une souplesse dans le calcul du modèle dynamique direct en utilisant le formalisme d'Euler-Lagrange.

Dans ce chapitre, nous allons introduire la notion de transformée de Denavit-Hartenberg ainsi que le formalisme d'Euler-Lagrange, afin de les utiliser pour le calcul des différents modèles de représentation des robots (Puma 560, Scara et le robot hybride) ; nous commençons par la modélisation géométrique, après l'obtention du modèle direct, nous présentons une méthode analytique pour le calcul des modèles inverses pour chaque robot. Puis, nous passons à la modélisation dynamique, nous présenterons dans un premier temps l'aspect théorique de cette modélisation et par la suite l'application de cette approche sur les deux robots ; Puma560 et Scara.

2.2 Modélisation géométrique :

Tout manipulateur peut être considéré comme une chaîne de liaisons connectées par des articulations charnières ou glissières. Chaque liaison localise les informations à son propre repère. A l'aide des matrices de passages d'ordre quatre nous arrivons à des informations globales sur le repère de la base appelé repère d'inertie.

2.2.1 Coordonnées homogènes d'un vecteur [16], [17],[18]:

Chaque liaison d'un manipulateur fait des rotations ou des translations par rapport au référentiel d'inertie fixe, par exemple un repère fixé à la base du robot. Le calcul des coordonnées des liaisons du manipulateur exprimées dans le référentiel d'inertie de la base est relativement difficile. Cette difficulté augmente suivant l'ordre de la liaison jusqu'à l'élément terminal. Pour ne pas alourdir les calculs et ramener toutes les informations géométriques au repère d'inertie de la base, il est judicieux de les localiser à leurs articulations correspondantes, et situer chaque liaison à son propre référentiel. Le passage d'un référentiel à un autre est garanti par des matrices de transformations. Lorsqu'on a uniquement des rotations on se satisfait d'une matrice de transformation R de troisième ordre, et lorsqu'il existe une translation autour d'un point on est obligé de passer vers une matrice de quatrième ordre pour permettre au référentiel de se déplacer à un autre référentiel en translation. Dans ce cas le vecteur de position p sera augmenté par une quatrième composante pour avoir un vecteur de position p' exprimé par ses coordonnées homogènes.

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (2-1)$$

Le vecteur homogène correspondant est :

$$p' = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad (2-2)$$

La matrice augmentée de transformation a la forme suivante :

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-3)$$

avec r_{ij} les composants de la matrice de rotation R , et d_x, d_y et d_z sont les composants du vecteur de translation d qui comporte les coordonnées du repère destination dans le repère source.

Si on appelle T la matrice de transformation du référentiel (x_1, y_1, z_1) vers le référentiel (x_2, y_2, z_2) alors :

$$p'_{x_2 y_2 z_2} = T \cdot p'_{x_1 y_1 z_1} \quad (2-4)$$

2.2.2 Paramètres de Denavit et Hartenberg [16], [17],[18],[19]:

La convention de Denavit et Hartenberg est une méthode systématique. Elle permet le passage entre articulations adjacentes d'un système robotique. Elle concerne les chaînes cinématiques ouvertes où l'articulation possède uniquement un degré de liberté, les surfaces adjacentes restent en contact. Pour cet aspect l'utilisation des charnières ou des glissières est indispensable. Le choix adéquat des repères dans les liaisons facilite le calcul des matrices homogènes de Denavit et Hartenberg et permet d'arriver à exprimer rapidement des informations de l'élément terminal vers la base ou l'inverse.

Les étapes à suivre pour cette technique sont les suivantes :

1. Numérotation des segments constitutifs du bras manipulateur de la base vers l'élément terminal. On associe le référentiel zéros à la base de celui-ci, et l'ordre n à l'élément terminal (effecteur) ;
2. Définition des axes principaux de chaque segment :
 - Si z_i et z_{i-1} ne se coupent pas on choisit x_i de manière à être la parallèle avec l'axe perpendiculaire à z_i et z_{i-1} .
 - Si z_i et z_{i-1} sont colinéaires on choisit x_i dans le plan perpendiculaire à z_{i-1} .

3. Fixer les quatre paramètres géométriques : d_i, θ_i, a_i et α_i (voir la figure 2.1) pour chaque articulation tel que :
- d_i cordonnées de l'origine o_i sur l'axe z_{i-1} . Pour une glissière d_i est une variable et pour une charnière d_i est une constante.
 - θ_i est l'angle que l'on obtient par vissage de x_{i-1} vers x_i autour de l'axe z_{i-1} . Pour une glissière θ_i est une constante et pour une charnière θ_i est une variable.
 - a_i est la distance entre les axes z_i et z_{i-1} mesurée sur l'axe x_i négatif à partir de son origine jusqu'à l'intersection avec l'axe z_{i-1} .
 - α_i est l'angle entre l'axe z_i et z_{i-1} obtenu en vissant z_{i-1} vers z_i autour de x_i .
4. On forme enfin la matrice homogène de Denavit et Hartenberg du déplacement qui lie la rotation et la translation. La partie supérieure gauche définit la matrice de rotation R_{i-1}^i , et le vecteur de translation à droite d_{i-1}^i .

Par la suite on aboutit à la matrice de transformation de Denavit et Hartenberg :

$$T_{i-1}^i = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-5)$$

La figure 2.1 représente les paramètres de la convention de Denavit et Hartenberg :

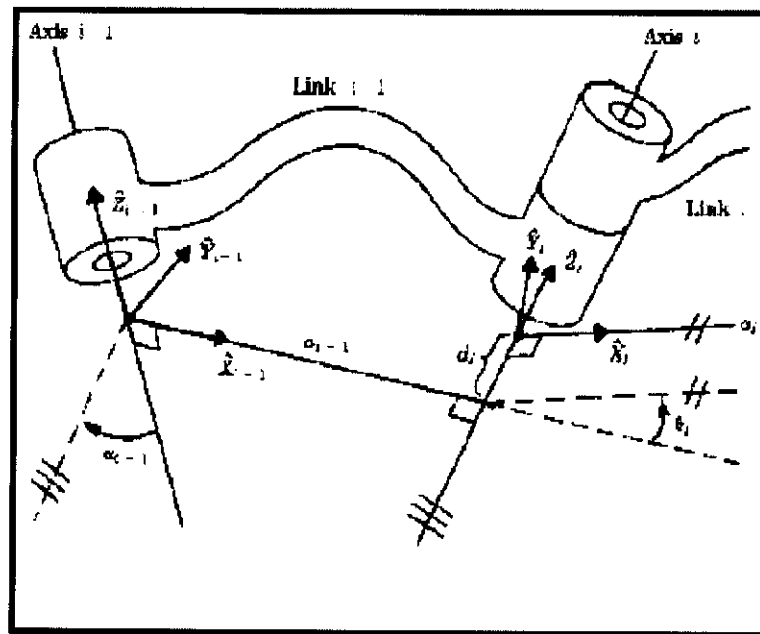


Figure 2.1: Système de coordonnées et les paramètres de Denavit et Hartenberg.

Dans ce qui va suivre, on va admettre quelques hypothèses dans le but de simplifier la modélisation des robots. Ces hypothèses sont les suivantes :

- Les liaisons du manipulateur sont rigides.
- Les jeux dans les articulations sont négligeables.
- Les capteurs ont un gain unitaire et de dynamique négligeable.

2.2.3 Modèles géométriques directs :

2.2.3.1 Modèle géométrique direct du robot Puma 560 [16], [17],[18], [20],[21]:

Le bras de robot manipulateur Puma 560 réalise six degrés de liberté caractérisés par des mouvements de rotations ; Les trois premières articulations de ce manipulateur [Waist , Shoulder, Elbow] caractérisent pour la première une rotation autour d'un axe vertical, la seconde et la troisième suivant deux axes horizontaux dont les mouvements sont identifiés par les variables q_1 , q_2 et q_3 . Les trois dernières articulations [Wrist rotation ,Wrist bend ,Flange] qui constituent le poignet du bras de robot sont caractérisées par les variables de rotation q_4 , q_5 et q_6 .

La représentation du bras de robot Puma 560 suivant la convention de Denavit et Hartenberg est présentée par la figure 2.2.

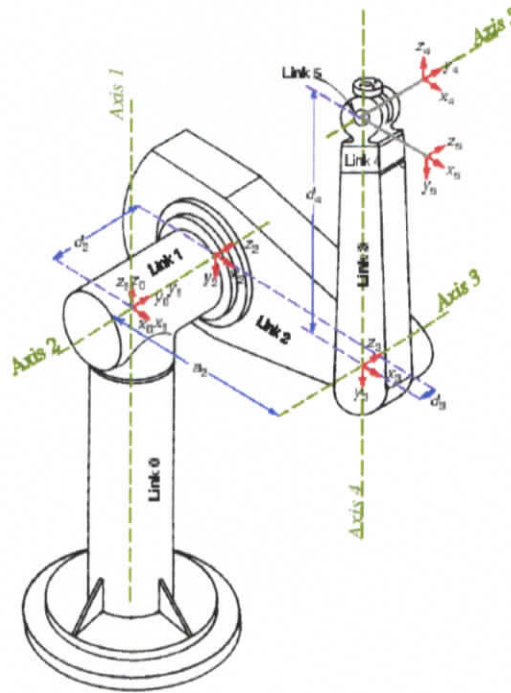


Figure 2.2: Système de coordonnées des liaisons et les paramètres des articulations pour le bras de robot Puma 560 à la position d'origine.

La définition des repères associés aux segments suivant la convention de Denavit et Hartenberg est la suivante [20],[17]:

Liaison 1 : Repère (x_0, y_0, z_0) .

L'origine est prise dans la liaison 1 à l'intersection de l'axe de la base avec l'axe de la liaison 1.

z_0 axe de rotation, $+z_0$ vers le haut.

$+y_0$ coïncide avec l'axe de la liaison 1 et l'axe $+z_1$.

$+x_0$ est en parallèle avec la liaison 2.

Liaison 2 : Repère (x_1, y_1, z_1) .

L'origine coïncide avec l'origine du repère (x_0, y_0, z_0) .

z_1 axe de rotation, $+z_1$ est perpendiculaire à la liaison 2 et en parallèle à l'axe $+z_2$.

$+y_1$ vers le bas, superposé avec l'axe de la base et en parallèle avec $+y_2$.

$+x_1$ est en parallèle à la liaison 2.

Liaison 3 : Repère (x_2, y_2, z_2).

L'origine est prise dans la liaison 2 à l'intersection de l'axe de la liaison 2 avec l'axe de l'articulation 3.

z_2 axe de rotation, $+z_2$ est perpendiculaire à la liaison 2 et l'axe $+z_3$.

$+y_2$ vers le bas, opposé avec $+z_3$.

$+x_2$ est en parallèle avec la liaison 2.

Liaison 4 : Repère (x_3, y_3, z_3).

L'origine est prise dans la liaison 3.

z_3 axe de rotation, $+z_3$ vers le poignet est perpendiculaire à $+z_4$.

$+y_3$ est perpendiculaire à la liaison 2, en parallèle avec $+z_4$.

$+x_2$ en parallèle avec la liaison 2.

Liaison 5 : Repère (x_4, y_4, z_4).

L'origine est prise dans le centre du poignet.

z_4 axe de rotation, $+z_4$ est perpendiculaire à la liaison 2 superposé avec $+y_5$.

$+y_4$ opposé à $+z_5$.

$+x_4$ en parallèle avec la liaison 2.

Liaison 6 : Repère (x_5, y_5, z_5).

L'origine coïncide avec l'origine du repère (x_4, y_4, z_4).

z_5 axe de rotation, $+z_5$ vers l'effecteur colinéaire avec $+z_6$.

$+y_5$ coïncide avec l'axe de l'articulation 5.

$+y_5$ perpendiculaire à l'axe de l'articulation 5

L'effecteur : Repère (x_6, y_6, z_6).

L'origine coïncide avec les origines des repères (x_4, y_4, z_4) et (x_5, y_5, z_5).

$+z_6$ est colinéaire avec $+z_5$.

$+y_6$ est en parallèle avec $+y_5$.

$+x_6$ est en parallèle avec $+x_5$.

Respectant la position d'origine la définition des repères des liaisons présentées par la figure 2.1, les paramètres du bras de robot Puma 560 suivant la convention de Denavit et Hartenberg sont représentés dans le tableau 2.1 [28].

i Numéro de la liaison	α_i [degrés]	θ_i variable	a_i [mètres]	d_i [mètres]
1	-90	q_1	0	0
2	0	q_2	0.4318	0.2435
3	+90	q_3	-0.0203	-0.0934
4	-90	q_4	0	0.4331
5	+90	q_5	0	0
6	0	q_6	0	-

Tableau 1.1 : Paramètres de Denavit et Hartenberg du bras de robot Puma 560.

La distance d_6 n'est pas représentée dans le tableau 1.1. Cette distance varie suivant l'effecteur utilisé pour l'application (l'effecteur est l'outil attaché au poignet sur la dernière articulation du robot pour la manipulation des objets). Dans notre application, la distance entre la fin de l'effecteur et l'axe du poignet est supposée égale à 0.05m ($d_6=5\text{cm}$).

La dynamique des trois dernières articulations est négligeable par rapport aux trois premières. Par conséquent, nous nous sommes intéressés à étudier le mouvement des trois premières articulations du bras de robot Puma 560 [Waist, Shoulder, Elbow], en fixant les autres à la position d'origine (i.e., poignet fixé à la position d'origine : $q_4 = 0$, $q_5 = 0$, $q_6 = 0$).

Les transformations appropriées en utilisant le formalisme de Denavit et Hartenberg pour les trois premières articulations sont :

$$T_0^1 = \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-6)$$

$$T_1^2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2 c_2 \\ s_2 & c_2 & 0 & a_2 s_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-7)$$

$$T_0^2 = \left[\begin{array}{ccc|c} c_1 c_2 & -c_1 s_2 & -s_1 & a_2 c_1 c_2 - d_2 s_1 \\ s_1 c_2 & -s_1 s_2 & c_1 & a_2 c_2 s_1 + d_2 c_1 \\ -s_2 & -c_2 & 0 & -a_2 s_2 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad (2-8)$$

$$T_2^3 = \left[\begin{array}{ccc|c} c_3 & 0 & s_3 & a_3 c_3 \\ s_3 & 0 & -c_3 & a_3 s_3 \\ 0 & 1 & 0 & d_3 \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad (2-9)$$

$$T_0^3 = \left[\begin{array}{ccc|c} c_1 c_{23} & -s_1 & c_1 s_{23} & c_1 (a_2 c_2 + a_3 c_{23}) - (d_2 + d_3) s_1 \\ s_1 c_{23} & c_1 & s_1 s_{23} & s_1 (a_2 c_2 + a_3 c_{23}) + (d_2 + d_3) c_1 \\ -s_{23} & 0 & c_{23} & -(a_2 s_2 + a_3 s_{23}) \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad (2-10)$$

avec $c_i = \cos \theta_i$; $s_i = \sin \theta_i$; $c_{ij} = \cos(\theta_i + \theta_j)$; $s_{ij} = \sin(\theta_i + \theta_j)$.

A l'aide du formalisme de Denavit et Hartenberg nous pouvons calculer facilement les coordonnées de l'élément terminal (fin de l'effecteur) dans le repère (x_0, y_0, z_0) fixé à la base. La notation simplificatrice r_3^6 signifie le vecteur des coordonnées homogènes de l'élément terminal exprimées dans le repère (x_3, y_3, z_3) . Avec la supposition (poignet fixé à la position d'origine $q_4 = 0$, $q_5 = 0$ et $q_6 = 0$), le vecteur r_3^6 est donné par l'équation suivante :

$$r_3^6 = \left[\begin{array}{c} 0 \\ 0 \\ d_4 + d_6 \\ 1 \end{array} \right] \quad (2-11)$$

Par la transformation de Denavit et Hartenberg T_0^3 , nous pouvons calculer les coordonnées de l'élément terminal exprimées dans le repère (x_0, y_0, z_0) par la formule suivante :

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = T_0^3 R_3^6 \quad (2-12)$$

donc :

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} c_1 c_{23} & -s_1 & c_1 s_{23} & c_1 (a_2 c_2 + a_3 c_{23}) - (d_2 + d_3) s_1 \\ s_1 c_{23} & c_1 & s_1 s_{23} & s_1 (a_2 c_2 + a_3 c_{23}) + (d_2 + d_3) c_1 \\ -s_{23} & 0 & c_{23} & -(a_2 s_2 + a_3 s_{23}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ d_4 + d_6 \\ 1 \end{bmatrix} \quad (2-13)$$

Enfin, nous définissons le vecteur position de l'élément terminal exprimé dans le repère fixé à la base (x_0, y_0, z_0) comme suit :

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} c_1 (a_2 c_2 + a_3 c_{23}) - (d_2 + d_3) s_1 + (d_4 + d_6) c_1 s_{23} \\ s_1 (a_2 c_2 + a_3 c_{23}) + (d_2 + d_3) c_1 + (d_4 + d_6) s_1 s_{23} \\ -(a_2 s_2 + a_3 s_{23}) + (d_4 + d_6) c_{23} \end{bmatrix} \quad (2-14)$$

2.2.3.2 Modèle géométrique direct du robot Scara [22]:

Le robot Scara comporte quatre degrés de liberté, les trois premiers caractérisent des rotations autour des axes Z et le quatrième représente une liaison prismatique qui caractérise une translation selon l'axe Z.

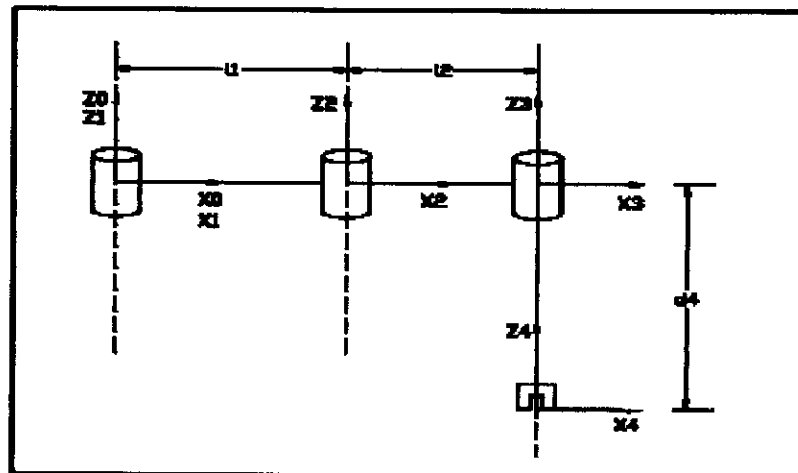


Figure 2.3 : Représentation du robot Scara.

Respectant la position d'origine la définition des repères des liaisons présentées par la figure 2.3, les paramètres du bras de robot Scara suivant la convention de Denavit-Hartenberg sont représentés dans le tableau 2.2.

i Numéro de la liaison	θ_i [variable]	a_i [mètres]	d_i [mètres]	α_i [degrés]
1	q_1	0	0	0
2	q_2	0.432	0	0
3	q_3	0.432	0	0
4	q_4	0	d_4	0

Tableau 2.2 : Paramètres de Denavit et Hartenberg du bras de robot Scara.

Les matrices de transformations Denavit-Hartenberg appropriées à ce modèle sont :

$$T_1^0 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2^1 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-15)$$

$$T_3^2 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_4^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-16)$$

avec $c_i = \cos \theta_i$; $s_i = \sin \theta_i$; $c_{ij} = \cos(\theta_i + \theta_j)$; $s_{ij} = \sin(\theta_i + \theta_j)$.

Par la transformation de Denavit et Hartenberg T_0^3 , on peut calculer les coordonnées de l'élément terminal exprimées dans le repère (x_0, y_0, z_0) par la formule suivante :

$$T_4^0 = T_1^0 T_2^1 T_3^2 T_4^3 = \begin{bmatrix} c_{123} & -s_{123} & 0 & a_2 c_1 + a_3 c_{12} \\ s_{123} & c_{123} & 0 & a_2 s_1 + a_3 s_{12} \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-17)$$

d'où

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} c_{123} & -s_{123} & 0 & a_2 c_1 + a_3 c_{12} \\ s_{123} & c_{123} & 0 & a_2 s_1 + a_3 s_{12} \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (2-18)$$

Enfin, nous définissons le vecteur position de l'élément terminal exprimé dans le repère fixé à la base (x_0, y_0, z_0) comme suit :

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} a_2 c_1 + a_3 c_{12} \\ a_2 s_1 + a_3 s_{12} \\ d_4 \\ 1 \end{bmatrix} \quad (2-19)$$

2.2.3.3 Modèle géométrique direct du robot hybride (robot parallélépipède) [23],[19]:

Le robot hybride réalise six degré de liberté caractérisés par des mouvements de rotations ; Les trois premières articulations de ce manipulateur caractérisent pour la première une rotation autour d'un axe vertical, la seconde et la troisième suivant deux axes horizontaux dont les mouvements sont identifiés par les variables q_1, q_2 et q_3 . Les trois dernières articulations qui constituent le poignet du bras de robot sont caractérisées par les variables de rotation q_4, q_5 et q_6 .

La représentation du bras de robot hybride suivant la convention de Denavit-Hartenberg est présentée par la figure 2.4.

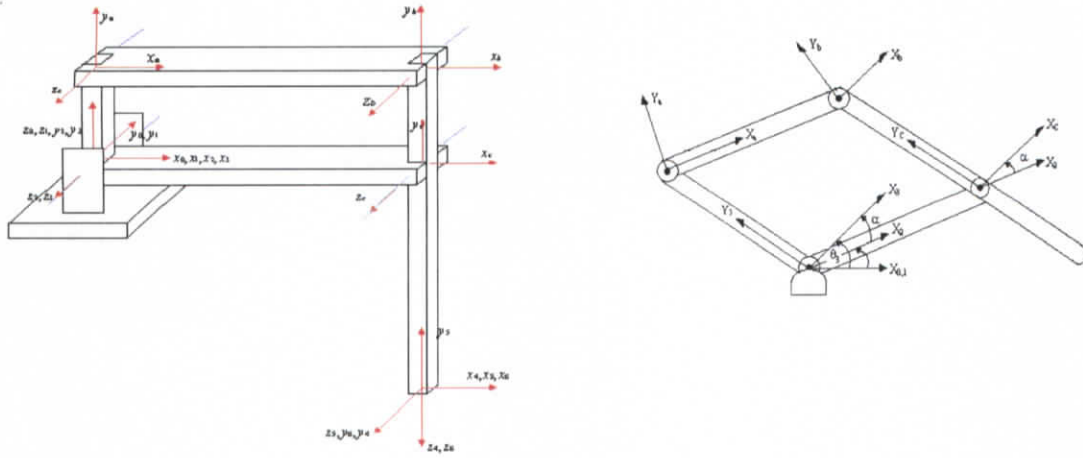


Figure 2.4 : paramètres de Denavit-Hartenberg du robot hybride.

Ce type de robot est dit « différentiel » car l'angle entre X_c et X_2 est de $\alpha = \theta_3 - \theta_2$. Le tableau suivant définit les paramètres de Denavit-Hartenberg du manipulateur [23] :

➤ **Chaîne principale :** $0 \rightarrow 1 \rightarrow 2 \rightarrow c \rightarrow 4 \rightarrow 5 \rightarrow 6$.

i Numéro de la liaison	α_i [degrés]	θ_i variable	a_i [mètres]	d_i [mètres]
1	0	q_1	0	0
2	90	q_2	0	0
c	0	q_3	l_2	$\alpha = \theta_3 - \theta_2$
4	90	q_4	0	l_4
5	-90	q_5	0	0
6	90	q_6	0	0

Tableau 2.2 : Paramètres de Denavit-Hartenberg du bras de robot hybride.

Par la suite nous déterminons les transformations homogènes dont la forme générale décrivant le passage d'un repère $i-1$ au repère i est donné ci –après (les notations que nous avons utilisées sont celles de [20]) avec $s_i = \sin \theta_i$, $c_i = \cos \theta_i$:

$$T_i^{i-1} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & d_i \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-20)$$

nous trouvons [23]:

$$T_1^0 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \quad T_2^1 = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-21)$$

$$T_c^2 = \begin{bmatrix} c\alpha & -s\alpha & 0 & l_2 \\ s\alpha & c\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \quad T_4^c = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ 0 & 0 & -1 & -l_4 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-22)$$

$$T_5^4 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_5 & -c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \quad T_6^5 = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-23)$$

➤ **Chaîne secondaire :** $0 \rightarrow 1 \rightarrow 3 \rightarrow a \rightarrow b \rightarrow c \rightarrow \dots 6$;

Celles-ci ne sont pas déterminées par la convention de Devavit-Hartenberg car les axes ne correspondent pas. Mais connaissant la forme de T_j^i contenant la matrice d'orientation des deux repères et leur position relative (formule). Nous trouvons:

$$T_3^1 = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ 0 & 0 & -1 & d \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \quad T_a^3 = \begin{bmatrix} c\alpha & s\alpha & 0 & 0 \\ -s\alpha & c\alpha & 0 & l_3 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-24)$$

$$T_b^a = \begin{bmatrix} c\alpha & -s\alpha & 0 & l_2 \\ s\alpha & c\alpha & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \quad T_a^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -l_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-25)$$

A cause de la chaîne fermée en c, les transformations que nous avons élaboré doivent vérifier une certaine contrainte mathématique due à cette contrainte mécanique [23], à savoir :

$$T_c^l = T_2^1 T_c^2 = T_3^1 T_a^3 T_b^a T_c^b \quad (2-26)$$

Ceci est vérifier, ainsi nous avons :

$$T_c^l = \begin{bmatrix} c_3 & -s_3 & 0 & l_2 c_2 \\ 0 & 0 & -1 & 0 \\ s_3 & c_3 & 0 & l_2 s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-27)$$

Nous avons aussi besoin pour la suite des calculs de l'équation suivante :

$$T_6^0 = \underbrace{T_1^0 T_2^1 T_c^2}_{T_c^0} \times \underbrace{T_4^c T_5^4 T_6^5}_{T_6^c} \quad (2-28)$$

Après des calculs intermédiaires très lourds, on obtient la matrice T_6^0 . Les composantes de la 4^{ème} colonne donnent la position du porteur (position du point P auquel sera attaché l'effecteur) exprimé dans le repère fixé à la base (x_0, y_0, z_0), soit :

$$\begin{cases} X = c_1(l_2 c_2 + l_4 s_3) \\ Y = s_1(l_2 c_2 + l_4 s_3) \\ Z = l_2 s_2 - l_4 c_3 \end{cases} \quad (2-29)$$

2.2.4 Modèles géométriques inverses [20]:

Le modèle géométrique inverse consiste à trouver l'angle de chaque articulation pour une position et orientation données de l'effecteur (l'élément terminal).

Afin de compléter l'étude géométrique des robots, nous présentons les résultats de calcul des modèles géométriques inverses pour les deux robots. Une approche analytique a été utilisée pour le robot de type série [20], elle consiste à éliminer à chaque étape une des coordonnées généralisées (articulaires) par la multiplication de la matrice de transfert finale T_6^0 par les matrices de transfert intermédiaires.

Pour le cas d'un robot à six degrés de liberté la matrice de transfert T_6^0 s'écrit sous la forme de l'équation (2-3) :

$$T_6^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_1^0(\theta_1).T_2^1(\theta_2).T_3^2(\theta_3).T_4^3(\theta_4).T_5^4(\theta_5).T_6^5(\theta_6) \quad (2-30)$$

Par exemple la première étape sera comme suit :

On multiplie de part et d'autre par la matrice $T_1^0(\theta_1)$, le résultat sera :

$$[T_1^0]^{-1}.T_6^0 = T_6^1 \quad (2-31)$$

et on s'intéresse toujours à la dernière colonne de la matrice, qui contient à chaque étape les équations découplées qui permettent de résoudre le problème du modèle géométrique inverse.

2.2.4.1 Modèle géométrique inverse du robot Puma 560 [17],[18], [20],[21]:

La portée de la charge est caractérisée par un point acentrique dans l'espace qui définit une position en coordonnées cartésiennes et donne à l'élément terminal la position et l'orientation pour manipuler les objets. La cinématique inverse est imposée. Pour cela nous pouvons citer quelques-unes des solutions qui existent :

- Méthode analytique [20],
- Transformation inverse, par Paul et Al 1981.
- Approche géométrique, par Lee et Ziegler 1984.
- Méthode des matrices duelles, par Denavit 1956.
- Quaternions duals, par Yang et Arenden Stein 1964.

Nous nous intéressons à la méthode analytique.

On a :

$$T_0^6 = T_0^1 \times T_1^6 \quad (2-32)$$

$$T_0^6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-33)$$

En utilisant les transformations de Denavit et Hartenberg déjà calculées précédemment et en faisant l'opération suivante :

$$[T_0^1]^{-1} \times T_0^6 = T_1^6 \quad (2-34)$$

on obtient le résultat suivant :

$$q_1 = A \tan 2(p_x, p_y) - A \tan 2(d_2 + d_3, \pm \sqrt{p_x^2 + p_y^2 - (d_2 + d_3)^2}) \quad (2-35)$$

et à travers des calculs lourds et complexes nous obtenons q_3 tel que :

$$q_3 = -A \tan 2(a_3, d_4 + d_6) + A \tan 2(k, \pm \sqrt{a_3^2 + (d_4 + d_6)^2 - k^2}) \quad (2-36)$$

avec

$$k = (p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2 - (d_4 + d_6)^2 - (d_2 + d_3)^2) / 2a_2 \quad (2-37)$$

de même, on a :

$$T_0^6 = T_0^3 \times T_3^6 \quad (2-38)$$

$$[T_0^3]^{-1} \times T_0^6 = T_3^6 \quad (2-39)$$

ainsi :

$$q_{23} = A \tan 2 \left((d_4 + d_6 + a_2 s_3)(c_1 p_x + s_1 p_y) - p_z (a_3 + a_2 c_3) \right. \\ \left. , (a_3 + a_2 c_3)(c_1 p_x + s_1 p_y) + p_z (d_4 + d_6 + a_2 s_3) \right) \quad (2-40)$$

et enfin,

$$q_2 = q_{23} - q_3 \quad (2-41)$$

Ainsi, nous avons obtenu les coordonnées généralisées en fonction des coordonnées cartésiennes.

2.2.4.2 Modèle géométrique inverse du robot Scara :

Par une méthode analytique [20],[24], on peut déterminer les coordonnées généralisées en fonction des coordonnées cartésiennes p_x et p_y , et en utilisant les transformations de Denavit et Hartenberg calculées précédemment, le développement sera comme suit :

On a :

$$T_4^0 = T_2^0 \times T_1^2 \quad (2-42)$$

$$[T_2^0]^{-1} \times T_4^0 = T_1^2 \quad (2-43)$$

avec

$$T_2^0 = \begin{bmatrix} c_{12} & -s_{12} & 0 & l_1 c_1 \\ s_{12} & c_{12} & 0 & l_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-44)$$

Alors

$$[T_2^0]^{-1} = \begin{bmatrix} c_{12} & s_{12} & 0 & -l_1 c_2 \\ -s_{12} & c_{12} & 0 & l_1 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-45)$$

Finalement par identification, on obtient les relations suivantes :

$$\begin{cases} \frac{p_x}{l} = c_1(c_2 + 1) - s_1 s_2 \\ \frac{p_y}{l} = s_1(c_2 + 1) - c_1 s_2 \\ p_z = d_4 \end{cases} \quad (2-46)$$

Sachant que : $l_1 = l_2 = l$.

En manipulant les dernières équations, on obtient le résultat suivant :

$$\tan q_1 = \frac{\frac{p_y}{p_x}(c_2 + 1) - s_2}{\frac{p_y}{p_x}s_2 + (c_2 + 1)} \quad (2-47)$$

d'où

$$q_1 = A \tan 2 \left(\frac{p_y(c_2 + 1) - p_x s_2}{p_x(c_2 + 1) + p_y s_2} \right) \quad (2-48)$$

et sachant le vecteur de position sur le plan OXY nous avons :

$$\rho^2 = l^2(1 + \cos q_2)^2 + (l \sin q_2)^2 = p_x^2 + p_y^2 \quad (2-49)$$

d'où, le résultat final :

$$\begin{cases} q_1 = A \tan 2 \left(\frac{p_y(c_2 + 1) - p_x s_2}{p_x(c_2 + 1) + p_y s_2} \right) \\ q_2 = \pm A \cos \left(\frac{1}{2} \times \left(\frac{\rho}{l} \right)^2 - 1 \right) \\ d_4 = p_z \end{cases} \quad (2-50)$$

Les conditions d'existence sont :

$$\begin{cases} q_2 \text{ existe si } p_x^2 + p_y^2 \leq 4l^2 \\ \text{et dans ce cas } q_1 \text{ existe toujours.} \end{cases} \quad (2-51)$$

Il est à noter que la troisième coordonnée ne rentre pas dans ce calcul car elle ne change rien au niveau de la position du robot mais son influence est au niveau de l'orientation.

2.2.4.3 Modèle géométrique inverse du robot hybride [20],[25]:

Par la méthode analytique [20], on peut déterminer les coordonnées généralisées en fonction des coordonnées cartésiennes p_x , p_y et p_z , et en utilisant les transformations de Denavit-Hartenberg calculées précédemment, le développement sera comme suit :

On a :

$$T_0^6 = T_0^1 \times T_1^6 \quad (2-52)$$

avec :

$$T_0^6 = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \quad (2-53)$$

en faisant l'opération suivante :

$$\left[T_0^1 \right]^{-1} \times T_0^6 = T_1^6 \quad (2-54)$$

et sachant que :

$$T_0^1 = T_c^1 \times T_6^c \quad (2-55)$$

On s'intéresse aux éléments de la dernière colonne de chaque matrice obtenue à travers l'équation (2-54). On obtient, après résolution de l'élément de la troisième ligne le résultat suivant :

$$q_1 = -A \tan(p_x, p_y) \pm \pi / 2 \quad (2-56)$$

On a aussi la relation suivante :

$$p_x^2 + p_y^2 + p_z^2 = l_2^2 + l_4^2 + 2l_2 l_4 \sin \alpha \quad (2-57)$$

c'est une condition nécessaire déduite du modèle géométrique direct donné par (2-29) et elle représente la condition d'existence du modèle géométrique inverse.

aussi nous avons les relations suivantes que nous avons obtenu après des calculs lourds :

$$\begin{cases} c_3(c_1 p_x + s_1 p_y) + s_3 p_z = l_2 c_2 \\ c_3 p_z + s_3(-c_1 p_x - s_1 p_y) = -l_4 - l_2 s_2 \end{cases} \quad (2-58)$$

on obtient le résultat suivant:

$$\begin{cases} s_\alpha = \frac{p_x^2 + p_y^2 + p_z^2 - l_2^2 - l_4^2}{2l_2 l_4} \\ c_\alpha = \pm \sqrt{1 - s_\alpha^2} \end{cases} \quad (2-59)$$

et par la suite on retrouve l'expression de q_3 tel que:

$$\begin{cases} c_3 = \frac{-p_z(l_4 + l_2 s_\alpha) + l_2 c_\alpha(c_1 p_x + s_1 p_y)}{(c_1 p_x + s_1 p_y)^2 + p_z^2} \\ s_3 = \frac{l_2 c_\alpha p_z + (l_4 + l_2 s_\alpha)(c_1 p_x + s_1 p_y)}{(c_1 p_x + s_1 p_y)^2 + p_z^2} \end{cases} \quad (2-60)$$

alors:

$$q_3 = A \tan(s_3, c_3) \quad (2-61)$$

et enfin:

$$q_2 = q_3 - \alpha \quad \text{avec} \quad \alpha = A \tan(s_\alpha, c_\alpha) \quad (2-62)$$

Ainsi, nous avons obtenu les coordonnées généralisées en fonction des coordonnées cartésiennes.

2.3 Modélisation dynamique [16], [17], [18],[26] :

Les modèles dynamiques des bras manipulateurs sont décrits par un ensemble d'équations mathématiques qui portent la dynamique de ceux-ci et peuvent être simulées sur ordinateur dans le but de synthétiser une commande conditionnée par des performances désirées. L'ensemble des équations dynamiques peut être déterminé par des lois mécaniques classiques Newtoniennes et Lagrangiennes. Les approches d'Euler-Lagrange et Newton-Euler permettent d'aboutir aux équations du mouvement des robots.

2.3.1 Approche d'Euler-Lagrange :

L'approche d'Euler-Lagrange sert à modéliser et présenter la dynamique des robots à travers les équations du mouvement. Elle s'adapte lors des calculs manuels ainsi que pour des calculs assistés par ordinateur. Le formalisme d'Euler-Lagrange et la transformation homogène de Denavit et Hartenberg amène à un algorithme compact pour présenter les équations dynamiques du mouvement.

L'équation d'Euler-Lagrange est :

$$\frac{d}{dt} \left[\frac{\partial L}{\partial \dot{q}_i} \right] - \frac{\partial L}{\partial q_i} + \frac{\partial E_p}{\partial q_i} = T_i \quad i=1, \dots, n \quad (2-63)$$

où L est le Lagrangien qui s'exprime par :

$$L = E_c - E_p \quad (2-64)$$

- E_c : énergie cinétique totale de toutes les liaisons ;
 E_p : énergie potentielle totale de toutes les liaisons ;
 E_D : énergie de dissipation en cas de présence de frottement visqueux ;
 T_i : force généralisée a la $i^{ème}$ articulation ;
 n : nombre de degré de liberté ;
 q_i : coordonnée généralisée d'ordre i ;
 $\dot{q}_i$: dérivée de la coordonnée généralisée.

Energie cinétique :

L 'énergie cinétique est calculée par l'expression de la vitesse.

$$\left\{ \begin{array}{l} V_0^i = \frac{dr_0^i}{dt} \\ r_0^i = T_0^i r_i^i \end{array} \right. \quad (2-65)$$

Où r_i^i est la coordonnée homogène du point (i) exprimée dans le repère R_i . Comme la liaison n'est pas flexible, on a $\frac{dr_i^i}{dt} = 0$ donc :

$$V_0^i = \sum_{j=1}^i \frac{\partial T_0^i}{\partial q_j} \frac{dq_j}{dt} r_i^i \quad (2-66)$$

avec T_{i-1}^i est la matrice de transformation homogène. Sous une forme plus compacte, nous pouvons écrire :

$$V_0^i = \sum_{j=1}^i \left[U_{ij} \dot{q}_j \right] r_i^i \quad (2-67)$$

et

$$U_{ij} = \begin{cases} T_0^{j-1} Q_j T_{j-1}^i & j \leq i \\ 0 & j \geq i \end{cases} \quad (2-68)$$

pour une liaison rotative on a :

$$Q_j = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2-69)$$

et pour une liaison de translation :

$$Q_j = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2-70)$$

L'énergie cinétique de l'élément i dans la liaison i est :

$$dE_{ci} = \frac{1}{2} \text{trace}(V_i V_i^T) dm \quad (2-71)$$

en développant l'expression précédente, on obtient :

$$dE_{ci} = \frac{1}{2} \text{trace} \left(\sum_{j=1}^i \sum_{k=1}^i U_{ij} \left(r_i^i r_i^{iT} dm \right) U_{ik}^T \dot{q}_j \dot{q}_k \right) \quad (2-72)$$

L'énergie cinétique de la liaison i est :

$$E_{ci} = \frac{1}{2} \text{trace} \left(\sum_{j=1}^i \sum_{k=1}^i U_{ij} J_i U_{ik}^T \dot{q}_j \dot{q}_k \right) \quad (2-73)$$

avec :

$$J_i = \begin{bmatrix} \int x_i^2 dm & \int x_i y_i dm & \int x_i z_i dm & \int x_i dm \\ \int x_i y_i dm & \int y_i^2 dm & \int y_i z_i dm & \int y_i dm \\ \int x_i z_i dm & \int y_i z_i dm & \int z_i^2 dm & \int z_i dm \\ \int x_i dm & \int y_i dm & \int z_i dm & \int dm \end{bmatrix} \quad (2-74)$$

L'énergie cinétique des actionneurs est définie par :

$$E_{ca} = \frac{1}{2} \sum_{i=1}^n I_i \dot{q}_i^2 \quad (2-75)$$

Où I_i caractérise un moment d'inertie dans le cas d'une rotation et une masse dans le cas d'une translation de l'actionneur i . L'énergie cinétique totale sera :

$$E_c = \sum_{i=1}^n E_{ci} + E_{ca} \quad (2-76)$$

Energie potentielle :

L'énergie potentielle est décrite par :

$$E_p = \sum_{i=1}^n -m_i g^T r_0^i = - \sum_{i=1}^n m_i g^T T_0^i r_i^i \quad (2-77)$$

$$g^T = \begin{bmatrix} 0 & 0 & -|g| & 1 \end{bmatrix} \quad (2-78)$$

où g est la gravité.

Enfin, l'énergie de dissipation est donnée par :

$$E_D = \frac{1}{2} \sum_{i=1}^n f_{vi} \dot{q}_i^2 \quad (2-79)$$

où f_{vi} le coefficient de frottement visqueux ($i=1,n$). L'expression finale du Lagrangien sera :

$$L = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^i \text{trace}(U_{ij} J_i U_{ik}^T) \dot{q}_j \dot{q}_k + \sum_i m_i g^T T_0^i r_i^i \quad (2-80)$$

Appliquant la formule d'Euler-Lagrange à la fonction lagrangienne donnée ci-dessous, on trouve la force ou le couple généralisé.

$$T_i = \sum_{j=1}^n \sum_{k=1}^j \text{trace}(U_{jk} J_j U_{ji}^T) \ddot{q}_k + \sum_{j=1}^n \sum_{k=1}^j \sum_{l=1}^l \text{trace}(U_{jkl} J_j U_{ji}^T) \dot{q}_k \dot{q}_l - \sum_{j=1}^n m_i g^T U_{ji} r_i^i + f_{vi} \dot{q}_i \quad (2-81)$$

avec

$$U_{ijk} = \begin{cases} T_0^{k-1} Q_k T_{k-1}^{j-1} Q_j T_{j-1}^i & k \leq j \leq i \\ T_0^{j-1} Q_j T_{j-1}^{k-1} Q_k T_{k-1}^i & j \leq k \leq i \\ 0 & j < i < k \end{cases} \quad (2-82)$$

Méthode directe 1 :

Cette méthode est basée essentiellement sur l'équation (1.30) qui est extraite par la méthode d'Euler-Lagrange, sa forme est :

$$T_i = \sum_{j=1}^n M_{ij}(q) \dot{q}_j + \sum_{j=1}^n \sum_{k=1}^n N_{ijk}(q) \dot{q}_j \dot{q}_k + G_i(q) + H_i(\dot{q}_i) \quad , i=1, \dots, n \quad (2-83)$$

avec

$$M_{ij}(q) = \sum_{k=\max(i,j)}^n \text{trace}(U_{ki} J_k U_{ki}^T) \quad \begin{matrix} i=1, \dots, n \\ j=1, \dots, n \end{matrix} \quad (2-84)$$

$$G_i(q) = - \sum_{j=1}^n m_j g^T U_{ji} r_j^j \quad (2-85)$$

$$N_{ijk}(q) = \sum_{l=\max(i,j,k)}^n \text{trace}(U_{ljk} J_l U_{li}^T) \quad \begin{matrix} i=1, \dots, n \\ j=1, \dots, n \\ k=1, \dots, n \end{matrix} \quad (2-86)$$

$$H_i(\dot{q}_i) = f_{vi} \dot{q}_i \quad (2-87)$$

on peut réécrire l'équation dynamique sous forme matricielle qui sera :

$$T(t) = M(q)\ddot{q} + N(q, \dot{q}) + G(q) + H(\dot{q}) \quad (2-88)$$

avec $q \in \mathbb{R}^n$, $\dot{q} \in \mathbb{R}^n$, $\ddot{q} \in \mathbb{R}^n$ représente respectivement les positions, les vitesses et les accélérations articulaires et :

$M(q) \in \mathbb{R}^{n \times n}$: matrice symétrique définie positive des accélérations inertielles dans l'élément M_{ij} de cette matrice est l'inertie de la $i^{\text{ème}}$ articulation sur la $j^{\text{ème}}$ articulation et vice versa ;

$N(q, \dot{q}) \in \mathbb{R}^n$: vecteur de forces et/ou couples aux accélérations de Coriolis et centrifuge ;

$G(q) \in \mathbb{R}^n$: vecteur de forces et/ou couples dus aux forces de gravitation ;

$H(\dot{q}) \in \mathbb{R}^n$: représente les frottements visqueux ;

$T(t) \in \mathbb{R}^n$: vecteur de force et/ou couples moteurs.

Méthode directe 2 :

La méthode directe citée auparavant est efficace mais elle est longue lors de la détermination du vecteur due aux accélérations de Coriolis et centrifuge. Pour cela on a trouvé une issue pour le calcul rapide de ce dernier. Le modèle dynamique est calculé en 3 étapes :

1. calcul des éléments de la matrice d'inertie par la méthode directe 1.
2. calcul de deux matrices centrifuges et Coriolis qui sont multipliées par leurs vecteurs, la somme des vecteurs obtenue nous donne le vecteur voulu N. L'extrait de ces dernières matrices se fait par la dérivation de la matrice d'inertie (principe de la conservation d'énergie).
3. enfin, le calcul des vecteurs gravités et visqueux, par la méthode directe se basant sur les équations (1.34) et (1.36) respectivement. La forme appropriée du modèle final sera :

$$T(t) = M(q)\ddot{q} + D(q)\begin{bmatrix} \dot{q} & \dot{q} \end{bmatrix} + C(q)\begin{bmatrix} \dot{q}^2 \end{bmatrix} + G(q) + H\left(\begin{bmatrix} \dot{q} \end{bmatrix}\right) \quad (2-89)$$

avec $q \in \mathbb{R}^n$, $\dot{q} \in \mathbb{R}^n$, $\ddot{q} \in \mathbb{R}^n$ représente respectivement les positions, les vitesses et les accélérations articulaires et :

$M(q) \in \mathbb{R}^{n \times n}$: matrice symétrique définie positive des accélérations inertielles dans l'élément M_{ij} de cette matrice est l'inertie de la $i^{\text{ème}}$ articulation sur la $j^{\text{ème}}$ articulation et vice versa ;

$C(q) \in \mathbb{R}^{n \times n}$: matrice des couples centrifuges ;

$D(q) \in \mathbb{R}^{n \times \frac{n(n-1)}{2}}$: matrice des couples de Coriolis ;

$H\left(\begin{bmatrix} \dot{q} \end{bmatrix}\right) \in \mathbb{R}^n$: représente les frottements visqueux ;

$T(t) \in \mathbb{R}^n$: vecteur de force ou couples généralisés ;

$\begin{bmatrix} \dot{q} & \dot{q} \end{bmatrix} \in \mathbb{R}^{\frac{n(n-1)}{2}}$: vecteur de produit des vitesses généralisées ;

$$\begin{bmatrix} \dot{q}^2 \end{bmatrix} \in \mathbb{R}^n \quad : \text{vecteur de carrée des vitesses généralisées ;}$$

où :

$$\begin{bmatrix} \dot{q} & \dot{q} \end{bmatrix} = \begin{bmatrix} \dot{q}_1 & \dot{q}_2 & \dots & \dot{q}_1 & \dot{q}_n & \dots & \dot{q}_2 & \dot{q}_3 & \dots & \dot{q}_2 & \dot{q}_n & \dots & \dot{q}_{n-2} & \dot{q}_n & \dots & \dot{q}_{n-1} & \dot{q}_n \end{bmatrix}^T \quad (2-90)$$

$$\begin{bmatrix} \dot{q}^2 \end{bmatrix} = \begin{bmatrix} \dot{q}_1^2 & \dot{q}_2^2 & \dots & \dot{q}_n^2 \end{bmatrix}^T \quad (2-91)$$

Le calcul de la matrice D de Coriolis se base essentiellement sur la formule suivante :

$$D_{ij} = 2 \beta^{i,jk} \quad (2-92)$$

où le symbole de Christoffel définit par :

$$\beta^{i,jk} = \frac{1}{2} \left[\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i} \right] \quad (2-93)$$

Puisque la matrice d'inertie est positive, on aura les propriétés suivantes :

$$\frac{\partial M_{ij}}{\partial q_k} = \frac{\partial M_{ji}}{\partial q_k} \quad \forall i, j, k \quad (2-94)$$

$$\frac{\partial M_{ij}}{\partial q_k} = 0 \quad \text{pour} \quad i \geq k, j \geq k \quad (2-95)$$

Les éléments de la matrice centrifuge sont définis par l'équation suivante :

$$C_{ij} = \beta^{i,jj} \quad (2-96)$$

2.3.2 Modèle dynamique général :

Le modèle dynamique général d'un robot manipulateur rigide à n degrés de liberté peut être représenté par un système d'équation différentielle non linéaire de second ordre à n entrées formant le vecteur de forces ou couples généralisés T , et n sorties qui forment le vecteur position q . Les équations de ce système à n liaisons, décrites dans l'espace des coordonnées articulaires, sont données sous forme matricielle comme suit :

$$M(q)\ddot{q} + B\left(q, \dot{q}\right)\dot{q} + K\left(q, \dot{q}\right)q + G(q) + H\left(\dot{q}\right) = T_p(t) + T(t) \quad (2-97)$$

avec $q \in \mathbb{R}^n$, $\dot{q} \in \mathbb{R}^n$, $\ddot{q} \in \mathbb{R}^n$ représente respectivement les positions, les vitesses et les accélérations articulaires et :

$M(q) \in \mathbb{R}^{n \times n}$: matrice d'inertie ;

$B\left(q, \dot{q}\right)\dot{q} + K\left(q, \dot{q}\right)q \in \mathbb{R}^n$: englobe les couples dus aux forces de Coriolis et centrifuges ;

$G(q) \in \mathbb{R}^n$: vecteur de forces ou couples dus aux forces de gravitation ;

$T_p \in \mathbb{R}^n$: le vecteur de forces ou couples de perturbation externe ;

$T(t) \in \mathbb{R}^n$: vecteur de force ou couples moteurs.

Les éléments de M , B , K , G et H sont généralement des fonctions très compliquées et non linéaires par rapport aux coordonnées généralisées du manipulateur. Le précédent modèle du robot est complexe mais vérifie certaines propriétés fondamentales qui peuvent être exploitées pour l'analyse du comportement du système et le calcul de commande.

Propriété 1 :

La matrice $M(q)$ est symétrique définie positive (SDP), par conséquent tous les éléments diagonaux de cette matrice sont positifs $\{M_{ii}(q) > 0, i=1, \dots, n\}$.

Propriété 2 :

Les matrices M , B , K et les vecteurs G , H sont uniformément bornés.

Propriété 3 :

L'entrée de commande est indépendante pour chaque articulation du manipulateur.

Propriété 4 :

Le vecteur des frottements visqueux et secs $H(q)$ est caractérisé par les n éléments $\{H_i(\dot{q}_i) \mid i=1, \dots, n\}$, tel que :

$$H_i(\dot{q}_i) = f_{vi} \dot{q}_i + f_{si} \operatorname{sgn}(\dot{q}_i) \quad (2.98)$$

avec f_{vi} et f_{si} sont respectivement les coefficients des frottements visqueux et secs de la $i^{\text{ème}}$ articulation.

Les propriétés 1-4 découlent de la nature physique du robot manipulateur. La propriété 3 est due au fait que les flexibilités des articulations et des structures n'ont pas été prises en compte. Dans ce cas de figure chaque degré de liberté (DDL) est piloté par un actionneur (moteur à courant continu par exemple).

2.3.2.1 Modèle dynamique du robot Puma 560 [17], [18],[26]:

Dans cette section, le modèle dynamique du bras de robot Puma 560 est présenté, où les trois dernières articulations de ce manipulateur qui constituent le poignet [Wrist] sont fixées à la position zéro (i.e., les positions $q_4(t) = q_5(t) = q_6(t) = 0$, les vitesses $\dot{q}_4(t) = \dot{q}_5(t) = \dot{q}_6(t) = 0$ et les accélérations $\ddot{q}_4(t) = \ddot{q}_5(t) = \ddot{q}_6(t) = 0$, $\forall t \in \mathbb{R}^+$). Nous avons adopté l'abréviation des fonctions trigonométriques des coordonnées articulaires du robot en écrivant c_i, s_i, c_{ij}, s_{ij} et c_{ijk} pour signifier $\cos(q_i), \sin(q_i), \cos(q_i + q_j), \sin(q_i + q_j)$ et $\cos(q_i + q_j + q_k)$ respectivement.

Le modèle dynamique pour les trois premières articulations du bras de robot Puma 560 est le suivant :

$$M(q)\ddot{q} + N(q, \dot{q}) + G(q) = T - m_0 J^T(q) [J(q)\ddot{q} + \dot{J}(q, \dot{q})\dot{q} + g] \quad (2-99)$$

avec : $q = [q_1, q_2, q_3]^T$ le vecteur des coordonnées généralisées, m est le poids de la charge à porter, $J(q)$ la matrice Jacobienne et $g = [0 \ 0 \ 9.81]^T$.

Les expressions des éléments des matrices M , N et le vecteur G sont les suivants :

- Les éléments de la matrice d'inertie $M(q)$:

$$\begin{aligned}
 m_{11} &\approx 1.1708 - 0.1766c_{23}^2 + 1.3337c_2^2 + 0.9423c_2c_{23} \\
 m_{12} = m_{21} &\approx 0.1627s_{23} + 0.8845s_2 \\
 m_{13} = m_{31} &\approx 0.1627s_{23} \\
 m_{22} &\approx 1.6907 + 0.9423c_3 \\
 m_{23} = m_{32} &\approx 0.212 + 0.4712c_3 \\
 m_{33} &\approx 0.212
 \end{aligned} \tag{2-100}$$

- Les éléments de la matrice $N(q, \dot{q})$:

$$\begin{aligned}
 n_{11} &\approx -(2(1.3337s_2c_2 - 0.1766s_{23}c_{23}) + 0.9423(c_2s_{23} + s_2c_{23}))\dot{q}_1\dot{q}_2 \\
 &\quad - (-0.3532s_{23}c_{23} + 0.9423s_{23}c_2)\dot{q}_1\dot{q}_3 + (0.8845c_2 + 0.1627c_{23})\dot{q}_2^2 \\
 &\quad + 0.3254c_{23}\dot{q}_2\dot{q}_3 + 0.1627c_{23}\dot{q}_3^2 \\
 n_{21} &\approx (1.3337c_2s_2 - 0.1766c_{23}s_{23} + 0.4712(s_2c_{23} + c_2s_{23}))\dot{q}_2^2 \\
 &\quad - 0.9423s_3\dot{q}_2\dot{q}_3 - 0.4712s_3\dot{q}_3^2 \\
 n_{31} &\approx (-0.1766s_{23}c_{23} + 0.4712c_2c_{23})\dot{q}_1^2 + 0.4712s_3\dot{q}_2^2
 \end{aligned} \tag{2-101}$$

- Les éléments du vecteur de gravité $G(q)$:

$$\begin{aligned}
 g_1 &= 0 \\
 g_2 &= -58.202c_2 - 10.7043c_{23} \\
 g_3 &= -10.7043c_{23}
 \end{aligned} \tag{2-102}$$

La démonstration et le calcul des éléments des matrices M , B et le vecteur G du modèle dynamique de ce bras de robot manipulateur est présenté dans la référence [28].

La matrice Jacobienne $J(q)$ est obtenue par les formules (2-14). Après quelques calculs simples nous trouvons les expressions des éléments de la matrice $J(q)$:

$$\begin{aligned}
j_{11} &= -s_1(a_2c_2 + a_3c_{23}) - (d_2 + d_3)c_1 - (d_4 + d_6)s_1s_{23} \\
j_{12} &= -c_1(a_2s_2 + a_3s_{23}) + (d_4 + d_6)c_1c_{23} \\
j_{13} &= -a_3c_1s_{23} + (d_4 + d_6)c_1c_{23} \\
j_{21} &= c_1(a_2c_2 + a_3c_{23}) - (d_2 + d_3)s_1 + (d_4 + d_6)c_1s_{23} \\
j_{22} &= -s_1(a_2s_2 + a_3s_{23}) + (d_4 + d_6)s_1c_{23} \\
j_{23} &= -a_3s_1s_{23} + (d_4 + d_6)s_1c_{23} \\
j_{31} &= 0 \\
j_{32} &= -(a_2c_2 + a_3c_{23}) - (d_4 + d_6)s_{23} \\
j_{33} &= -a_3c_{23} - (d_4 + d_6)s_{23}
\end{aligned} \tag{2-103}$$

avec $a_2 = 0.4319m$, $a_3 = -0.0203m$, $d_2 = 0.2435m$, $d_3 = -0.0934m$, $d_4 = 0.4331m$ et $d_6 = 0.05m$.

Les paramètres physiques du bras de robot Puma 560 sont représentés dans l'annexe (1). Pour plus d'informations voir [28]. Le tableau (Annexe 1.1) représente les masses des différentes liaisons du manipulateur ; la masse de la première liaison n'est pas présentée dans ce tableau puisque cette liaison ne fait pas une translation par rapport à la base du robot mais seulement une rotation de l'articulation 1 autour de l'axe z_0 (énergie potentielle nulle). La mesure de cette masse n'est pas nécessaire. Les différents moments d'inerties relatives à chaque liaison en respectant les repères des articulations sont représentés par le tableau (Annexe 1.2). La notation simplificatrice I_{xx} , I_{yy} et I_{zz} signifie respectivement le moment d'inertie suivant l'axe x, y et z relative à chaque articulation, et la notation I_{moteur} le moment d'inertie du moteur. Le rapport d'engrenage et couple maximum de chaque articulation sont représentés par le tableau (Annexe 1.3).

2.3.2.2 Modèle dynamique du robot Scara [20],[27]:

Afin de contribuer à l'étude des différents types de robots, nous avons opté pour l'architecture d'un robot Scara à deux degrés de liberté. Ce bras de robot manipulateur se déplace sur le plan horizontal suivant deux mouvements de rotation identifiés par les variables q_1 et q_2 autour des axes verticaux. Les effets gravitationnels n'interviennent pas.

Avant d'établir le modèle dynamique, plusieurs hypothèses doivent être prises en considération [17]:

- Les frottements sont secs et visqueux.
- Les différentes liaisons sont rigides
- Les capteurs ont un gain unitaire et de dynamique négligeable.

Le modèle dynamique du robot est obtenu en utilisant le formalisme de Lagrange en passant par le calcul des énergies cinétiques et potentielles. Il est donné par [27] :

$$T = (M(q) + I)\ddot{q} + N(q, \dot{q}) + G(q) + H(\dot{q}) + T_{m0} \quad (2-104)$$

avec :

$$M(q) = \begin{bmatrix} (m_2 + m_1)l_1^2 + l_2^2 m_2 + 2m_2 l_1 l_2 \cos q_2 & l_2^2 m_2 + m_2 l_1 l_2 \cos q_2 \\ l_2^2 m_2 + m_2 l_1 l_2 \cos q_2 & l_2^2 m_2 \end{bmatrix} \quad (2-105)$$

$$I = \begin{bmatrix} I_{M1} & 0 \\ 0 & I_{M2} \end{bmatrix} \quad (2-106)$$

$$N(q, \dot{q}) = \begin{bmatrix} -m_2 l_1 l_2 (2\dot{q}_1 \dot{q}_2 + \dot{q}_2^2) \sin q_2 \\ m_2 l_1 l_2 \dot{q}_1^2 \sin q_2 \end{bmatrix} \quad (2-107)$$

$$H(\dot{q}) = \begin{bmatrix} f_{v1} \dot{q}_1 + f_{s1} \operatorname{sgn}(\dot{q}_1) \\ f_{v2} \dot{q}_2 + f_{s2} \operatorname{sgn}(\dot{q}_2) \end{bmatrix} \quad (2-108)$$

$$T = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix} \quad (2-109)$$

et

$$G(q) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (2-110)$$

Le vecteur des couples additifs T_{m0} représente l'effet de la charge, il est calculé par la matrice Jacobienne. Cette dernière est la dérivée du vecteur position de l'effecteur.

$$J_j(q) = \frac{\partial P}{\partial q_j} \quad (2-111)$$

Particulièrement, pour ce robot Scara, on a :

$$J(q) = \begin{bmatrix} -l_1 \sin q_1 - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \\ l_1 \cos q_1 + l_2 \cos(q_1 + q_2) & l_2 \cos(q_1 + q_2) \end{bmatrix} \quad (2-112)$$

le couple dû à la portée de la charge sera :

$$T_{m0} = m_0 J^T(q) [J(q)\ddot{q} + \dot{J}(q, \dot{q})\dot{q} + g] \quad (2-113)$$

Avec $g = [0 \quad 0]^T$.

Les valeurs numériques des différents paramètres du robot sont [17] :

$$m_1 = 15.91kg$$

$$m_2 = 11.36kg$$

$$l_1 = l_2 = 0.432m$$

$$f_{v1} = f_{v2} = 7 \times 10^{-4} N \text{ rad}^{-1}s$$

$$f_{s1} = f_{s2} = 10^{-3}$$

$$I_{M1} = I_{M2} = 10^{-4} kgm^2.$$

2.4 Conclusion :

La modélisation des robots manipulateurs est basée sur la modélisation de la structure mécanique, les actionneurs et les capteurs. Elle nous permet la détermination des relations directes et inverses entre les coordonnées généralisées et les coordonnées cartésiennes et leurs dérivées respectives.

La convention de Denavit et Hartenberg fournit une souplesse dans le calcul pour la localisation de la position de l'élément terminal. La position initiale du robot influe sur toute la modélisation. La définition des axes selon Denavit et Hartenberg influe sur le modèle dynamique parce que les moments d'inertie des différents axes changent.

Le passage de l'espace des coordonnées cartésiennes de l'élément terminal à celui des coordonnées généralisées ou articulaires présente des singularités physiques (point en dehors de l'espace de travail), ces singularités apparaissent sous forme de contraintes lors du calcul du modèle géométrique inverse.

Les formalismes de Lagrange et Euler permettent d'établir un système d'équations différentielles reliant les coordonnées généralisées aux forces et/ou couples généralisés. Ce formalisme présente plusieurs approches dont le résultat est le même, mais pour des applications différentes il est des fois nécessaire d'avoir des séparations au niveau des termes d'inerties, centrifuges, Coriolis et de gravité.

Les modèles obtenus mettent en évidence la difficulté de commander ce genre de bras manipulateurs, étant donné les fortes non-linéarités qui régissent le fonctionnement dynamique de ceux-ci.

Chapitre 3

Commande Classique des Robots Manipulateurs

3.1 Introduction :

L'étude de la dynamique des robots manipulateurs consiste à résoudre deux problèmes essentiels : le problème de la commande des manipulateurs et le problème de la simulation de cette dernière. Ces deux problèmes se basent sur la connaissance du modèle dynamique du manipulateur. Pour réaliser des exigences bien définies, il faut posséder un bon modèle dynamique qui tient compte de la plupart des dynamiques négligées dans d'autres applications moins complexes.

Dans la commande des manipulateurs [20],[19], l'étude de la dynamique des robots manipulateurs consiste à calculer le vecteur couple ou force (Γ / F) correspondant aux coordonnées généralisées $q, \dot{q}$ et $\ddot{q}$.

Par contre dans le problème dit de simulation [20],[19], l'étude consiste à calculer les coordonnées généralisées $q, \dot{q}$ et $\ddot{q}$ dues aux couples /forces représentés par le vecteur (Γ / F), ceci représente le modèle inverse de la commande des robots manipulateurs.

Dans ce présent chapitre, nous allons introduire la notion de commande classique des robots manipulateurs, ce type de commande va non seulement nous permettre de commander le robot (réaliser la poursuite de trajectoires bien définies), mais aussi, la simulation du mouvement du robot manipulateur en temps réel.

3.2 Commande classique des robots manipulateurs (méthode du couple calculé) :

Afin d'offrir à l'utilisateur, à travers notre logiciel, un espace de commande et de contrôle des robots manipulateurs, nous proposons le calcul de la commande (pour les deux robots : le robot Puma 560 et le robot Scara) pour assurer le suivi (poursuite) d'une trajectoire désirée.

Le modèle dynamique des robots manipulateurs est sous la forme [17],[20],[19]:

$$\Gamma = M(q)\ddot{q} + N(q, \dot{q}) + G(q) + H(\dot{q}) \quad (3-1)$$

où :

- Γ : vecteur de couple des actionneurs.
 $M(q)$: matrice d'inertie.
 q : vecteur de coordonnées articulaires (généralisées).
 $N(q, \dot{q})$: vecteur de couples de Coriolis et centrifuge.
 $G(q)$: vecteur de couples dus aux forces de gravitation.
 $H(q)$: vecteur de couples représentant les frottements.

Le modèle dynamique est du second ordre, multivariable (MIMO), non linéaire et interconnecté (couplé), d'où la complexité du modèle et par la suite de la commande.

La méthode proposée [19],[20],[29],[30] consiste à chercher une commande par retour d'état dite : **Commande par découplage non-linéaire**, qui permet le calcul du couple ou force motrice pour chaque articulation indépendamment des autres.

Le schéma de la commande est représenté par la figure (3.1) :

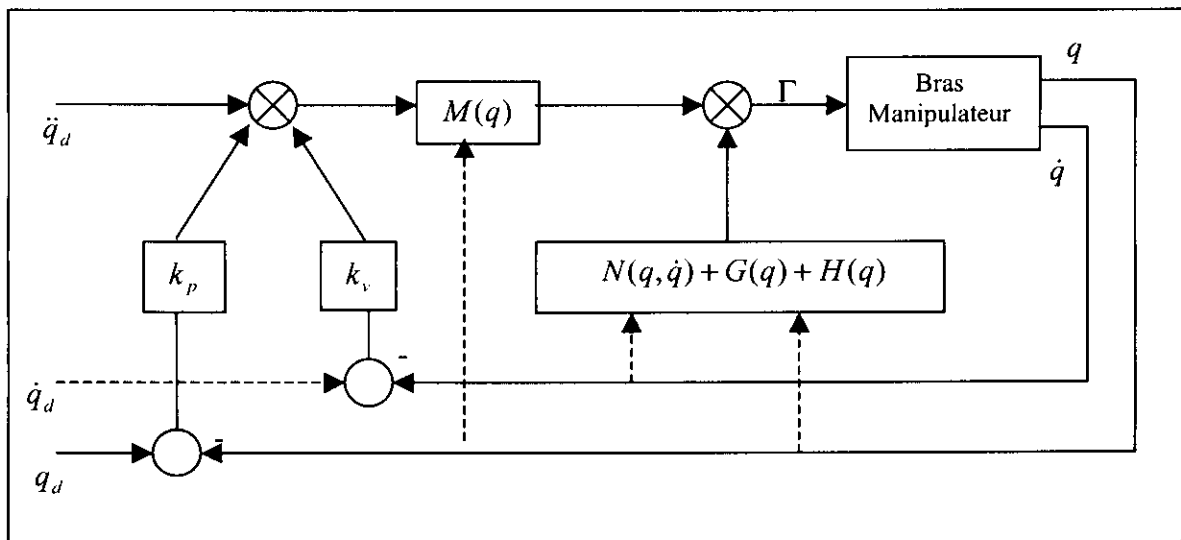


Figure 3.1 : Commande par découplage non-linéaire.

Dans ce schéma, nous constatons l'absence des actionneurs, donc c'est un schéma où la dynamique des actionneurs est supposée parfaite (la commande est égale au couple moteur). Pour faire apparaître cet effet (la dynamique des actionneurs), nous proposons le schéma suivant qui comporte deux boucles de réglage en cascade, une boucle interne pour l'actionneur et une autre externe pour le bras manipulateur.

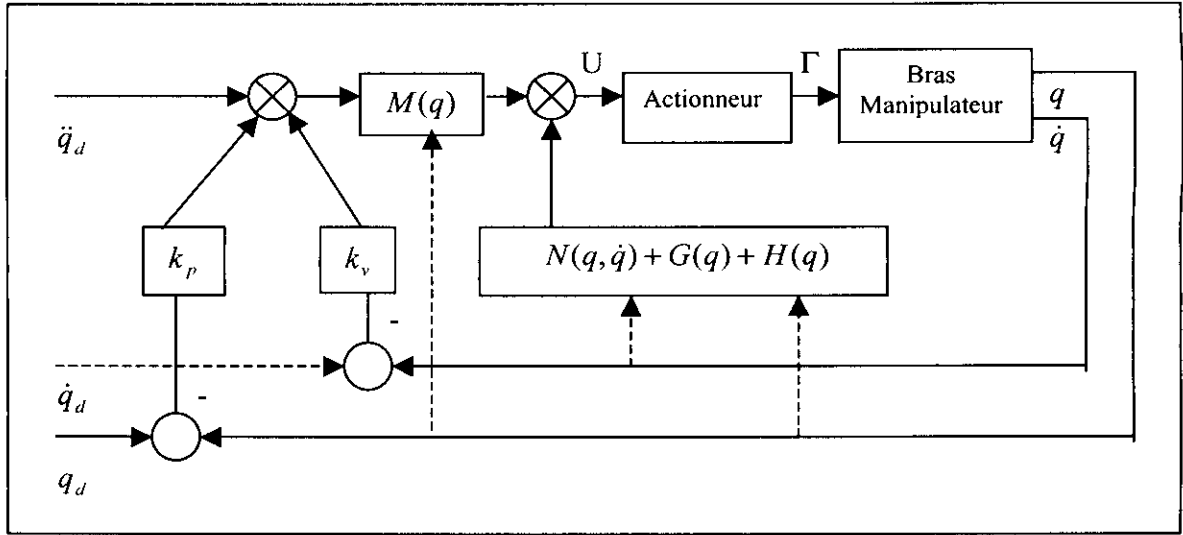


Figure 3.2 : Commande par découplage non-linéaire avec prise en compte de l'actionneur.

Pour une trajectoire désirée q_d et donc $\dot{q}_d, \ddot{q}_d$, nous proposons de générer une commande U tel que :

$$U = M^{-1}(q)[\ddot{q}_d + k_v(\dot{q}_d - \dot{q}) + k_p(q_d - q)] + N(q, \dot{q}) + G(q) + H(\dot{q}) \quad (3-2)$$

avec :

k_p : matrice diagonale représentant les constantes de proportionnalité du régulateur.

k_v : matrice diagonale représentant les constantes de dérivation du régulateur.

Si nous négligeons dans un premier temps la dynamique des actionneurs, il en résulte :

$$\Gamma = U \quad (3-3)$$

Et par conséquent :

$$M[(\ddot{q}_d - \ddot{q}) + k_v(\dot{q}_d - \dot{q}) + k_p(q_d - q)] = 0 \quad (3-4)$$

Si nous appelons $(q_d - q)$ l'erreur en position, avec : $e^T = [q_{1d} - q_1 \quad q_{2d} - q_2 \quad q_{3d} - q_3]$.

Alors :

$$M(\ddot{e} + k_v \dot{e} + k_p e) = 0 \quad (3-5)$$

et puisque M est inversible, nous obtenons :

$$(\ddot{e} + k_v \dot{e} + k_p e) = 0 \quad (3-6)$$

L'équation (3-6) représente un système d'équations différentielles du second ordre découplées (dû aux matrices k_p et k_v qui sont diagonales) et facile à résoudre. Cette dernière représente un régulateur PD, donc nous avons pu transformer le modèle de commande complexe en un modèle de commande de base pour chaque articulation indépendamment des autres.

Cette méthode est la plus utilisée dans l'industrie vu sa simplicité et ses résultats très performants[19],[20].

Nous remarquons qu'il s'agit d'une commande par retour d'état puisque les termes calculés dans l'élaboration de la consigne dépendent de l'état actuel du système dynamique [19],[31]. Dans les ouvrages anglais cette méthode est connue sous le nom : **computed torque** ou **couple calculé** qui est en fait la base de cette méthode[31],[30].

La commande présentée ici est bien entendu idéale, en réalité, il y aura des erreurs de modélisation en particulier sur M , N et H . Il conviendrait donc d'étudier la robustesse de la commande vis-à-vis des erreurs de modélisation. D'autre part, il est utile d'étudier pour chaque architecture de robot et certains mouvements types, l'influence des termes centrifuges et de Coriolis vis-à-vis des termes d'accélération [19],[20].

Généralement, pour les robots industriels, les régulateurs sont de type PID, d'où l'ajout d'un terme d'intégration de l'erreur de position angulaire [19],[20],[32],[33]: $e = q_d - q$.

Dans ce cas la commande doit être écrite sous la forme suivante :

$$U = M(q) \left[(\ddot{q}_d + k_v \dot{e} + k_p e + k_i \int e dt) \right] + N(q, \dot{q}) + G(q) + H(\dot{q}) \quad (3-7)$$

et la relation finale du régulateur devient :

$$\ddot{e} + k_v \dot{e} + k_p e + k_i \int e dt = 0 \quad (3-8)$$

Si nous posons maintenant $\varepsilon = \int e dt$, nous trouvons :

$$\ddot{\varepsilon} + k_v \dot{\varepsilon} + k_p \varepsilon + k_i \varepsilon = 0 \quad (3-9)$$

Cette dernière équation représente le réglage par action de type PID.

Dans notre logiciel, les paramètres de réglage (éléments des matrices k_v , k_p et k_i) sont laissés à l'utilisateur afin de les fixer, et voir les différentes réponses obtenues à travers la simulation telle que : l'erreur, la trajectoire suivie et la trajectoire désirée, et c'est à lui de juger de la validité ou non des résultats qu'il a obtenu.

Les trajectoires choisies dans notre application sont de type sinusoïdal et polynomial cubique [20],[19]. Nous avons opté pour une trajectoire sinusoïdale parce qu'elle est souvent rencontrée dans la littérature comme étant une trajectoire type de test, à travers laquelle, nous pouvons évaluer nos critères de performances.

Par ailleurs, la trajectoire polynomiale cubique a été choisie car elle nécessite 4 conditions initiales (ce qui correspond à notre cas, car nous avons une équation différentielle du second ordre) q_0 , $\dot{q}_0$, q_f et $\dot{q}_f$. En outre, elle est la mieux adaptée pour ce type de commande car elle permet d'exciter toutes les dynamiques du manipulateur [20],[21].

Nous notons ici, que lors de la simulation, nous avons utilisé la méthode de Runge Kutta d'ordre 4 à pas variable (en utilisant le logiciel Matlab), les résultats sont ensuite sauvegardés dans un fichier nommé data.txt avant d'être représentés sous forme de graphes en fonction du temps pour chaque articulation.

3.3 Influence des actionneurs :

Dans cette partie, nous allons étudier l'influence des actionneurs dans la boucle de commande. Afin d'élargir notre étude, nous allons étudier l'influence de deux types d'actionneurs : le moteur à courant continu et le moteur asynchrone. Il faut mentionner que le moteur à courant continu est le type d'actionneur le plus utilisé en robotique vu la simplicité de son modèle et la facilité de son réglage [10],[19],[20].

La boucle de commande interne sera représentée comme suit :

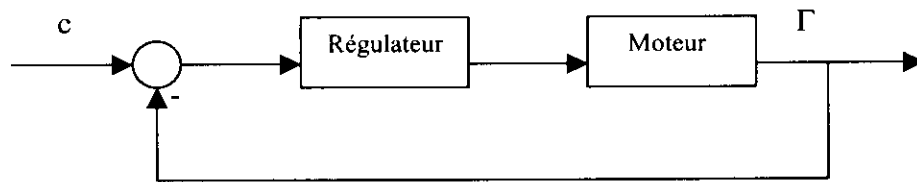


Figure 3.3 : La boucle interne du système.

Où le bloc régulateur changera selon le type d'actionneur (moteur).

3.3.1 Moteur à courant continu [34],[35]:

Ce type de moteur est régi par l'équation électrique différentielle suivante :

$$U = Ri + L \frac{di}{dt} + E = Ri + L \frac{di}{dt} + k_1 N \phi \quad (3-10)$$

avec :

- U : tension d'alimentation d'induit.
- R : résistance d'induit.
- L : inductance de l'induit.
- E : force contre-électromotrice du moteur.
- k_1 : constante.
- i : courant dans l'inducteur.
- ϕ : flux utile de l'inducteur.
- N : vitesse de rotation.

Pour des faibles vitesses :

$$E \approx 0. \quad (3-11)$$

alors l'équation devient :

$$U = Ri + L \frac{di}{dt} \quad (3-12)$$

après le passage par la transformation de Laplace :

$$U(s) = [R + Ls] i(s) \quad (3-13)$$

d'où :

$$i(s) = \frac{1}{R + Ls} U(s) \quad (3-14)$$

le couple fournit pour le moteur est donné par :

$$\Gamma = k_2 \phi i \quad (3-15)$$

tel que :

k_2 : constante du couple .

après le passage par la transformation de Laplace :

$$\Gamma(s) = k_2 \phi(s) i(s) \quad (3-16)$$

pour un flux constant :

$$\phi(s) = \phi_0 \Rightarrow \Gamma(s) = k_2 \phi_0 i(s) \quad (3-17)$$

et en remplaçant $i(s)$ par sa valeur, nous obtenons :

$$\Gamma(s) = \frac{k_2 \phi_0}{R + Ls} U(s) \quad (3-18)$$

alors :

$$\Gamma(s) = \frac{k_2 \phi_0}{R} \frac{1}{1 + \frac{L}{R} s} U(s) \quad (3-19)$$

et finalement, nous pouvons écrire :

$$\frac{\Gamma(s)}{U(s)} = \frac{K}{1 + \tau s} \quad (3-20)$$

avec :

$$K = \frac{k_2 \phi_0}{R} \quad \text{et} \quad \tau = \frac{L}{R}$$

C'est un système du premier ordre, stable. Nous allons le réguler avec un régulateur classique de type PI dont la boucle fermée sera sous la forme d'un modèle de référence :

$$\frac{\Gamma(s)}{c(s)} = \frac{\alpha}{s + \alpha} \quad (3-21)$$

Où α est un paramètre laissé à l'utilisateur, pour imposer une dynamique désirée de l'actionneur, et par la suite les paramètres du régulateur sont donnés par identification par les relations suivantes :

$$K_p = \frac{\tau \times \alpha}{K} \quad \text{et} \quad T_i = \frac{K}{\alpha} K_p \quad (3-22)$$

Nous notons ici, que le modèle de référence possède un gain statique égal à 1, pour assurer le réglage (si $t \rightarrow \infty \Rightarrow s \rightarrow 0$ et donc $\Gamma \rightarrow c$).

Ce modèle, va être utilisé pour les trois articulations. Donc le bloc actionneur de la figure (3.2) sera représenté par la matrice de fonction de transfert suivante :

$$A = \begin{bmatrix} \frac{\alpha}{s+\alpha} & 0 & 0 \\ 0 & \frac{\alpha}{s+\alpha} & 0 \\ 0 & 0 & \frac{\alpha}{s+\alpha} \end{bmatrix} \quad (3-23)$$

Dans notre application, nous nous sommes limités à l'utilisation d'un même modèle pour les trois moteurs, et par conséquent d'un même modèle de référence. C'est à dire que l'utilisateur doit seulement faire rentrer les paramètres (k , τ et α) d'un seul moteur, ces derniers seront les mêmes pour les deux autres moteurs qui restent. Ceci a été fait, dans le but d'alléger les calculs, mais aussi dans le but d'étudier l'influence de la dynamique des actionneurs en boucle fermée sur la dynamique du bras manipulateur.

3.3.2 Moteur asynchrone [36]:

Toute commande de système, entre autre, celle des machines électriques exige une modélisation approuvée et accessible pour l'automaticien. Dans le contexte fréquent d'hypothèses simplificatrices, le modèle mathématique de la machine asynchrone est un système de cinq équations non linéaires et à coefficients variables dans le temps. En conséquence, la résolution d'un tel système s'avère très complexe et le contrôle du couple séparément du flux n'est pas possible.

Pour réduire cette complexité et retrouver un système d'équations à coefficients constants, la transformation de Park est introduite (voir annexe 2). Quant au découplage, nous ferons appel à la notion de référentiel lié au champ tournant.

Dans ce qui suit, nous présentons le système d'état après chaque étape : transformation de Park, découplage, ... etc.

En tenant compte des hypothèses détaillées dans l'annexe 2, les tensions statoriques et rotoriques sont données par :

$$\begin{cases} [V_s] = [R_s] [i_s] + \frac{d[\phi_s]}{dt} \\ [V_r] = [R_r] [i_r] + \frac{d[\phi_r]}{dt} \end{cases} \quad (3-24)$$

Les expressions du flux sont :

$$\begin{cases} [\phi_s] = [l_s] [i_s] + [M_{sr}] [i_r] \\ [\phi_r] = [l_r] [i_r] + [M_{sr}]^T [i_s] \end{cases} \quad (3-25)$$

avec :

$$R_x = \begin{bmatrix} R_x & 0 & 0 \\ 0 & R_x & 0 \\ 0 & 0 & R_x \end{bmatrix}, \quad l_x = \begin{bmatrix} l_x & M_x & M_x \\ M_x & l_x & M_x \\ M_x & M_x & l_x \end{bmatrix}, \quad x=s, r. \quad (3-26)$$

et

$$[M_{sr}] = \begin{bmatrix} \cos\theta & \cos(\theta + 2\pi/3) & \cos(\theta - 2\pi/3) \\ \cos(\theta - 2\pi/3) & \cos\theta & \cos(\theta + 2\pi/3) \\ \cos(\theta + 2\pi/3) & \cos(\theta - 2\pi/3) & \cos\theta \end{bmatrix} \quad (3-27)$$

Le moteur diphasé équivalent au moteur triphasé équilibré est donné (après avoir transformé les équations précédentes en utilisant la transformation de Park) par :

$$\begin{cases} V_{dz} = R_z i_{dz} + \frac{d\phi_{dz}}{dt} - \frac{d\theta_z}{dt} \phi_{qz} \\ V_{qz} = R_z i_{qz} + \frac{d\phi_{qz}}{dt} + \frac{d\theta_z}{dt} \phi_{dz} \end{cases}, \quad z = s, r \quad (3-28)$$

$$\begin{cases} \phi_{ds} = L_s i_{ds} + M i_{dr} \\ \phi_{qs} = L_s i_{qs} + M i_{qr} \end{cases}; \quad \text{avec} \quad \begin{cases} L_s = l_s - M_s \\ M = 3M_{sr}/2 \end{cases} \quad (3-29)$$

$$\begin{cases} \phi_{dr} = L_r i_{dr} + M i_{ds} \\ \phi_{qr} = L_r i_{qr} + M i_{qs} \end{cases} ; \text{ avec } L_r = l_r + M_r \quad (3-30)$$

nous posons : $\omega_s = \frac{d\theta_s}{dt}$, $\omega_r = \frac{d\theta_r}{dt}$ et $\omega = \frac{d\theta}{dt}$, de plus : $\omega_s - \omega_r = \omega = p\Omega$

L'équation mécanique du couple est donnée par :

$$C_{em} = \frac{pM}{L_r} (\phi_{dr} i_{qs} - \phi_{qr} i_{ds}) \quad (3-31)$$

d'autre part nous disposons de l'expression :

$$C_{em} = J \frac{d(\omega/p)}{dt} + K_f (\omega/p) + C_r \quad (3-32)$$

où C_r représente le couple résistant.

Et si nous choisissons un référentiel lié au champ tournant avec orientation du flux rotorique ($\phi_{dr} = \phi_r, \phi_{qr} = 0$), et pour un vecteur d'état $x = (x_1, x_2, x_3, x_4, x_5) = (i_{ds}, i_{qs}, \phi_{dr}, \phi_{qr}, \omega)$. Le système d'état s'écrit :

$$\begin{cases} \dot{x}_1 = -a_1 x_1 + b_1 x_3 + c_1 \omega_s x_2 + d_1 x_4 x_5 + e V_{ds} \\ \dot{x}_2 = -a_1 x_2 + b_1 x_4 - c_1 \omega_s x_1 - d_1 x_3 x_5 + e V_{qs} \\ \dot{x}_3 = -a_2 x_3 + b_2 x_1 + c_1 \omega_s x_4 - c_1 x_4 x_5 \\ \dot{x}_4 = -a_2 x_4 + b_2 x_2 - c_1 \omega_s x_3 + c_1 x_3 x_5 \\ \dot{x}_5 = -a_3 x_5 + d_3 x_2 x_3 - d_3 x_1 x_4 - f \times C_r \end{cases} \quad (3-33)$$

où :

$$\begin{aligned}\omega_s &= \omega + \frac{M}{T_r} \frac{i_{qs}}{\phi_r} \\ a_1 &= (1/(\sigma \times T_s) + (1-\sigma)/(\sigma \times T_r)) \quad , \quad a_2 = 1/T_r \quad , \quad a_3 = K_f / j . \\ b_1 &= ((1-\sigma)/(\sigma \times M \times T_r)) \times (\phi_n / I_m) \quad , \quad b_2 = M \times I_m / (T_r \times \phi_n) \\ d_1 &= ((1-\sigma)/(\sigma \times M)) \times (\phi_n \times \omega_n / I_m) \quad , \quad d_3 = ((p \times p \times M)/(j \times L_r)) \times (I_m \times \phi_n / \omega_n) \\ c_1 &= \omega_n \quad , \quad T_r = L_r / R_r \\ e &= V_m / (\sigma \times L_s \times I_m) \quad , \quad f = p \times C_n / (j \times \omega_n)\end{aligned}$$

Le système est non-linéaire, il s'écrit sous la forme : $\dot{x} = f(x) + g(x)u$. Nous appliquons alors, la méthode du Backsteeping (pour le développement de cette méthode consulter l'annexe 3) pour linéariser le système. Nous choisissons alors comme sorties fictives :

$$h_1(x) = \frac{1}{2}(x_3^2 + x_4^2) = \frac{1}{2}\phi_r^2 \quad \text{et} \quad h_2(x) = x_5 = \omega \quad (3-34)$$

Le degré relatif pour chaque sortie est égal à 2, alors nous complétons le système par (pour plus d'explications voir annexe 3) :

$$z_5 = \arctg\left(\frac{x_3}{x_4}\right) \quad (3-35)$$

Le système final est le suivant :

$$\left\{ \begin{array}{l} \dot{z}_1 = z_2 \\ \dot{z}_2 = V_1 \\ \dot{z}_3 = z_4 \\ \dot{z}_4 = V_2 \\ \dot{z}_5 = \frac{x_3 \times f_4 - x_4 \times f_3}{x_3 \phi_r} \end{array} \right. \quad (3-36)$$

La dynamique de z_5 étant bornée, donc nous avons obtenu deux systèmes complètement découplés et linéaires.

Nous nous intéressons au système suivant :

$$\begin{cases} \dot{z}_3 = z_4 = \dot{\omega} \\ \dot{z}_4 = V_2 = \dot{\omega} \end{cases} \quad (3-37)$$

posant :

$$\begin{cases} x_1 = \omega \\ x_2 = \dot{\omega} \end{cases} \quad \text{donc :} \quad \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} V \quad (3-38)$$

en choisissant C_{em} comme sortie, nous obtenons :

$$\dot{y} = \frac{J}{p} \dot{\omega} + K_f \omega + C_r \quad (3-39)$$

Donc le système s'écrit :

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \quad , \text{ tel que : } y = y' - C_r \text{ et } C = [K_f \quad J/p] \quad (3-40)$$

Pour ce système, nous appliquerons la commande par retour d'état afin d'imposer les pôles voulus, donc le système en boucle fermée devient :

$$\begin{cases} \dot{x} = -Kx + U \\ y = C \times (s \times I - A + BK)^{-1} BU \end{cases} \quad (3-41)$$

Pour l'obtention de la poursuite, nous multiplions l'entrée c'est à dire la référence (ici U) par un pré compensateur N tel que :

$$N = [C \times (-A + BK)^{-1} \times B]^{-1} \quad (3-42)$$

ainsi quand $t \rightarrow \infty \Rightarrow y \rightarrow U$.

Le développement de l'équation (3-41) donne :

$$y(s) = \frac{K_1}{K_f} \frac{Js + K_f}{s^2 + K_2s + K_1} U(s) - C_r \quad (3-43)$$

d'où :

$$y'(s) = \frac{K_1}{K_f} \frac{Js + K_f}{s^2 + K_2s + K_1} U(s) \quad (3-44)$$

Finalement, le bloc actionneur dans la figure (3.3) devient :

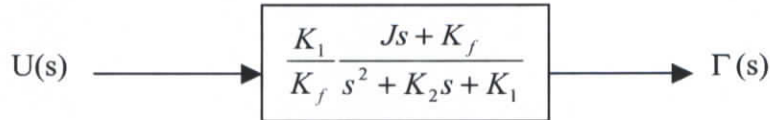


Figure 3.4 : La fonction de transfert de la boucle fermée de la boucle interne.

Dans notre logiciel, l'utilisateur peut choisir le type d'actionneur « moteur asynchrone », et ainsi faire rentrer les paramètres J , K_f et C_r (caractéristiques spécifiques du moteur asynchrone), et choisir K_1 ainsi que K_2 (paramètres du régulateur par placement de pôle), ceci va faire intervenir la dynamique des moteurs asynchrones dans la dynamique des différentes articulations. Dans cette partie, nous avons aussi supposé que les paramètres des différents moteurs sont identiques sauf leur inertie J , car cette dernière rentre dans le calcul de la dynamique des différentes articulations.

3.4 Résultats de simulations:

Dans cette partie, nous allons présenter une série de simulations appliquées aux robots :Puma 560 et Scara Pour cela, nous utiliserons notre application développée avec le langage C++ ainsi que, la méthode de runge-kutta d'ordre 4 à pas variable pour la résolution des systèmes différentiels.

Cette série de simulations, représente les résultats de la partie simulation de notre application, en plus, elles nous permet de vérifier les comportements de la commande (couple calculé).

Nous allons dans ce qui suit essayer d'appliquer la méthode du couple calculé sur les deux robots et ceci seulement pour les trois premières articulations en tenant compte à chaque fois de l'effet de l'actionneur (moteur à courant continu ou moteur asynchrone).

Simulation 01 :

Nous commençons avec le robot **Puma 560**: les paramètres de cette simulation sont les suivants:

Actionneur: moteur à courant continu; $k = 1, \tau = 1$ et on choisi un modèle de référence tel qu'il est décrit par l'équation (3-21), avec $\alpha = 1$.

L' inertie des trois moteurs est donnée par: $I_{m1} = 1.14 (kg.m^2)$, $I_{m2} = 4.71 (kg.m^2)$
 $I_{m3} = 0.83 (kg.m^2)$

Trajectoire: polynomiale cubique avec:

$q_{10} = 0, q_{1f} = 120$ (degré) , $q_{20} = -30, q_{2f} = 45$ (degré), $q_{30} = 30, q_{3f} = 90$ (degré).
 $t_f = 8$ (second).

Régulateur: les paramètres sont donnés par le tableau suivant.

Articulation Paramètre	Articulation 1	Articulation 2	Articulation 3
k_p	4	2	5
k_v	1	4	1
k_i	1	6	1

Tableau 3.1: Paramètres du régulateur

Les résultats sont représentés sur la figure 3.5:

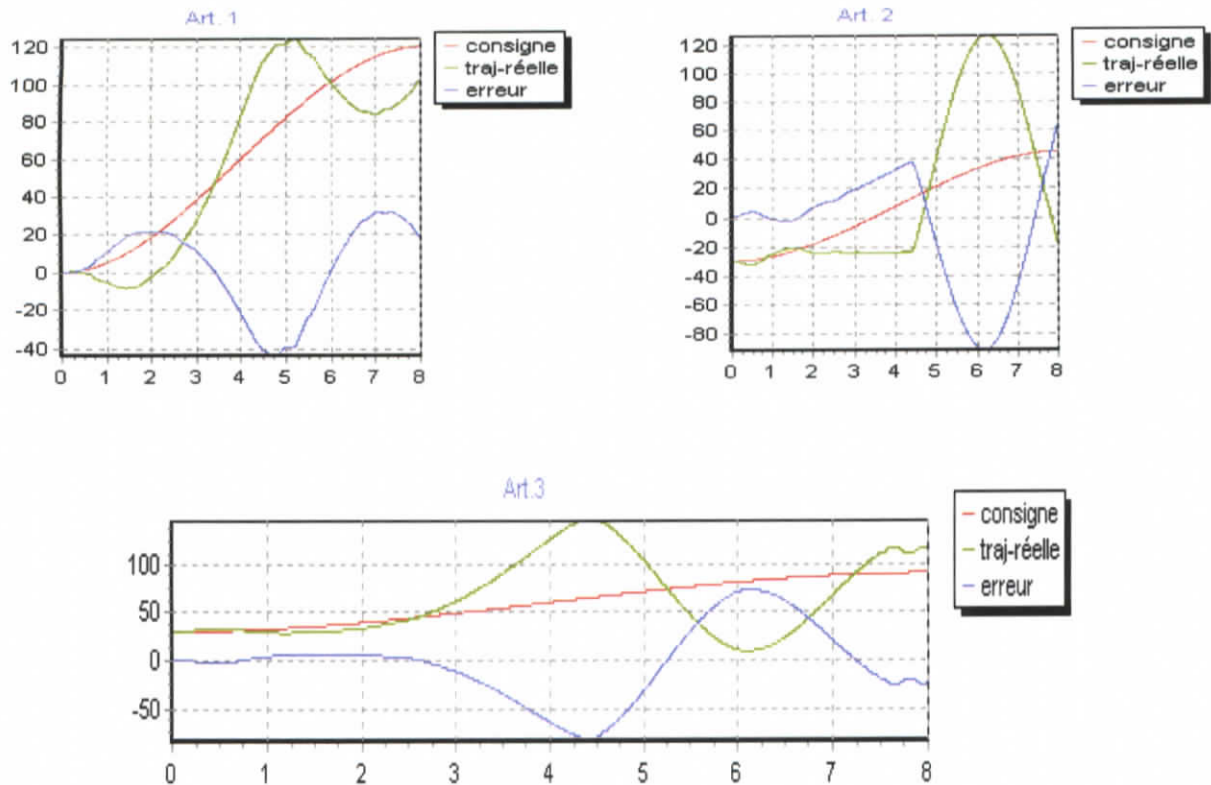


Figure 3.5 : Réponses en position des trois articulations.

Remarque:

Nous constatons qu'il n'y a pas de poursuite de trajectoires. Ceci est dû, aux dynamiques des moteurs qui sont plus lentes que la dynamique du système global.

Simulation 02 :

Nous allons garder pour la prochaine simulation les mêmes paramètres pour les moteurs, la trajectoire et le modèle de référence et nous essayerons d'améliorer les réponses par le changement des paramètres du régulateur. Le nouveau tableau des paramètres de régulateur est le suivant:

Articulation \ Paramètres	Articulation 1	Articulation 2	Articulation 3
k_p	10	12	10
k_v	8	8	6
k_i	8	12	10

Tableau 3.2: Paramètres du régulateur

Les nouvelles réponses seront comme suit:

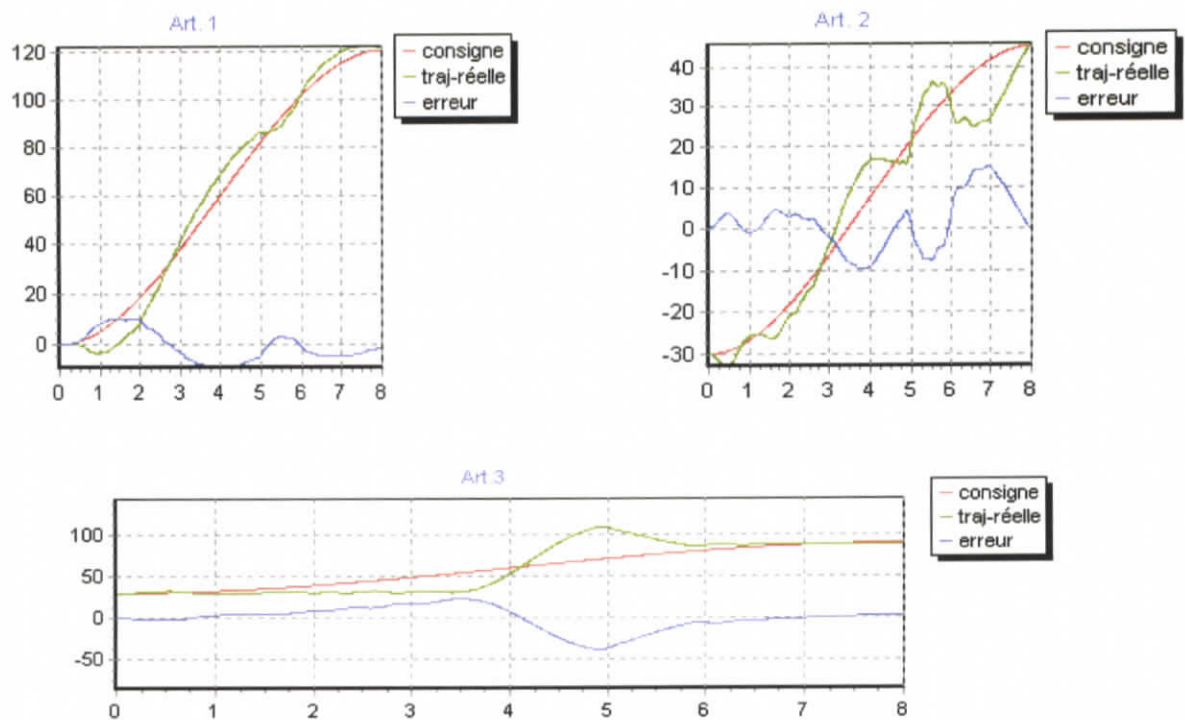


Figure 3.6 : Réponses en position des trois articulations.

Remarque:

Le changement de paramètres n'a pas beaucoup d'influence sur la dynamique des articulations, car la boucle interne n'est pas assez rapide (les dynamiques des moteurs en boucle fermée sont lentes)

Simulation 03 :

Nous proposons par la suite de revenir au premier essai, avec le seul changement de $\alpha = 10$ (nous gardons les paramètres du tableau 3.1). Le but de cet essai est de voir l'influence de la commande interne sur le système globale (le robot).

Les nouvelles réponses sont représentées sur la figure 3.7.

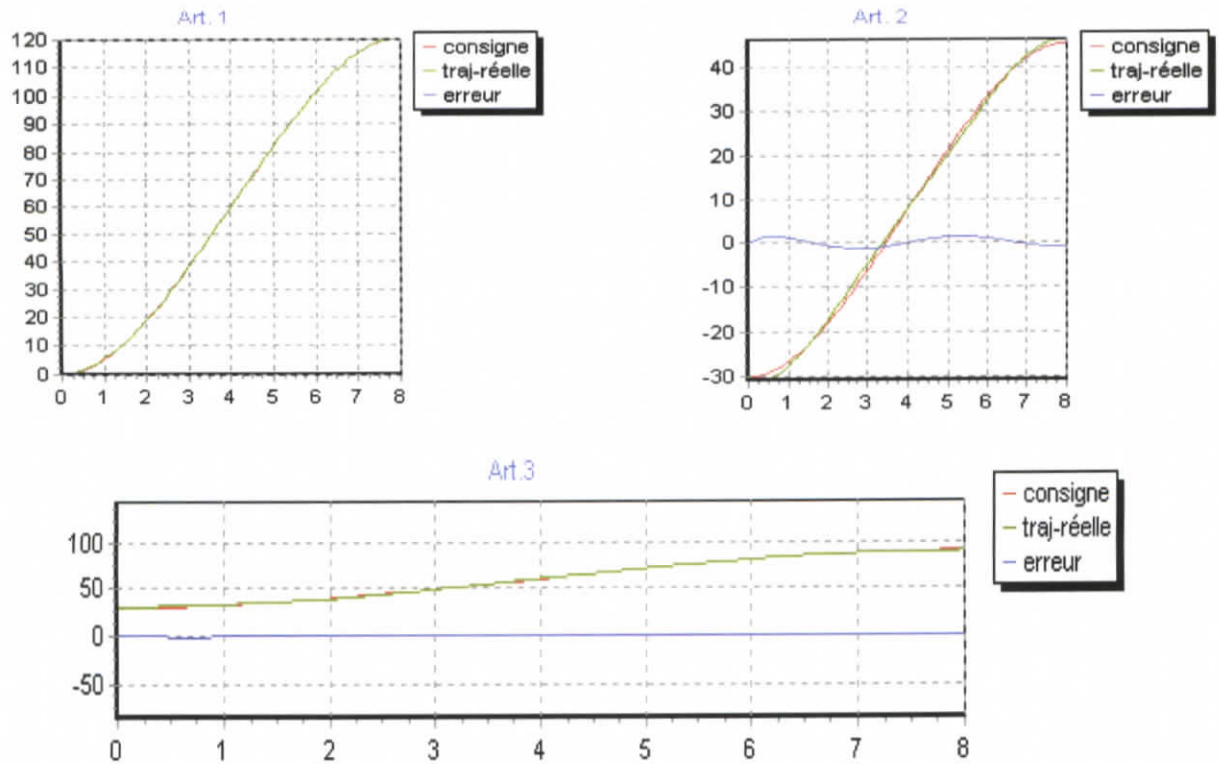


Figure 3.7: Réponses en position des trois articulations.

Remarque:

Nous constatons bien dans ce cas, que les réponses sont nettement meilleures et l'erreur est de l'ordre de 1 degré.

Simulation 04:

Nous passons maintenant, à la trajectoire sinusoïdale, cette dernière est en fait considérée comme une trajectoire de test, c'est pour cela que nous proposons une simulation, afin de voir la poursuite d'une telle trajectoire.

Les paramètres de cette simulation sont les suivants:

Actionneur: moteur à courant continu; $k = 1, \tau = 1$ et on choisi un modèle de référence tel $\alpha = 20$.

L'inertie des trois moteurs est donnée par: $I_{m1} = 1.14 (kg.m^2)$, $I_{m2} = 4.71 (kg.m^2)$
 $I_{m3} = 0.83 (kg.m^2)$

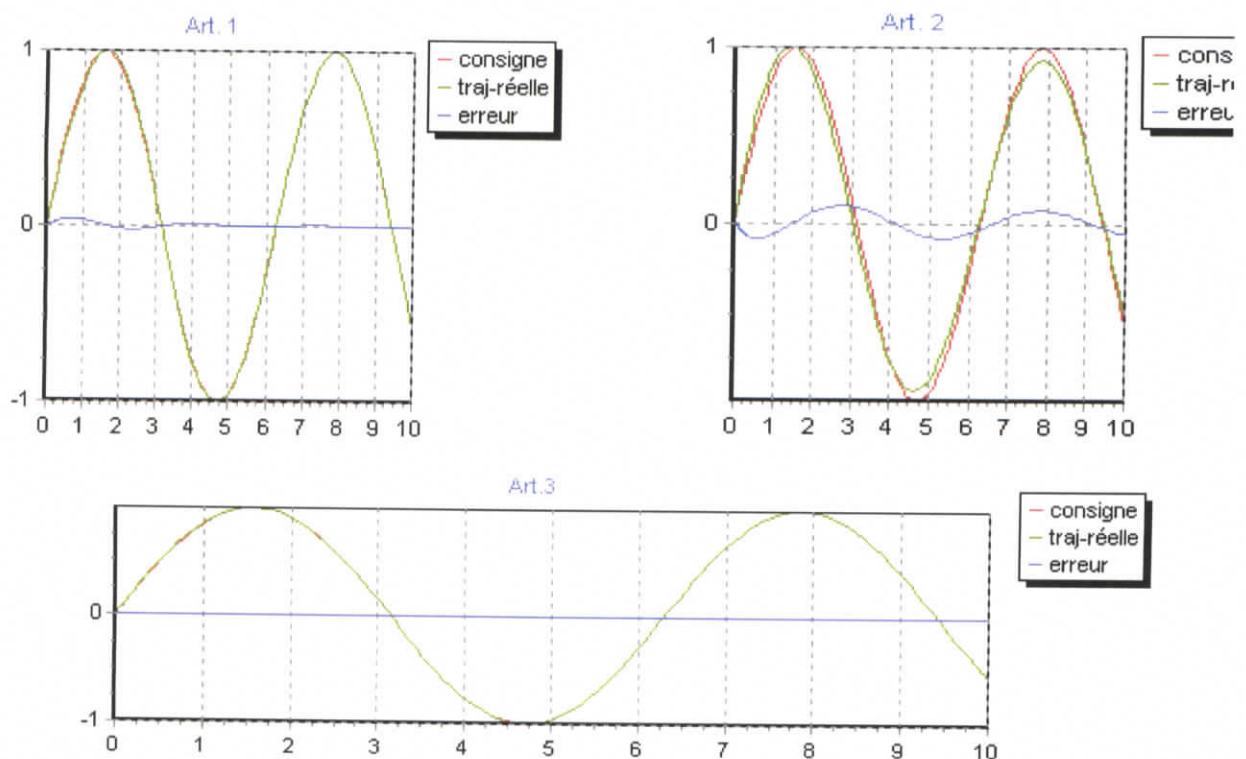
Trajectoire: sinusoïdale avec $t_f = 8 (second)$.

Régulateur:

Articulation Paramètre	Articulation 1	Articulation 2	Articulation 3
k_p	4	2	5
k_v	1	4	1
k_i	1	6	1

Tableau 3.3: Paramètres du régulateur

Les nouvelles réponses seront comme suit:

**Figure 3.8 :** Réponses en position des trois articulations.**Conclusion :**

La prise en compte de l'influence de l'actionneur sur la dynamique totale des articulations montre bien que la dynamique de la boucle de réglage interne (de l'actionneur) doit être plus rapide que celle des articulations. Par conséquent, nous devons choisir un modèle de référence avec un α élevé pour assurer la poursuite des trajectoires de référence.

Simulation 05 :

Nous utilisons dans cette partie le moteur asynchrone comme actionneur. Les paramètres de cette simulation seront :

Actionneur: moteur asynchrone avec les paramètres : $k_f = 0.0018 \text{ (N.m/rad.s}^{-1}\text{)}$, $C_r = 28 \text{ (N.m)}$

L' inertie des trois moteurs est donnée par: $I_{m1} = 1 \text{ (kg.m}^2\text{)}$, $I_{m2} = 2 \text{ (kg.m}^2\text{)}$

$$I_{m3} = 1 \text{ (kg.m}^2\text{)}$$

le régulateur secondaire (de la boucle interne) est défini par : $k_1 = 1, k_2 = 1$.

Trajectoire: polynomiale cubique avec:

$$q_{10} = 0, q_{1f} = 45 \text{ (degré)} , q_{20} = 0, q_{2f} = -90 \text{ (degré)}, q_{30} = -30, q_{3f} = 60 \text{ (degré)}.$$

$$t_f = 5 \text{ (second)} .$$

Régulateur: les paramètres sont donnés par le tableau suivant :

Articulation Paramètre	Articulation 1	Articulation 2	Articulation 3
k_p	8	8	8
k_v	5	2	5
k_i	6	1	6

Tableau 3.4: Paramètres du régulateur

Les réponses sont représentées sur la figure 3.9 :

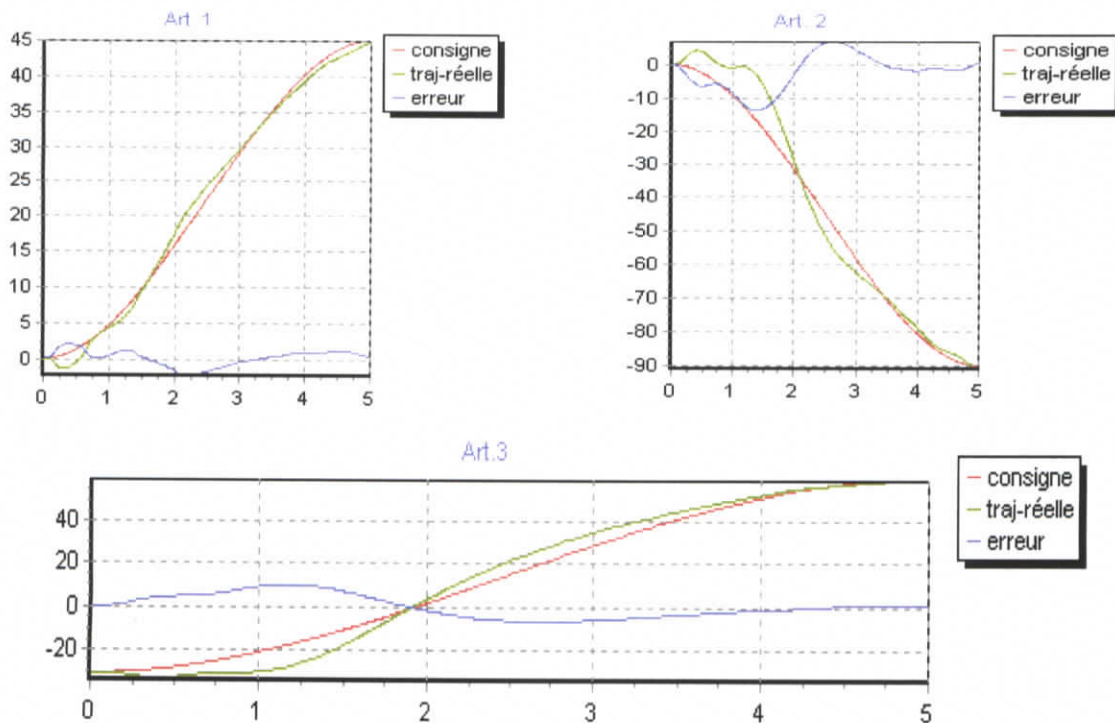


Figure 3.9 : Réponses en position des trois articulations.

Remarque :

L'erreur est plus grande pour la troisième articulation, elle est de l'ordre de 10 degrés.

Simulation 06 :

Nous allons jouer sur les paramètres de réglage pour diminuer l'erreur en position des différentes articulations.

Les nouvelles valeurs choisies sont les suivantes :

Articulation \ Paramètre	Articulation 1	Articulation 2	Articulation 3
k_p	12	14	10
k_v	10	10	8
k_i	10	8	8

Tableau 3.5: Paramètres du régulateur

Les réponses obtenues sont ci-dessous :

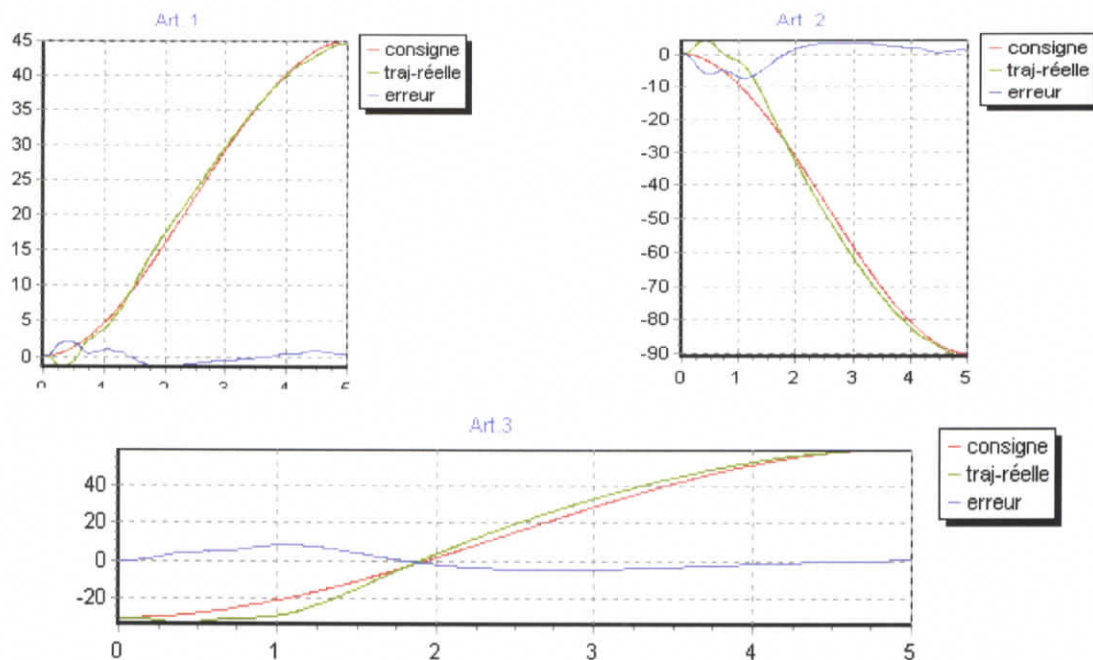


Figure 3.10 : Réponses en position des trois articulations.

Remarque :

Les réponses montrent que les paramètres du réglage n'influent pas trop sur la dynamique des trois articulations, par la suite nous allons essayer de changer la dynamique de la boucle interne par la méthode de placement de pôles.

Simulation 07 :

Nous allons maintenant changer le modèle de la boucle interne (placement de pôles) et on garde les mêmes paramètres de réglage pour le régulateur ; on prend $k_1 = 20$ et $k_2 = 30$ (les paramètres de réglage sont les mêmes que ceux du tableau 3.4).

Trajectoire: polynomiale cubique avec:

$q_{10}=0$, $q_{1f}=60$ (degré) , $q_{20}=0$, $q_{2f}=-45$ (degré), $q_{30}=20$, $q_{3f}=-45$ (degré) et $t_f = 5$ (second) .

Les nouvelles réponses sont les suivantes :

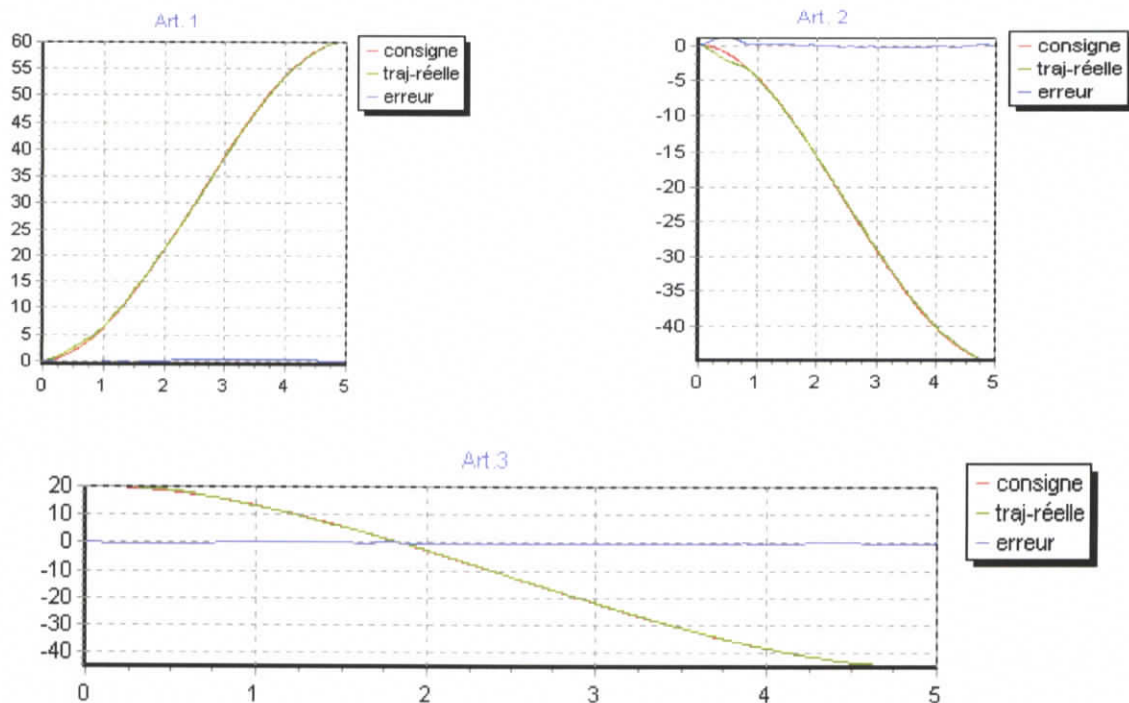


Figure 3.11 : Réponses en position des trois articulations.

Remarque :

Nous remarquons que l'erreur est très petite, elle est de l'ordre de 0.4 degrés. Ce changement de dynamique totale des articulations est dû au changement de la dynamique de l'actionneur.

Simulation 08 :

Nous allons par la suite tester la validité de cette dernière commande pour une trajectoire de type sinusoïdale et avec les paramètres suivants :

Trajectoire : sinusoïdale

Régulateur : placement de pôle tel que : $k_1=2$ et $k_2=3$ et les paramètres du réglage par la méthode du couple calculé sont donnés dans le tableau 3.6.

Articulation \ Paramètre	Articulation 1	Articulation 2	Articulation 3
k_p	10	24	10
k_v	6	10	6
k_i	8	15	8

Tableau 3.6: Paramètres du régulateur

On obtient par la suite les réponses suivantes :

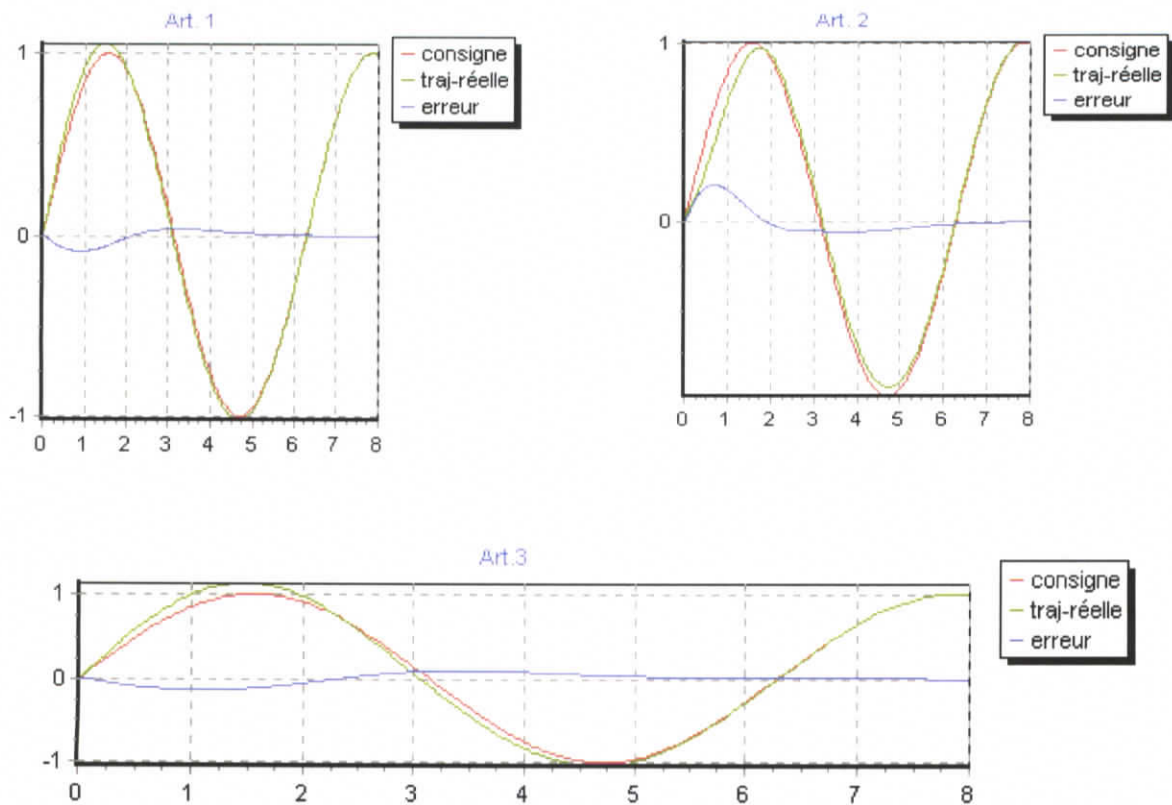


Figure 3.11 : Réponses en position des trois articulations.

Remarque :

Nous remarquons bien que l'erreur est de l'ordre de 0.1 degré et elle s'annule au bout de 7 secondes. Cette durée peut être réduite par l'ajustement des paramètres de réglage.

Conclusion :

Après cette série de simulations, nous pouvons dire que la boucle interne joue un rôle important dans la synthèse de la commande du couple calculé. Cette dernière, doit avoir une dynamique plus rapide que celle des articulations et donc du robot. L'implémentation de la commande du couple calculée avec la prise en compte de deux type d'actionneurs montre bien que les paramètres de la boucle interne sont en fait les paramètres de réglage les plus importants, car le réglage en cascade ne peut être réalisé que si la dynamique de boucle interne soit plus rapide que celle de la boucle externe.

Simulation 09 :

La simulation suivante est réalisée pour le robot **Scara** et avec une trajectoire de type polynomiale cubique, les paramètres sont donnés par :

Actionneur: moteur à courant continu; $k = 1, \tau = 1$ et on choisi un modèle de référence tel que $\alpha=2$.

L' inertie des trois moteurs est donnée par: $I_{m1}=10^{-4} (kg.m^2)$, $I_{m2}=10^{-4} (kg.m^2)$

Trajectoire: polynomiale cubique avec:

$q_{10}=40, q_{1f}=-110$ (degré) , $q_{20}=45, q_{2f}=-60$ (degré).

$t_f=5$ (second).

Régulateur:

Articulation Paramètre	Articulation 1	Articulation 2
k_p	2	4
k_v	2	5
k_i	3	6

Tableau 3.7: Paramètres du régulateur

Les réponses sont les suivantes :

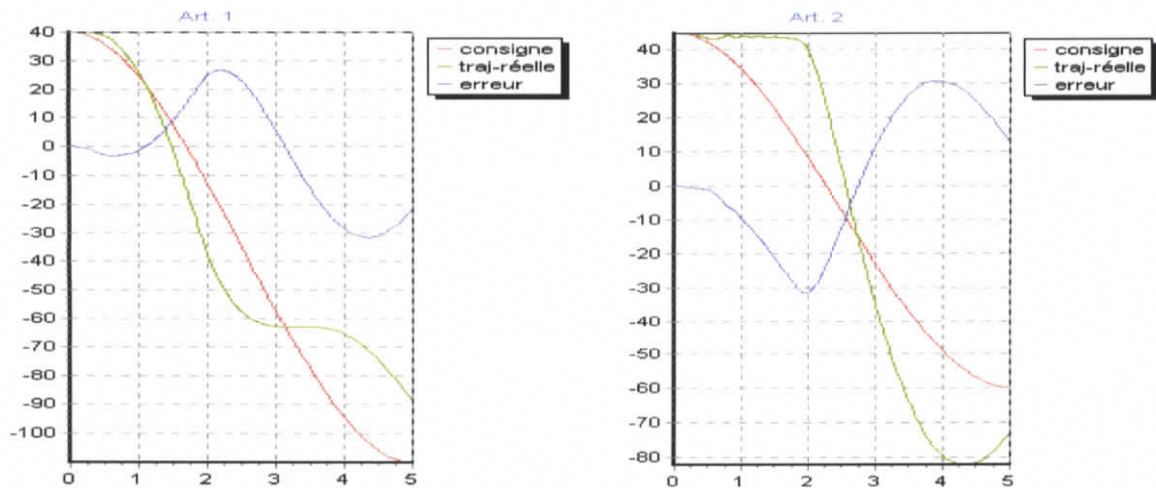


Figure 3.12 : Réponses en position des deux articulations.

Remarque :

Nous remarquons que le système diverge et les erreurs sont importantes, par conséquent nous allons effectuer une série de réglage sur les deux boucles (interne et externe) afin d'obtenir une bonne poursuite.

Simulation 10 :

Dans un premier temps, nous allons changer le paramètre de la boucle interne (le paramètre du modèle de référence)

Pour $\alpha=10$ et avec les mêmes régulateurs illustrés dans le tableau 3.7, on obtient les réponses :

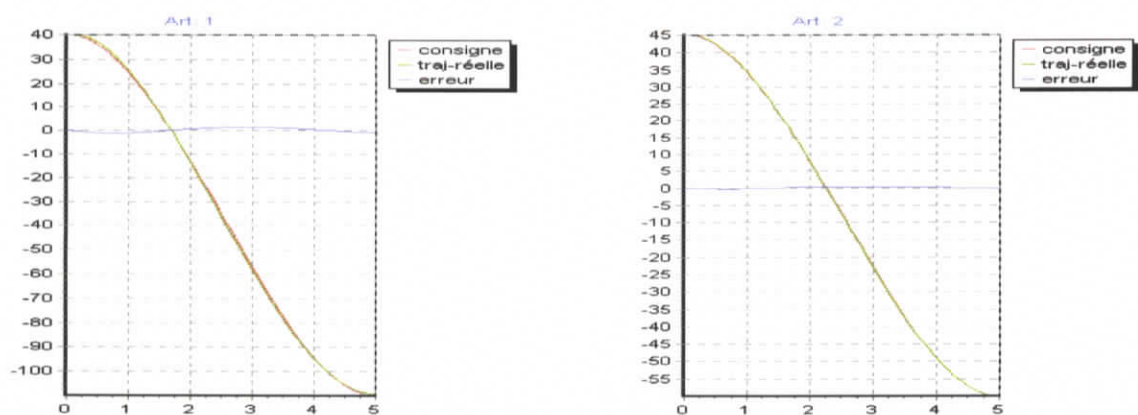


Figure 3.13 : Réponses en position des deux articulations.

Remarque :

L'erreur est de l'ordre de 1.5 degré, elle peut être améliorée par l'ajustement des paramètres du régulateur externe.

Simulation 11 :

Nous proposons maintenant un autre tableau de régulateurs et avec les mêmes trajectoires (polynomiales cubiques) mais avec $\alpha=2$.

Articulation \ Paramètre	Articulation 1	Articulation 2
k_p	10	20
k_v	4	8
k_i	8	12

Tableau 3.8: Paramètres du régulateur

Les réponses sont données par :

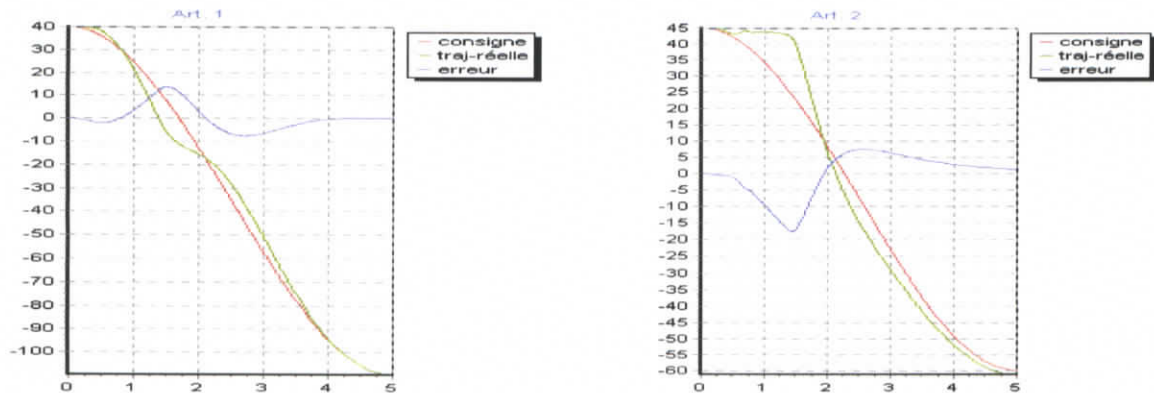


Figure 3.14 : Réponses en position des deux articulations.

Remarque :

Le réglage de la boucle interne joue toujours un rôle très important et donc on doit à chaque fois fixer un bon α et par la suite changer les paramètres du régulateur externe afin d'obtenir les performances désirées.

Simulation 12:

L'essai suivant consiste à choisir une trajectoire désirée de type sinusoïdale avec $\alpha=10$ et $t_f=8$. Les nouveaux paramètres de réglage sont :

Articulation \ Paramètre	Articulation 1	Articulation 2
k_p	2	4
k_v	2	5
k_i	3	6

Tableau 3.9: Paramètres du régulateur

Les réponses obtenues sont représentées par la figure ci-dessous :

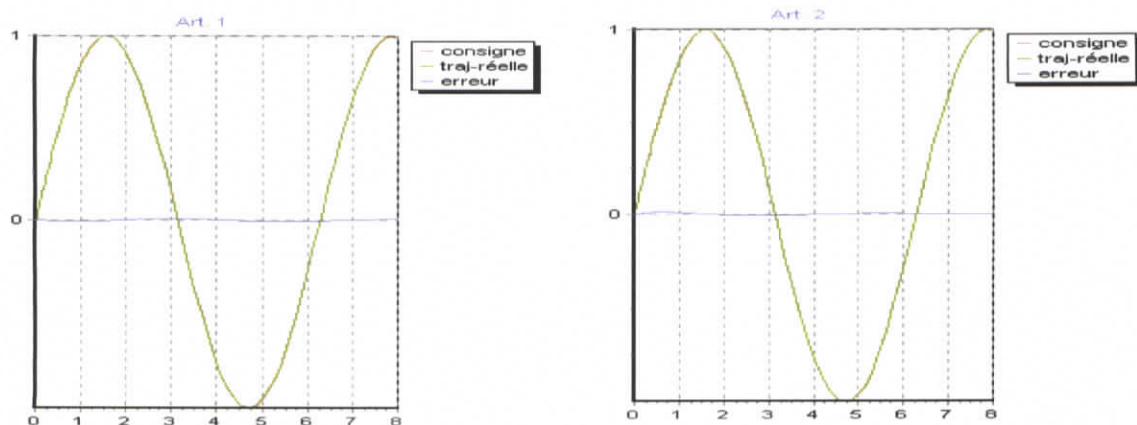


Figure 3.15 : Réponses en position des deux articulations.

Remarque :

L'erreur est de l'ordre de $5 \cdot 10^{-3}$ degré, nous pouvons dire que c'est un bon réglage et la poursuite est atteinte après un temps de 4 secondes.

Conclusion :

Dans cette partie, nous remarquons que l'utilisation du moteur à courant continu assure de bonnes réponses si la dynamique de ce dernier en boucle fermée est plus rapide que la dynamique des articulations.

Simulation 13:

Nous allons continuer toujours avec le même robot (Scara) mais cette fois avec un deuxième type de moteur (moteur asynchrone). Les paramètres de cette simulation sont les suivants :

Actionneur: moteur asynchrone avec les caractéristiques suivantes :

$$k_f = 0.0018 \text{ (N.m/rad.s}^{-1}\text{)}, C_r = 28 \text{ (N.m)}, I_{m1} = 1 \text{ (kg.m}^2\text{)}, I_{m2} = 2 \text{ (kg.m}^2\text{)}.$$

Le régulateur interne est défini par : $k_1 = 4$ et $k_2 = 5$

Trajectoire : polynomiale cubique avec définie par :

$$t_f = 6 \text{ (second)}, q_{10} = 30, q_{1f} = -45 \text{ (degré)} \text{ pour la première articulation}$$

$$\text{et } q_{10} = -20, q_{2f} = 90 \text{ (degré)} \text{ pour la seconde articulation.}$$

Régulateur : les paramètres du réglage sont représentés par le tableau 3.10 :

Articulation \ Paramètre	Articulation 1	Articulation 2
k_p	5	8
k_v	2	5
k_i	3	6

Tableau 3.9: Paramètres du régulateur

Les réponses en position de chaque articulation sont données par la figure 3.16 :

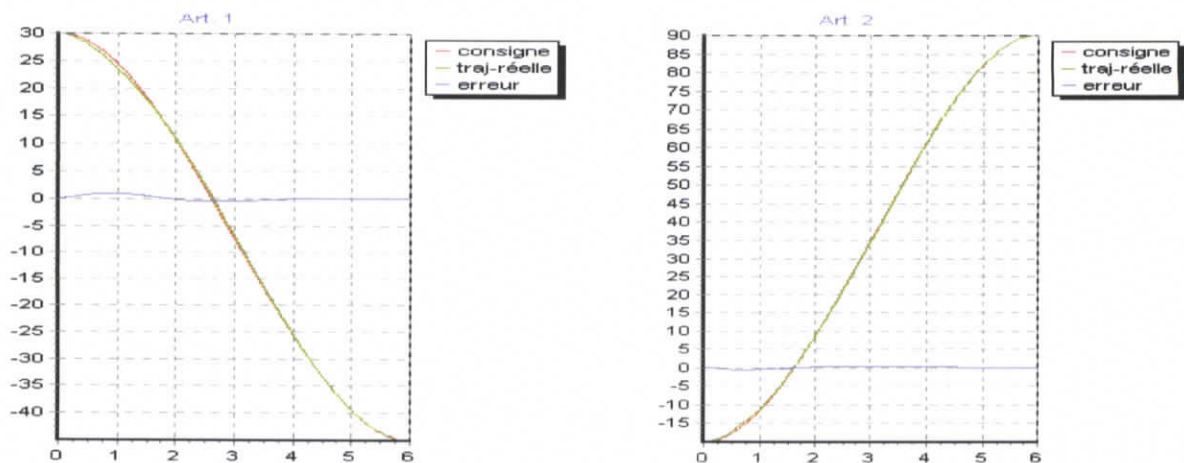


Figure 3.16 : Réponses en position des deux articulations.

Remarque :

l'erreur maximale est de l'ordre de 1 degré et on peut dire que la poursuite est atteinte après 4 secondes.

Simulation 14 :

Nous gardons les mêmes paramètres de la dernière simulation mais avec le changement des paramètres du retour d'état de la boucle interne : $k_1 = 10$ et $k_2 = 20$

Les nouvelles réponses sont données par la figure 3.17:

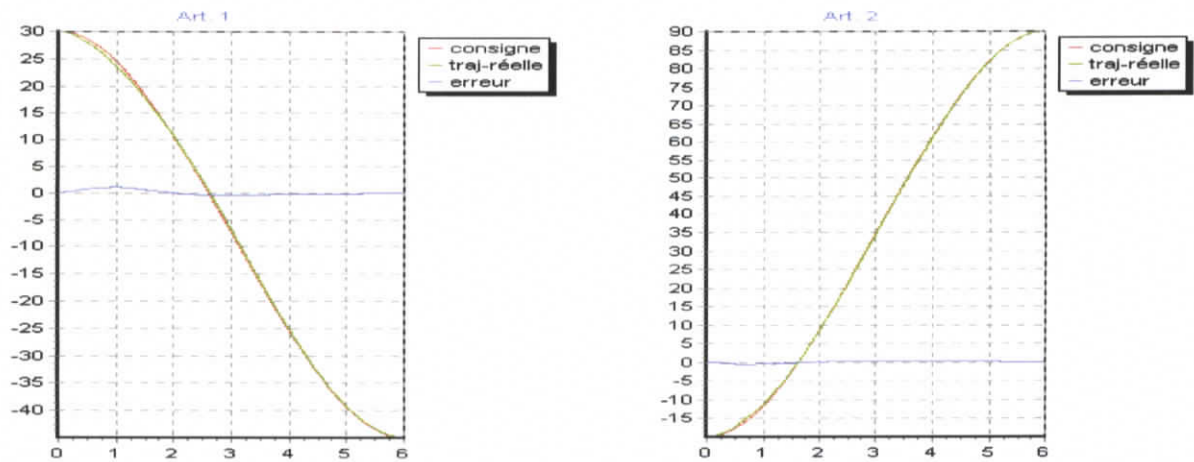


Figure 3.17 : Réponses temporelles des trois articulations.

Remarque :

la comparaison entre les deux dernières simulation à travers les figures 3.16 et 3.17 montre bien qu'il n'y a presque aucune différence entre les réponses, donc nous sommes amenés à faire varier les paramètres du contrôleur principal afin d'améliorer les réponses.

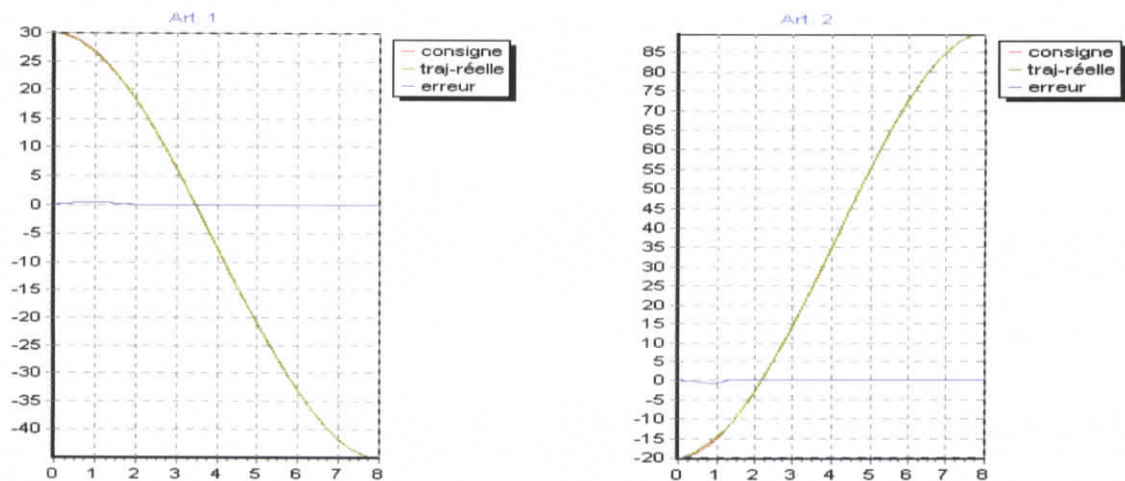
Simulation 15 :

si on conserve les derniers paramètres du régulateur interne mais on change les paramètres du régulateur principal comme l'indique le tableau 3.10 :

Articulation \ Paramètre	Articulation 1	Articulation 2
k_p	8	8
k_v	3	2
k_i	4	3

Tableau 3.10: Paramètres du régulateur

Avec un temps final $t_f = 8$ secondes, les réponses en position des trois articulations de notre robot seront :

**Figure 3.18 :** Réponses en position des deux articulations.**Remarque :**

l'erreur pour cette simulation est de l'ordre de 0.3 degré et la poursuite parfaite est atteinte après 1.8 secondes.

Simulation 16 :

Nous conservons les paramètres du régulateur interne et ceux du régulateur global, et nous changeons le type de trajectoire (trajectoire sinusoïdale).

Les réponses de cette dernière simulation sont représentées par la figure suivante :

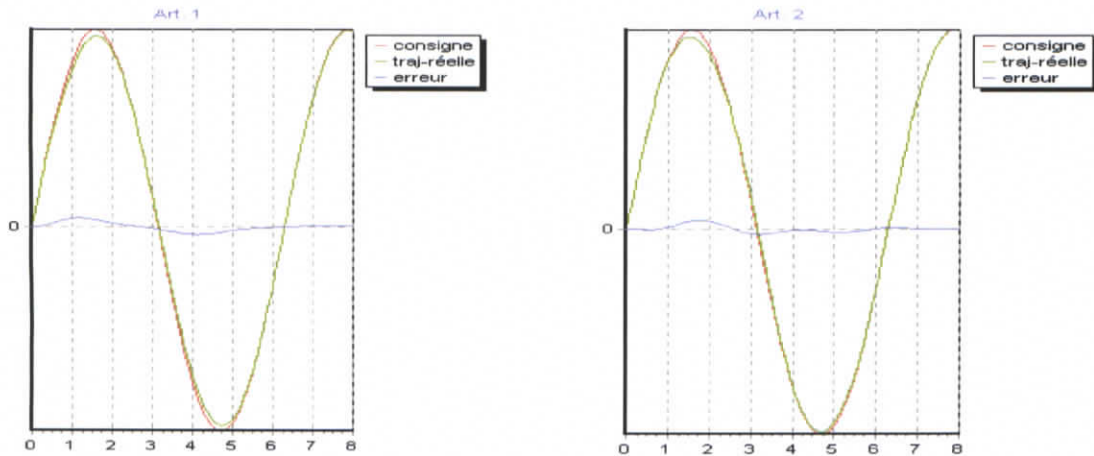


Figure 3.19 : Réponses en position des deux articulations.

Remarque :

Nous avons pu avoir des bonnes réponses pour chacune des trajectoires avec le même régulateur.

En résumé, les simulations effectuées montrent que le réglage de la boucle interne joue un rôle important pour commander le système correctement. Certaines performances ne peuvent être atteintes que si on réalise un compromis entre le réglage du moteur et le réglage des articulations proprement dit. La méthode implémentée facilite la synthèse de la commande adéquate pour la réalisation de performances ou de cahiers des charges donnés, car elle ne contient que trois paramètres de réglage simple à manipuler.

3.5 Conclusion :

Ce chapitre, représente le support théorique de la partie simulation de notre application. C'est pour cela, que nous avons explicité les différentes étapes de modélisation des deux moteurs (actionneurs) utilisés dans la simulation de la commande des robots, ainsi que le principe de la commande classique, qui représente le type de commande le plus utilisé dans l'industrie.

De plus, l'implémentation de cette commande nous a permis d'apprécier sa simplicité et la facilité de manipuler ses paramètres. Et enfin, nous avons pu constater l'influence des actionneurs dans une boucle de réglage où le système est un bras manipulateur, et ainsi nous avons déduit que la dynamique de la boucle interne (des actionneurs) doit être plus rapide que celle des articulations du robot.

Chapitre 4

Présentation du Logiciel

4.1 Introduction :

Le but principal de ce travail, est la conception d'une interface graphique permettant d'animer et de commander plusieurs robots (trois plus exactement), et enfin d'établir la communication à travers le port RS 232 entre un microcontrôleur et l'ordinateur pour la commande en temps réel des bras manipulateurs. Pour atteindre ce but, nous avons été amené à utiliser deux types de logiciels de programmation à savoir : le logiciel Builder C++ 5.3 et le logiciel Matlab 6.1 ainsi que la bibliothèque graphique OpenGL et la librairie d'outils Glut.

Dans ce chapitre, nous allons tout d'abord présenter brièvement les différents logiciels utilisés, en expliquant leurs choix et les avantages qu'ils présentent. Puis nous passerons à la présentation de notre logiciel. Dans cette partie, nous allons exposer les différentes palettes (pages) qui constituent le logiciel et expliquer le rôle de chacune d'entre elles.

4.2 Logiciels utilisés :

4.2.1 Builder C++ [37],[38],[39],[40],[41]:

Tout d'abord C++ est un outil RAD, c'est à dire tourné vers le développement rapide d'applications (Rapid Application Development) sous Windows. En un mot, C++ Builder permet de réaliser de façon très simple l'interface des applications et de relier aisément le code utilisateur aux événements Windows, quelle que soit leur origine (souris, clavier, événement système, etc.) Pour ce faire, C++ Builder repose sur un ensemble très complet de *composants visuels* prêts à l'emploi. La quasi-totalité des contrôles de Windows (boutons, boîtes de saisies, listes déroulantes, menus et autres barres d'outils) y sont représentés, regroupés par famille. Leurs caractéristiques sont éditables directement dans une fenêtre spéciale intitulée *éditeur d'objets*. L'autre volet de cette même fenêtre permet d'associer du code au contrôle sélectionné. Il est possible d'ajouter à l'environnement de base des composants fournis par des sociétés tierces et même d'en créer soit même. Un outil RAD c'est également un ensemble de *squelettes* de projets qui permettent de créer plus facilement des applications et des objets. A chacun de ces squelettes est habituellement associé un *expert* qui par une série de boîtes de dialogues permet de fixer une partie des options essentielles à la réalisation du projet associé.

Ainsi, c'est pour la simplicité de son utilisation et pour la diversité des objets (prés à l'emploi) qu'il offre, que nous nous sommes intéressés au langage Builder C++. La figure 4.1 représente l'espace de travail du langage Builder C++.

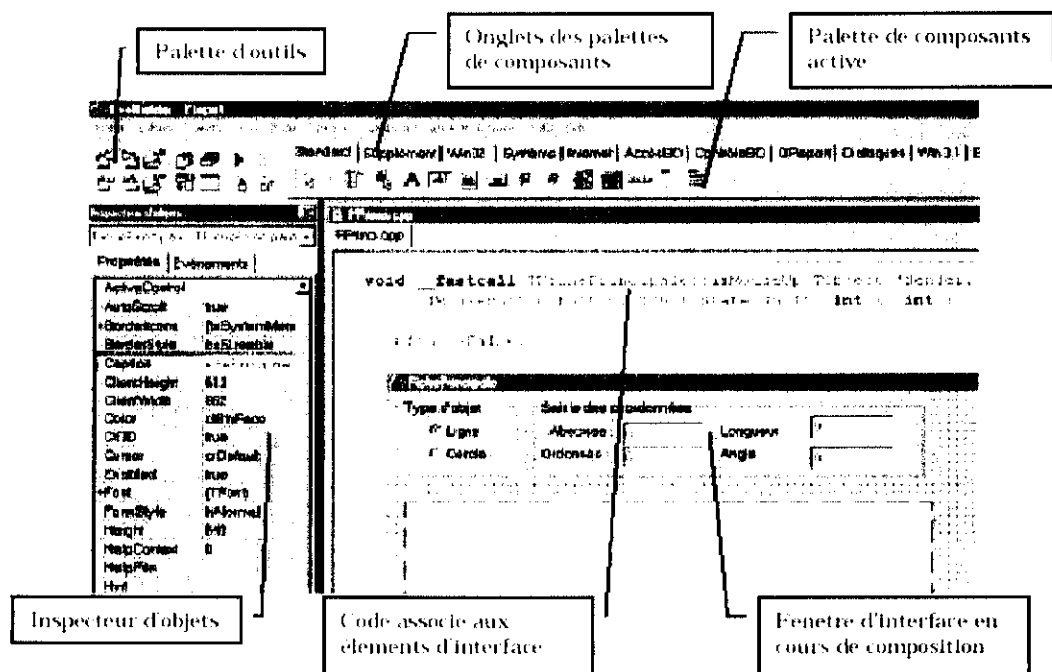


Figure 4.1 : Espace de travail du langage Builder C++

4.2.3 Matlab [42],[43],[44] :

Le logiciel Matlab consiste en un langage interprété qui s'exécute dans une fenêtre dite *d'exécution*. L'intérêt de Matlab tient, d'une part, à sa simplicité d'utilisation : pas de compilation, déclaration implicite des variables utilisées et, d'autre part, à sa richesse fonctionnelle : arithmétique matricielle et nombreuses fonctions de haut niveau dans de nombreux domaines (analyse numérique, graphique, ...). La programmation sous Matlab consiste à écrire des *scripts* de commandes Matlab, exécutables dans la fenêtre *d'exécution*. En outre, grâce aux diverses *Toolboxes* spécialisés (ensemble de scripts Matlab), Matlab s'enrichit au fur et à mesure.

En ce qui nous concerne, nous avons utilisé la version Matlab 6.1, pour deux raisons, la première est une raison évidente (car le logiciel Matlab est construit autour), c'est sa puissance en calcul matriciel, d'autant plus que les modèles dynamiques des robots sont représentés sous forme matricielle, mais aussi ils sont fortement non linéaires, c'est pour cela que nous avons utilisé la fonction ODE 45 pour résoudre les équations différentielles afin d'obtenir les dynamiques des différentes articulations pour les différents robots. La deuxième raison, est l'utilisation du générateur de code C, pour l'obtention des exécutables des programmes Matlab utilisés afin de les utiliser sous l'environnement Builder C++.

4.2.3 Bibliothèques utilisées :

4.2.3.1 La bibliothèque graphique OpenGL [45],[46],[47]:

OpenGL [48] est une librairie graphique, c'est à dire une interface logiciel permettant d'effectuer des opérations d'affichage sur un écran graphique. Cette interface, développée par Silicon Graphics, est constituée d'environ 150 fonctions graphiques, de l'affichage de points et segments jusqu'au plaquage de textures sur des objets tridimensionnels. OpenGL présente l'intérêt majeur d'être indépendante du matériel utilisé, donc de l'architecture sur laquelle l'application graphique est développée. De fait, OpenGL n'a pas de fonctions de contrôle du matériel, en sortie ou en entrée. Le développeur doit, soit utiliser directement les fonctions du système sur lequel il développe son application (Xwindow par exemple pour Unix), soit utiliser une interface logiciel située en amont d'OpenGL et qui propose des fonctions pour cela (Glut par exemple).

De manière équivalente, OpenGL ne propose pas de fonctions graphiques de haut niveau utilisant des primitives géométriques complexes (sphères, splines, etc.). La réalisation de ces fonctions est laissée au développeur ou à l'interface logiciel en amont d'OpenGL. Cette spécification très ciblée de la librairie lui assure une portabilité importante et a fait d'OpenGL le standard graphique actuel.

En résumé OpenGL est une librairie graphique de bas niveau utilisant les primitives géométriques de base : points, segments et polygones. L'interfaçage avec le matériel (carte graphique, mémoire vidéo, etc.) ou avec des commandes de haut niveau (modèles 3D) doit être ajouté pour construire une application graphique.

4.2.3.2 La librairie Glut [49],[50]:

Glut est une librairie d'outils (Utility Toolkit) permettant l'interfaçage de la librairie OpenGL avec plusieurs architectures matérielles : stations de travail (SGI, SUN, ...), PCs, etc. Glut propose un ensemble de fonctions pour la gestion de fenêtres, de la souris et du clavier. Ces fonctions sont simples à utiliser et permettent de réaliser facilement de petites applications graphiques.

Par ailleurs, Glut propose certaines primitives de haut niveau (sphère, cylindre par exemple). Glut reste par contre limitée à la réalisation d'applications graphiques simples, permettant de tester et/ou de valider des algorithmes graphiques. Pour la réalisation d'applications graphiques plus complexes, le développeur aura intérêt à utiliser des librairies plus complètes (OpenInventor par exemple).

4.3 Organisation du programme:

Dans ce paragraphe, nous allons décrire quelques étapes du programme, chaque partie de ce dernier est faite sous une forme modulaire pour simplifier les appels entre les différentes « Forms » de l'application. L'architecture choisie permet aussi des modifications, des corrections et des optimisations faciles et indépendantes.

Le programme principal « main » représente la chaîne principale de notre application, il comporte toutes les déclarations : bibliothèques, objets utilisés, variables, fonctions et procédures, ainsi que tous les appels de fonctions ou procédures pour accomplir une tâche donnée.

Ce programme ne suit pas une architecture séquentielle évidente car le langage C++ utilisé est « orienté objet » et par conséquent, nous ne pouvons pas décrire un organigramme principal. Néanmoins, nous allons illustrer quelques fonctions qui sont importantes et fréquemment utilisées dans le logiciel.

4.3.1 Les fonctions MGD:

Ces fonctions permettent le calcul du modèle géométrique direct pour une pose donnée du robot, leurs expressions sont détaillées dans le deuxième chapitre. Selon le choix du robot, nous divisons cette famille de fonctions en trois groupes :

- Les fonctions MGD_Puma560
- Les fonctions MGD_Scara
- Les fonctions MGD_Robot_Hybride

La programmation de ces fonctions est faite indépendamment pour chaque coordonnée et d'une manière identique pour les trois types de robots. Nous allons illustrer ces différentes fonctions intermédiaires pour le calcul du modèle géométrique direct du robot Puma 560.

Dans ces fonctions, nous introduisons les variables articulaires, c'est à dire les angles de rotation du robot pour obtenir la position du poignet (l'effecteur).

float MGDpuma1(float v1,float v2,float v3)

Cette fonction représente la première fonction intermédiaire, elle a comme paramètres les trois angles de rotations du robot et elle retourne la position du poignet selon l'axe x.

float MGDpuma2(float v1,float v2,float v3)

Cette fonction représente la deuxième fonction intermédiaire, elle a comme paramètres les trois angles de rotations du robot et elle retourne la position du poignet selon l'axe y.

float MGDpuma3(float v2,float v3)

Cette fonction représente la troisième fonction intermédiaire, elle a comme paramètres deux des angles de rotations du robot et retourne la position du poignet selon l'axe z.

L'appel de ces fonctions pour le calcul du modèle géométrique direct est illustré dans l'organigramme suivant qui est valable pour les trois robots.

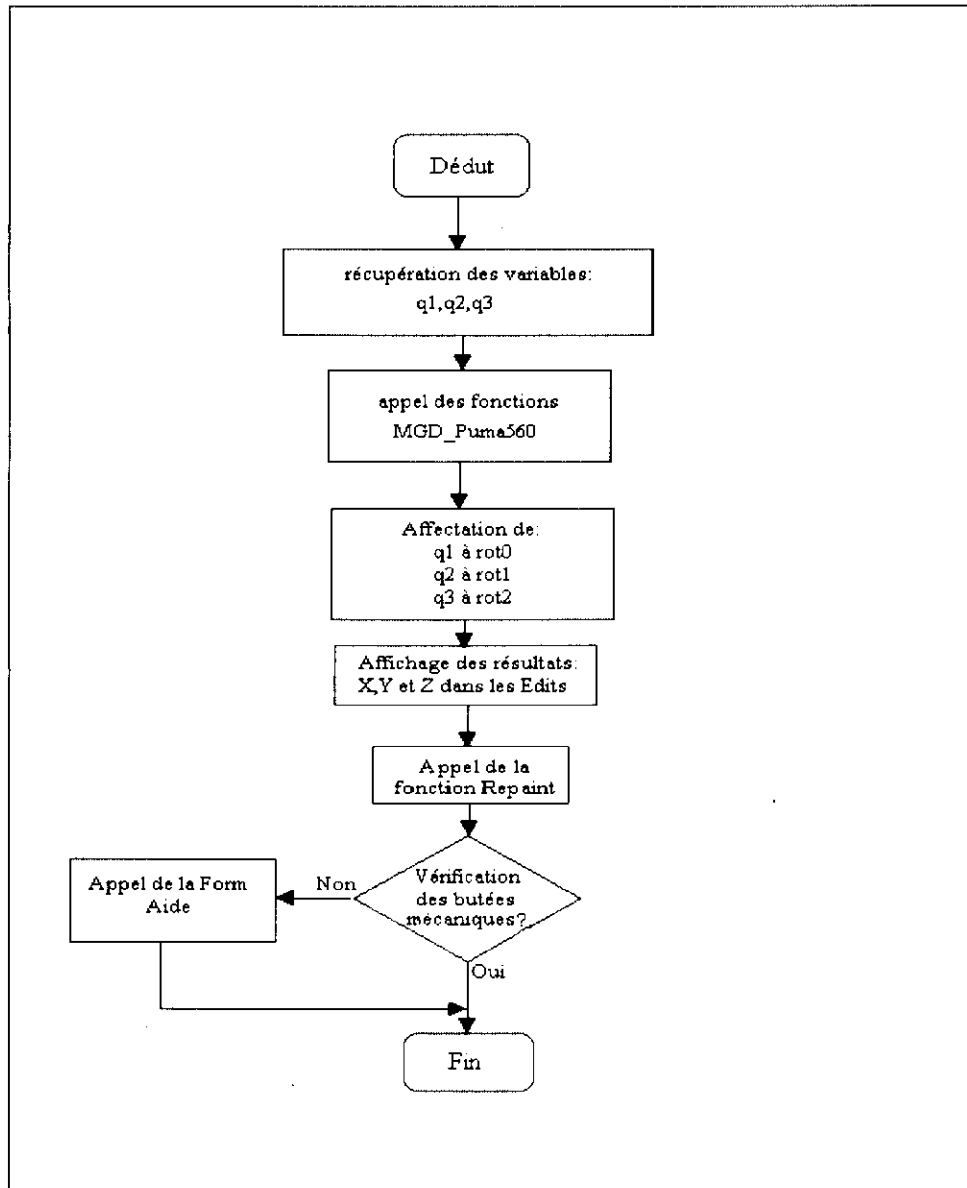


Figure 4.2 : Organigramme de calcul du modèle géométrique direct du robot Puma 560.

4.3.2 Les fonctions MGI:

La famille de ces fonctions permet le calcul du modèle géométrique inverse. Elle comporte trois groupes de fonctions selon le choix du robot :

- Les fonctions MGI_Puma560
- Les fonctions MGI_Scara
- Les fonctions MGI_Robot_Hybride

La programmation de cette famille de fonctions est faite d'une manière identique pour tous les robots. Par la suite, nous allons détailler ces fonctions pour le robot Scara.

Dans ces fonctions, nous introduisons les variables cartésiennes, c'est à dire la position désirée du poignet (l'effecteur) en 3D (x,y,z) comme paramètres et ces fonctions nous retournent les angles de rotations possibles afin d'atteindre cette position. Ces fonctions donnent plusieurs résultats (exactement 2 pour ce type de robot) pour une même position du poignet, c'est pour cela qu'il faut choisir la manière avec laquelle le robot doit atteindre la position cible. Voici les différentes possibilités :

Possibilité 1 :

```
float MGIs cara21(float x,float y)
float MGIs cara1(float x,float y, float q2)
float MGIs cara3(float z)
```

Possibilité 2:

```
float MGIs cara22(float x,float y)
float MGIs cara1(float x,float y, float q2)
float MGIs cara3(float z)
```

Ces fonctions sont des appels de prototypes déjà définis par les expressions du deuxième chapitre, pour avoir les formules analytiques, l'utilisateur trouvera les déclarations des prototypes au début de notre programme.

Nous pouvons schématiser l'appel de ces fonctions dans l'organigramme présenté par la figure 4.3.

4.3.3 La génération des trajectoires :

La génération de trajectoires peut se faire dans les deux espaces :

- L'espace articulaire.
- L'espace cartésien.

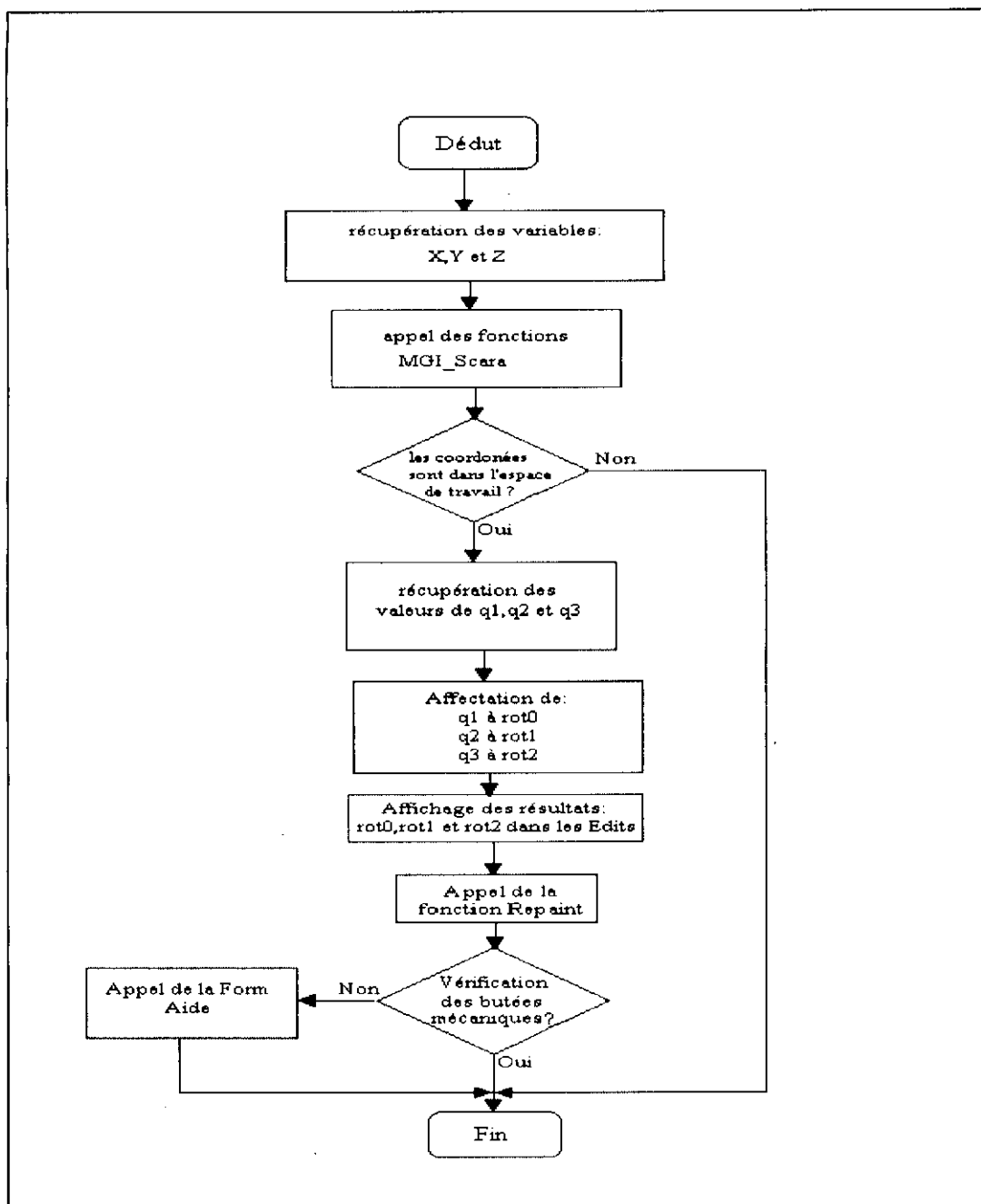


Figure 4.3 : Organigramme du calcul du modèle géométrique inverse du robot Scara.

La génération de trajectoire est basée essentiellement sur la manipulation des variables articulaires. Dans cette partie, nous avons voulu faire une animation des bras de robots dans l'espace articulaire ; en offrant à l'utilisateur trois types de trajectoires (le choix se fait par l'intermédiaire du ComboBox2). Nous avons utilisé l'objet « Timer1 » qui se trouve dans la palette « Système », le principe est simple car le Timer1 va nous permettre de découper de la trajectoire

choisie en petits segments en utilisant une période d'échantillonnage « la propriété Intervalle du Timer1 » fixée à l'avance. Les vecteurs obtenus sont affectés aux angles de rotation de chaque articulation pour chaque robot. Le schéma de la figure 4.4 illustre le déroulement d'une fonction de génération de trajectoire dans l'espace articulaire

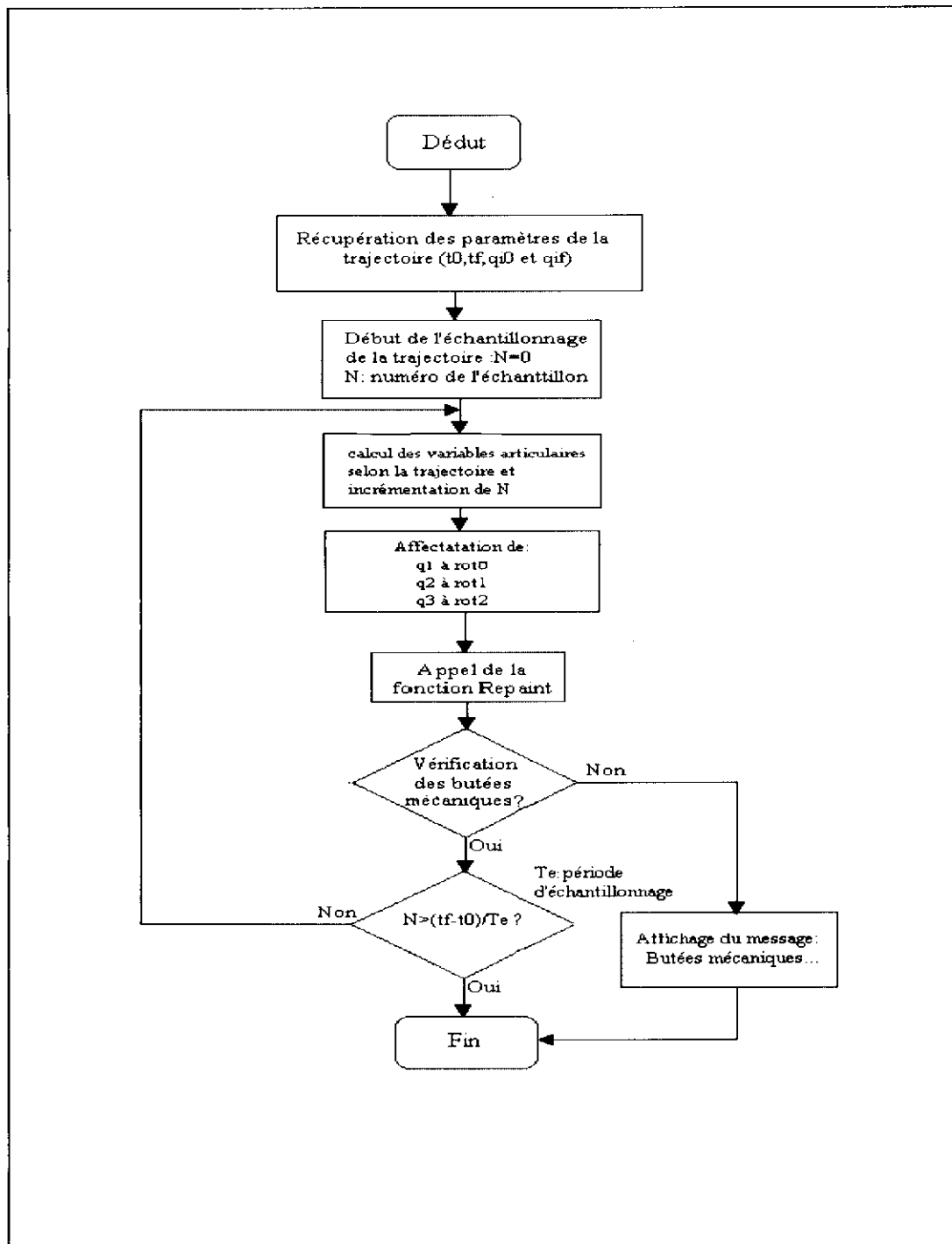


Figure 4.4 : Organigramme de la fonction de génération d'une trajectoire dans l'espace articulaire

Le principe de la génération de trajectoires dans l'espace cartésien est un peu différent car nous ne manipulons pas les coordonnées articulaires directement mais nous les récupérons par l'appel des fonctions de calcul du MGI à chaque période d'échantillonnage.

Le principe est simple ; nous donnons au robot une position d'origine et nous lui imposons une position finale, le robot va atteindre cette position cible en suivant une trajectoire échantillonnée, ceci est possible en utilisant le Timer 2 auquel nous fixons la propriété Intervalle (période d'échantillonnage). Ainsi, l'échantillonnage de la trajectoire donne un vecteur comportant les différentes positions intermédiaires par lesquelles doit passer le bras manipulateur pour atteindre sa cible. Chaque composante de ce vecteur est injectée dans le MGI du robot choisi, bien sûr en ayant d'abord la configuration (manière avec laquelle le robot va atteindre la cible) ceci donne les valeurs des variables articulaires correspondantes, ces dernières seront affectées aux variables de rotation du robot, ainsi nous pouvons observer l'animation du robot sur l'écran. L'organigramme de la figure 4.5 représente les étapes suivies pour générer une trajectoire.

4.3.4 Simulation du modèle dynamique :

Dans cette partie du logiciel, nous sommes amenés à résoudre des systèmes d'équations différentielles non-linéaire. La méthode choisie pour cette résolution est celle de Runge Kutta d'ordre 4.

Les fonctions qui peuvent résoudre des problèmes pareils existent dans la librairie Math du logiciel Matlab. Nous avons donc utilisé la fonction «ode45» pour ce calcul et nous avons pu intégrer cette fonction sous forme d'un exécutable.

Nous avons commencé par écrire quatre programmes en code source Matlab qui nous permettent de résoudre les systèmes différentiels et que nous avons nommé par la suite : «Puma1.m» et «Puma2.m» pour le robot Puma et «Scara1.m» et «Scara2.m» pour le robot Scara. Ces programmes nécessitent certains paramètres (les paramètres des actionneurs, du régulateur principal, etc.), pour cela nous avons programmé quelques instructions qui gèrent les flots d'entrées / sorties, car ils nous retournent aussi les résultats de simulation à chaque itération, ces résultats sont stockés dans un fichier de type «data.txt».

La deuxième étape consiste à compiler ces fonctions sous le logiciel Matlab «la version 6.1» à l'aide du compilateur 5.0 du langage C++. Les résultats de cette compilation sont quatre applications exécutables «Puma_moteur, Puma_MAS, Scara_moteur, Scara_MAS» qui seront appelés à chaque fois que l'utilisateur appuie sur le bouton «calcul» de la palette simulation du logiciel. La création des fichiers d'entrées/sorties se fait automatiquement lors de l'exécution de l'une des applications en arrière plan.

La récupération des résultats du fichier «data.txt» est faite par une allocation dynamique de la mémoire pour un nombre donné de vecteur et une dimension bien définie selon les paramètres de la simulation.

L'utilisateur pourra par la suite visualiser le mouvement sur la palette «robot» ou analyser les réponses sur la palette «graphe».

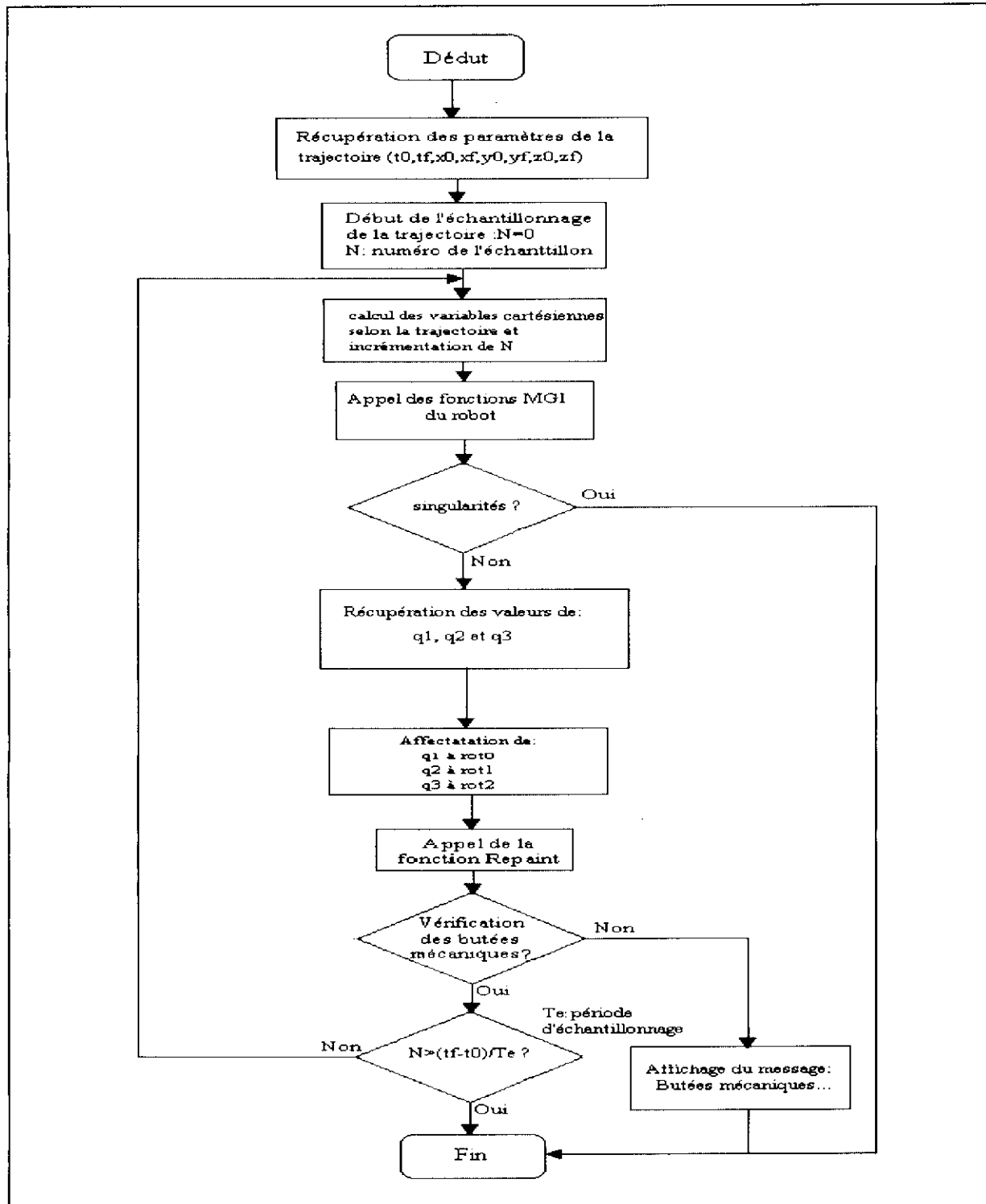


Figure 4.5 : Organigramme de la fonction de génération d'une trajectoire dans l'espace cartésien.

4.4 Interface développée :

Ayant exposé les différents logiciels et bibliothèques utilisés pour le développement de notre interface, nous passons maintenant, à la description de cette dernière. Dans notre interface, nous proposons un espace graphique, c'est à dire que nous offrons à l'utilisateur un espace tridimensionnel, où il peut projeter trois sortes de robots mais aussi, faire animer ces derniers et obtenir les différentes possibilités des modèles géométriques directes et inverses de ces robots.

Ensuite, il y a la partie commande, où l'utilisateur doit choisir le type d'actionneur utilisé, et ainsi constater les réponses des dynamiques des différentes articulations des robots et animer les robots, en tenant compte leurs dynamiques et celles des actionneurs. Dans ce qui suit, nous allons décrire les différentes parties de notre interface.

En premier lieu, l'utilisateur est soumis à un choix, c'est à dire qu'une fenêtre nommée **Choix** où sont présentés les trois robots vient interroger l'utilisateur sur le type de robot, cette dernière est représentée par la figure 4.6 .

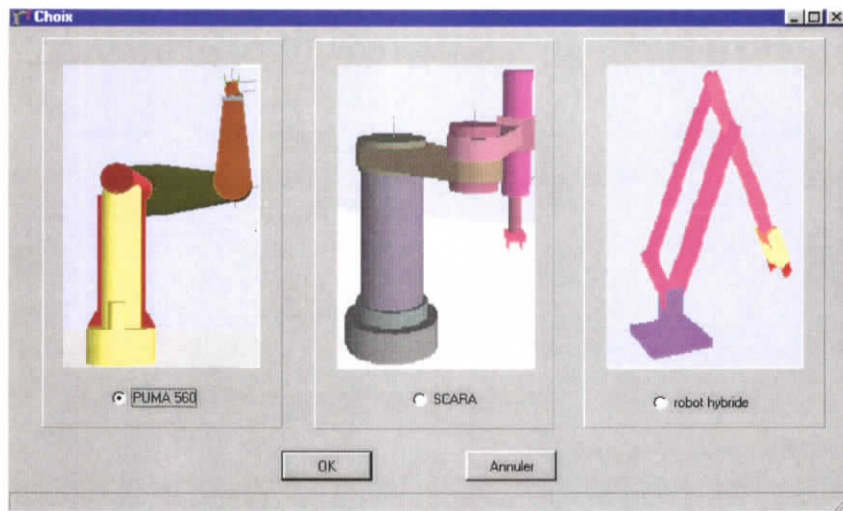


Figure 4.6 : Fenêtre Choix

En faisant un choix particulier, et en appuyant sur le bouton OK, l'utilisateur pourra accéder à la fenêtre suivante qui peut être l'une des fenêtres ci-dessous.

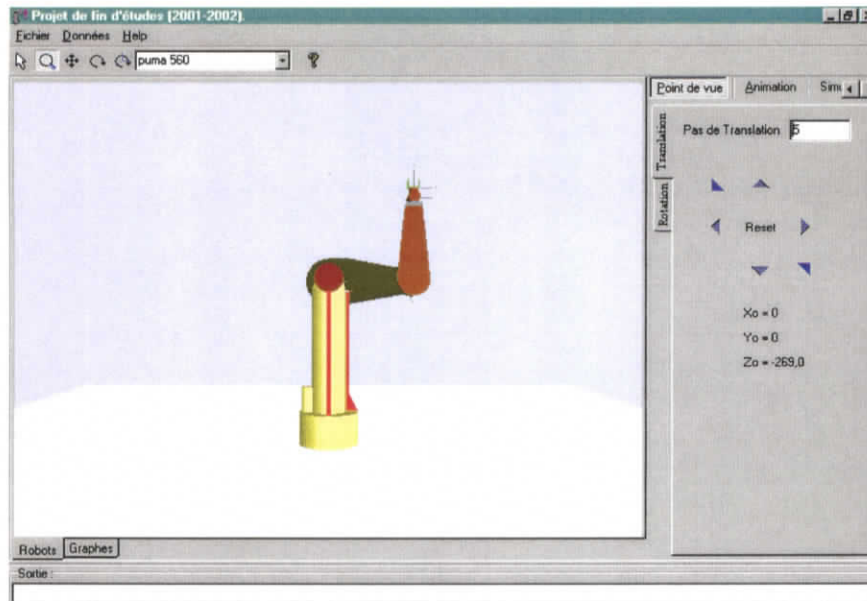


Figure 4.7 : Choix du robot Puma 560.

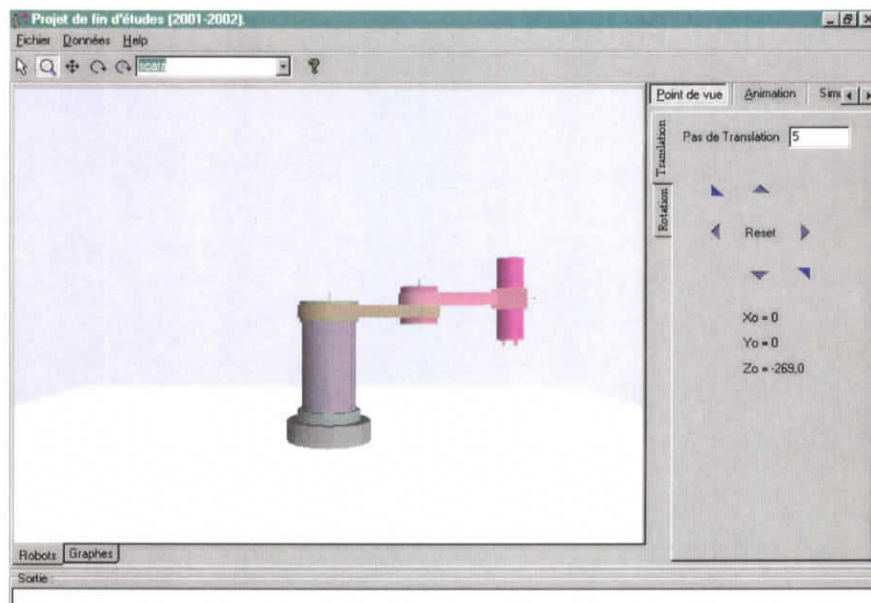


Figure 4.8 : Choix du robot Scara.

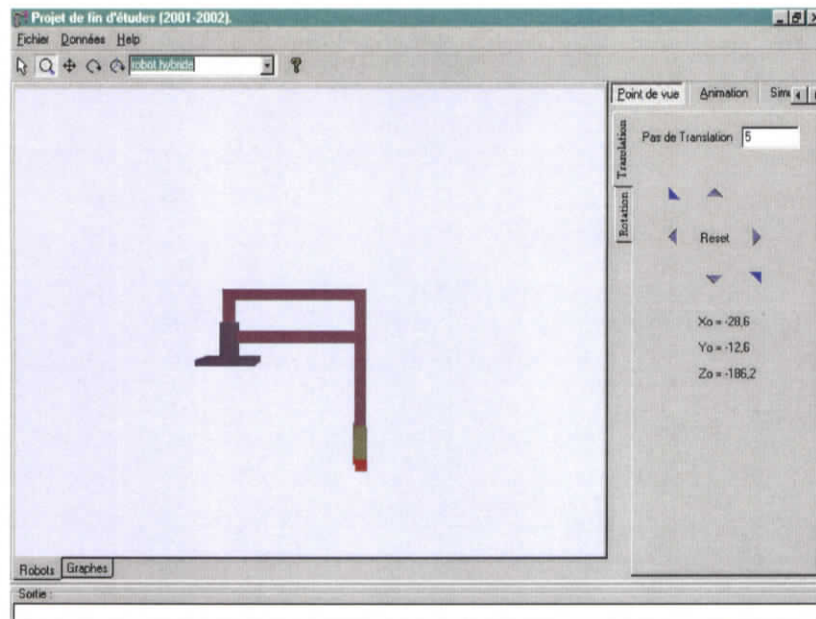


Figure 4.9 : Choix du robot Hybride.

Mais, si l'utilisateur a omis de choisir un des trois robots, et a appuyé sur le bouton OK, il peut faire son choix ultérieurement. La fenêtre qui apparaît si cela se produit est représentée dans la figure 4.10.

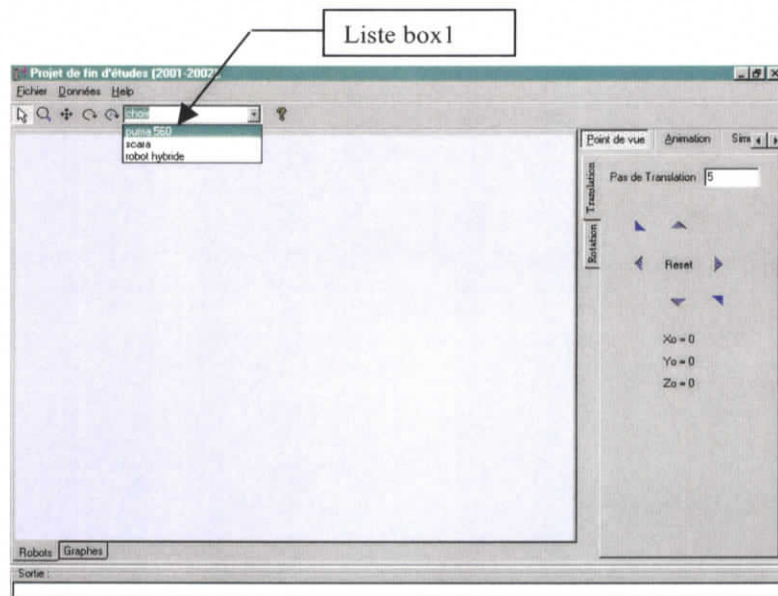


Figure 4.10 : Aucun Choix.

Nous remarquons, qu'aucun robot n'est représenté car comme nous l'avons dit cela se produit si l'utilisateur ne fait aucun choix au départ. Mais il a la possibilité de faire son choix

maintenant à travers le liste box1. Ceci nous pousse à décrire la barre de menu qui est représentée par la figure 4.11.

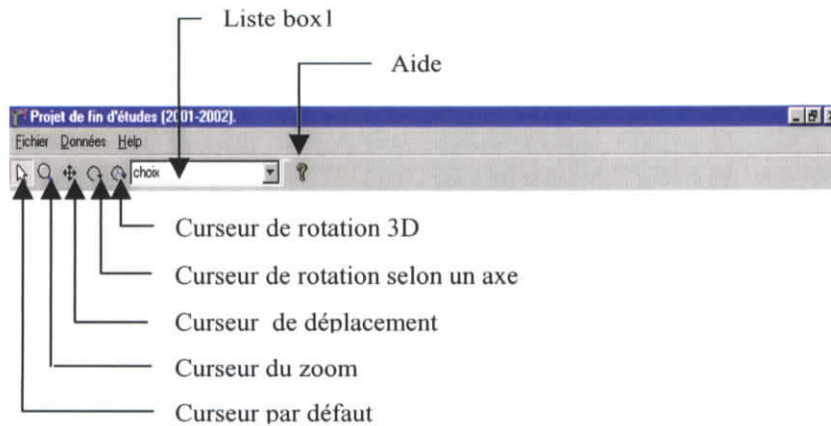


Figure 4.11 : La barre menu.

Nous remarquons, que ce menu comporte deux barres, la première porte les différents types de curseurs, le liste box1 ainsi que l'aide, la deuxième, comporte les menus : fichier, données et help auxquels nous ferons appel plus tard, aussi nous pouvons atteindre ces menus par des touches de raccourci tel que Alt+F, Alt+D et Alt+H pour fichier, données et help respectivement.

En ce qui concerne, la palette **Point de Vue**, celle ci comprend deux palettes secondaires : la palette rotation, qui permet de spécifier selon quel axe (x, y et z) nous voulons faire pivoter le robot ainsi que le sens de rotation (positif ou négatif). La palette translation, permet de translater le robot selon une direction positive ou négative par rapport à un axe. Mais le but principal de la palette **Point de Vue** est l'initialisation du dessin du robot, c'est à dire que si l'utilisateur a choisi un emplacement pour le robot et il a utilisé pour cela les curseurs (zoom, déplacement...), tous les déplacements et rotations effectués en utilisant les palettes translation et rotation seront effacés en appuyant sur le bouton **reset** de ces palettes et ainsi le robot retrouvera la position d'origine que l'utilisateur a choisi comme référence

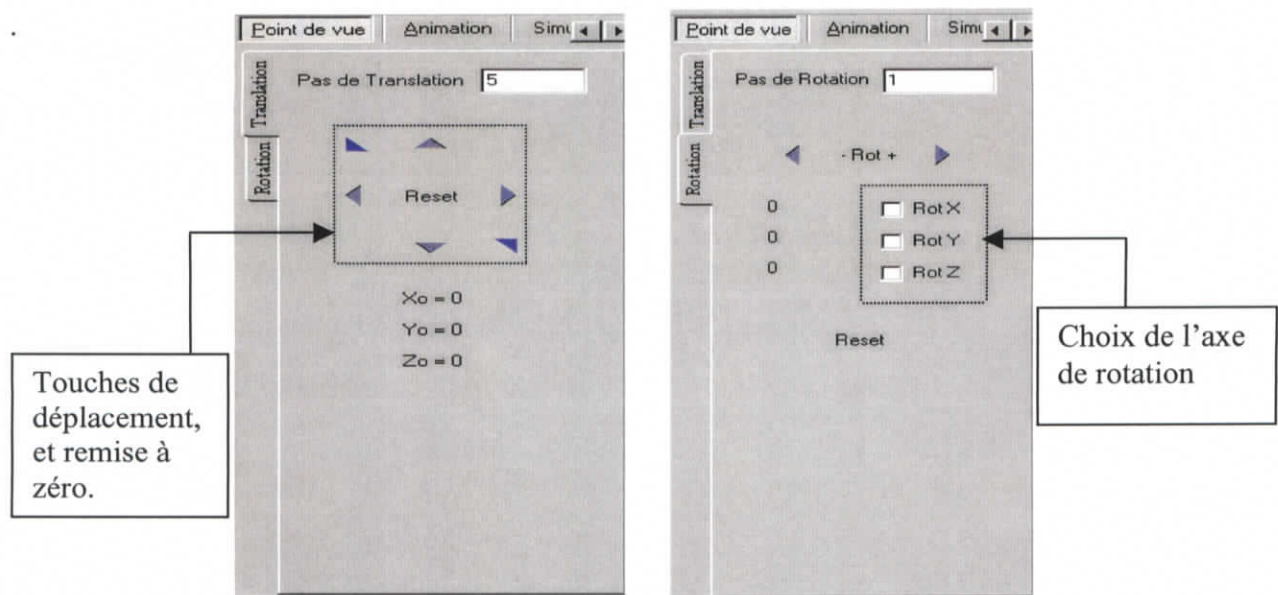


Figure 4.12 : les palettes secondaires de la palette **Point de Vue**.

L'utilisateur peut atteindre la palette **Point de Vue** avec la touche de raccourci Alt+P, pour les pas de translation et rotation, l'utilisateur est libre de les choisir. En ce qui concerne la translation, les déplacements effectués par le robot seront inscrits dans (X_0, Y_0, Z_0) à chaque instant et pour tout déplacement, ceci est valable aussi pour la rotation, car l'utilisateur peut choisir un, deux ou trois axes selon lesquels il va effectuer ses rotations, les angles relatifs par rapport à chaque axe seront inscrits à côté.

En appuyant sur la touche de raccourci Alt+A, ou tout simplement en cliquant sur la palette **Animation**, cette dernière se met en avant (elle s'active), nous remarquerons alors, qu'elle comprend trois palettes secondaires : la palette dessin, la palette trajectoires et la palette MGD/MGI.

En ce qui concerne la palette **Dessin**, celle ci nous offre la possibilité de voir le mouvement de chaque articulation indépendamment des autres, ceci en cliquant sur les différents Trackbars associés aux différentes coordonnées généralisées des robots. Aussi, à travers cette palette nous pouvons, à tout instant, savoir la position de l'effecteur dans l'espace, cette position est inscrite dans les Edits x,y et z que nous retrouvons tout en bas de la palette. Pour un choix particulier d'un robot, cette palette s'initialise et fait apparaître les coordonnées généralisées associées à ce dernier. La figure 4.13 représente la palette dessin et les figures 4.14, 4.15 et 4.16 correspondent à un choix particulier de robot et à la palette dessin associée à ce dernier initialisé.

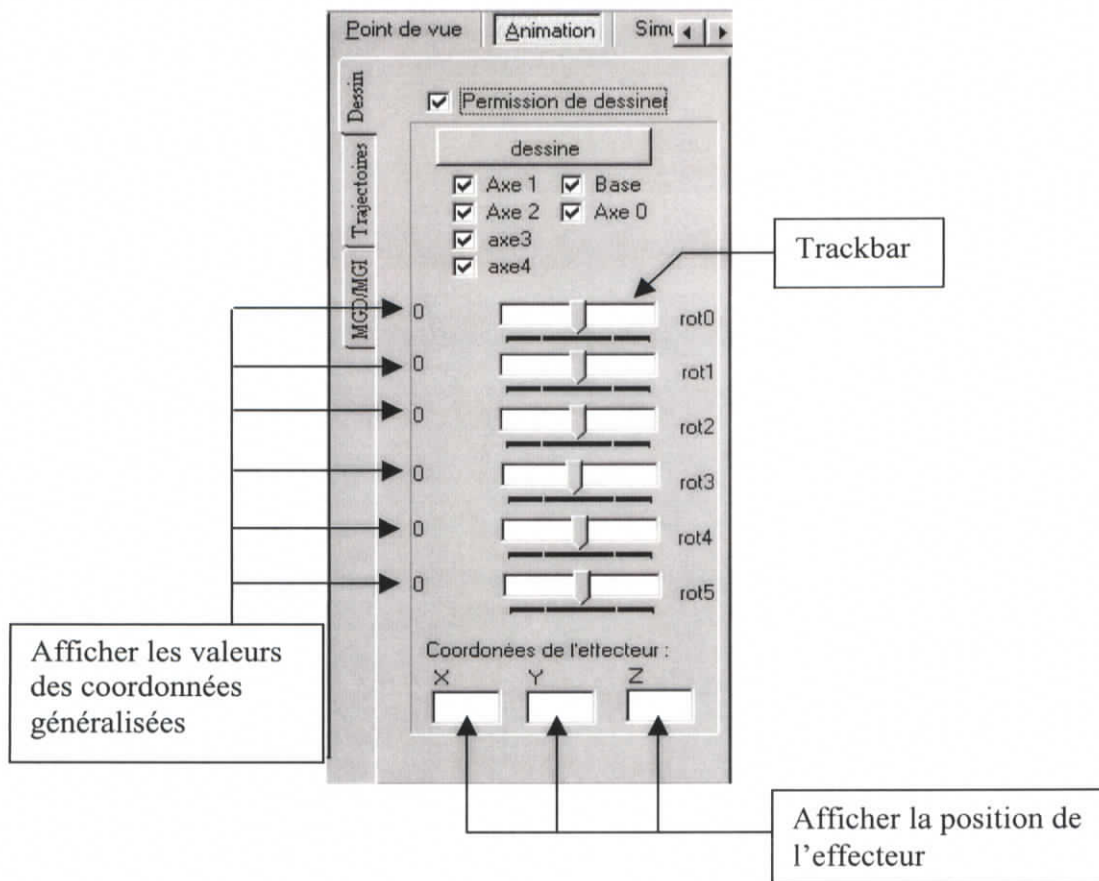


Figure 4.13 : La palette Dessin

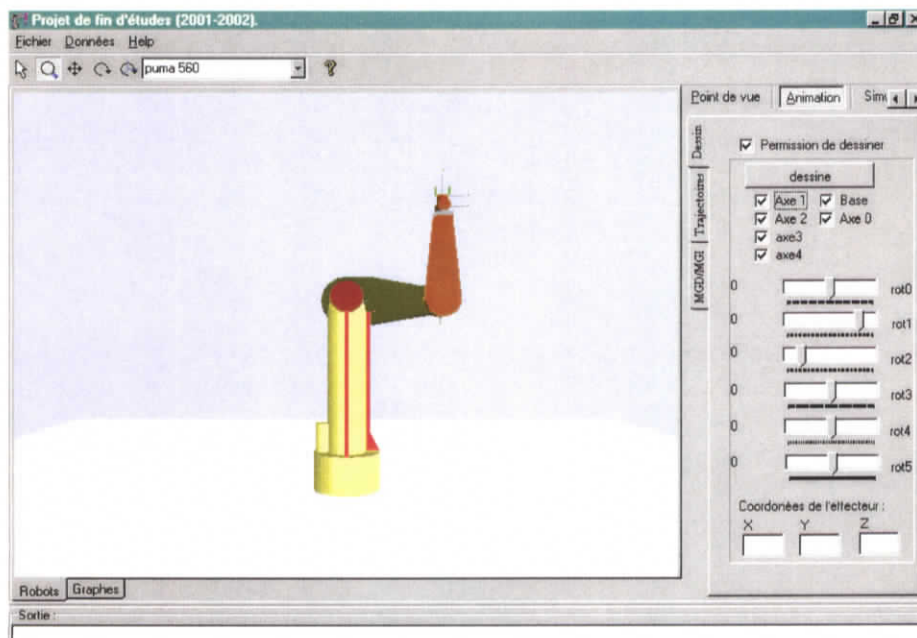


Figure 4.14 : La palette Dessin initialement pour le robot Puma 560.

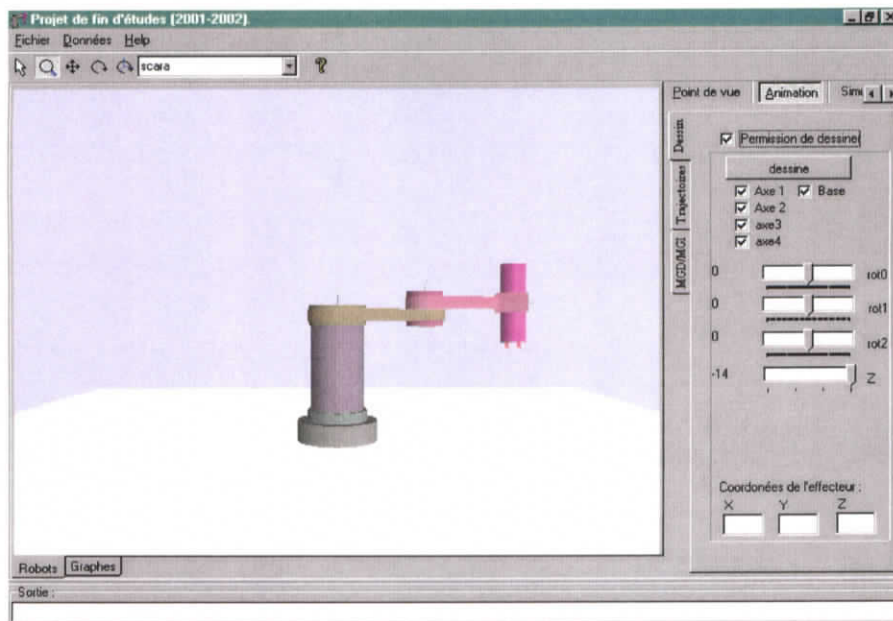


Figure 4.15 : La palette Dessin initialement pour le robot Scara.

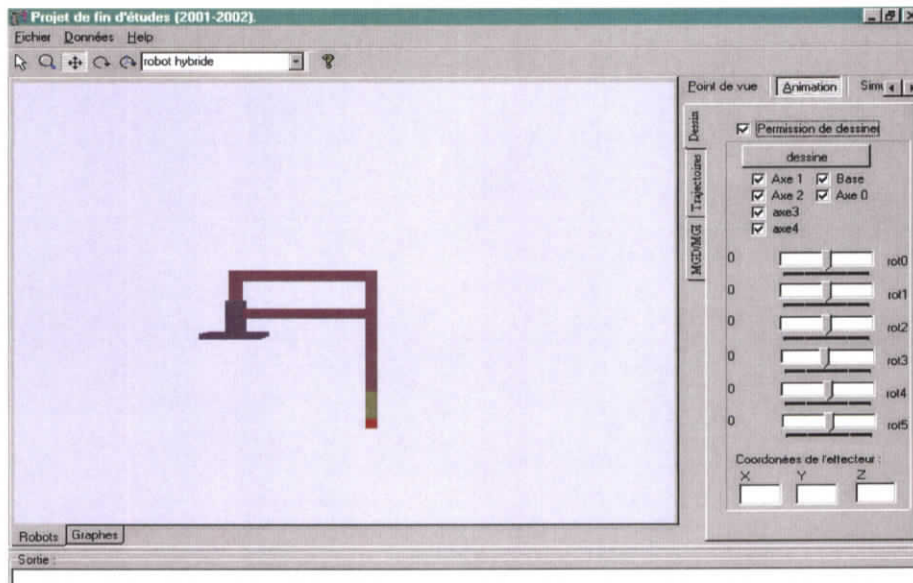


Figure 4.16 : La palette Dessin initialement pour le robot Hybride.

Les figures 4.17, 4.18 et 4.19, permettent de connaître les valeurs des coordonnées généralisées de chaque robot pour une position donnée, ainsi que la position de l'effecteur dans l'espace. C'est à dire, que nous allons faire bouger chacune des articulations du robot choisi en utilisant les Trackbars. Les déplacements ou rotations seront affichés à côté et tout en bas de la palette nous retrouvons la position de l'effecteur dans l'espace.

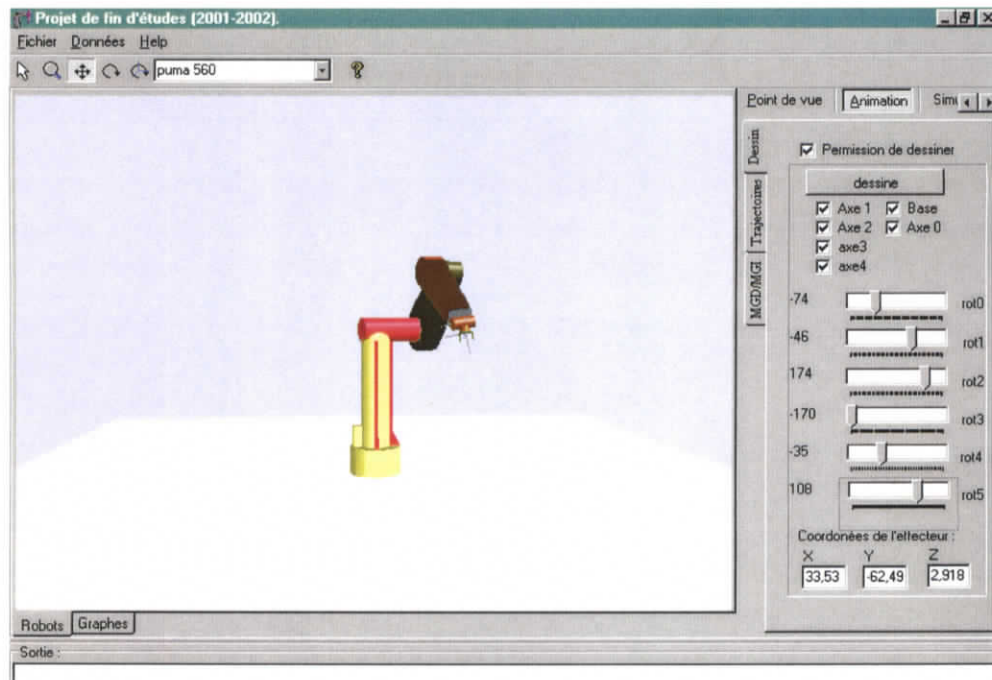


Figure 4.17 : Obtention des valeurs des coordonnées généralisées et la position de l'effecteur pour une position donnée du robot Puma 560.

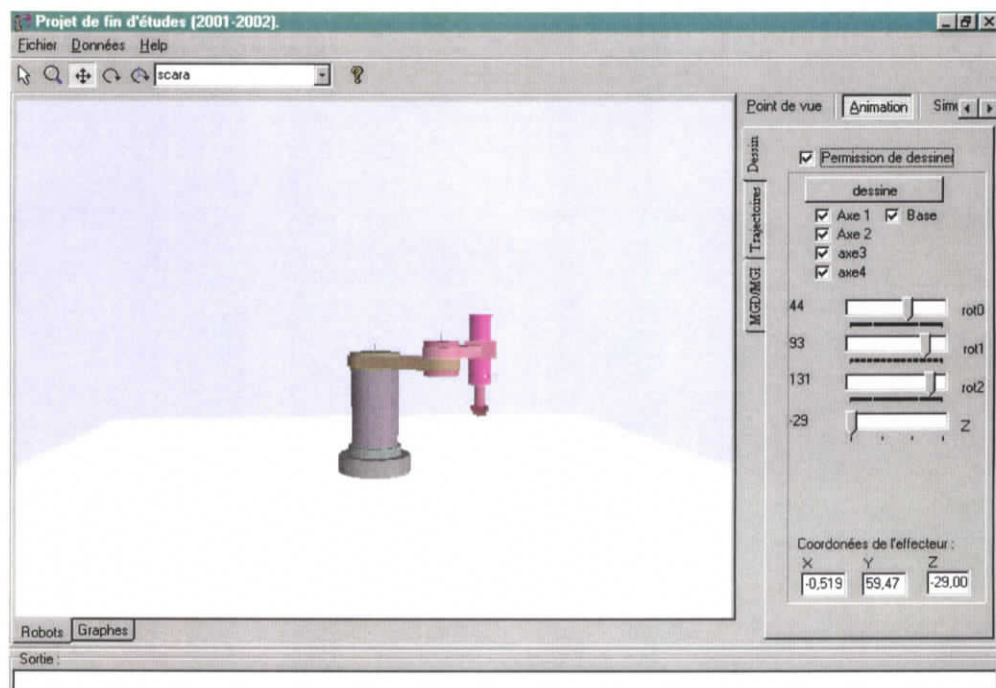


Figure 4.18 : Obtention des valeurs des coordonnées généralisées et la position de l'effecteur pour une position donnée du robot Scara.

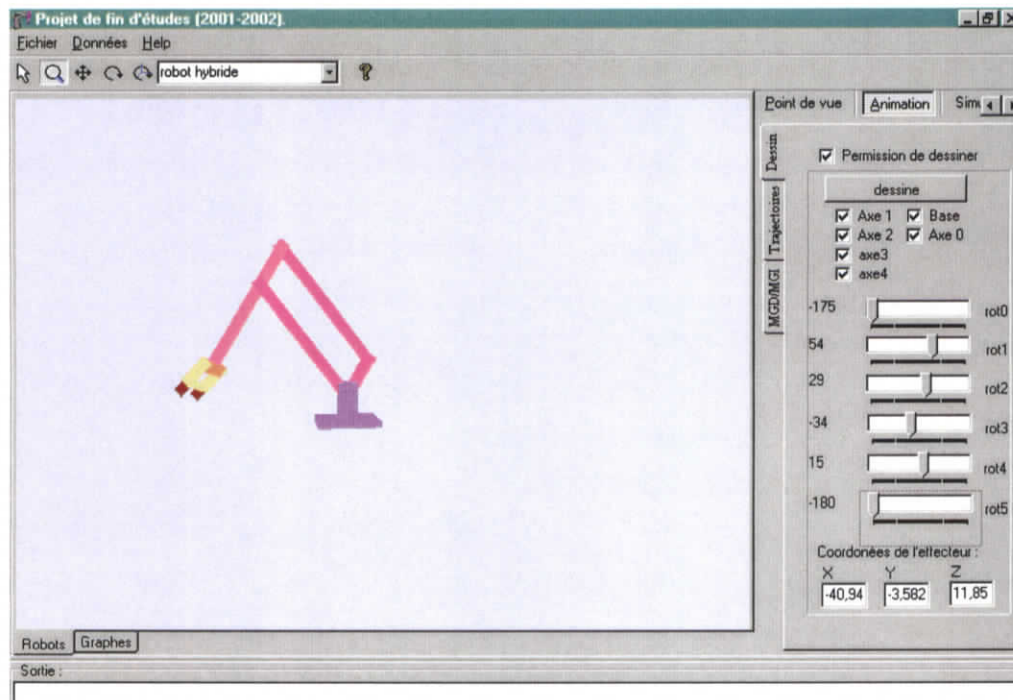


Figure 4.19 : Obtention des valeurs des coordonnées généralisées et la position de l'effecteur pour une position donnée du robot Hybride.

Nous passons maintenant, à la palette **Trajectoires** qui représente la palette secondaire numéro 2 de la palette **Animation**, dans cette dernière nous offrons deux types d'animation : une animation dans l'espace articulaire et une autre dans l'espace cartésien.

En ce qui concerne l'animation dans l'espace articulaire, l'utilisateur doit tout d'abord choisir le type de trajectoire, puis introduire les positions initiales et finales désirées pour chaque articulation du robot, préciser le temps du mouvement (la base de temps que nous avons choisi est de l'ordre de 0.1s, c'est à dire que si l'utilisateur introduit un temps de 50 ceci est égale à 5s en temps réel.), et enfin cliquer sur le bouton Animation pour animer le robot, si l'utilisateur introduit des paramètres qui représentent des butées mécaniques (c'est à dire des positions que le robot ne peut pas atteindre) un message d'erreur apparaît sur l'écran afin de prévenir l'utilisateur. Ceci stoppera l'animation du robot, mais l'utilisateur peut stopper l'animation du robot à tout instant et ceci en cliquant sur le bouton stop.

Le principe de la génération de trajectoire dans l'espace articulaire, repose sur l'échantillonnage de cette dernière (avec un pas d'échantillonnage égale à 0.1s), puis affectation des valeurs obtenues aux coordonnées généralisées du robot ceci en tenant compte des conditions initiales et finales ainsi que le temps de mouvement que l'utilisateur introduit à travers la palette **Trajectoires**. Nous proposons trois types de trajectoires : la trajectoire linéaire, la trajectoire polynomiale cubique et la trajectoire cycloïdale [20][29].

La figure 4.20 représente la palette **Trajectoires**. La figure 4.21 représente le robot Puma 560 dans une position initiale, et la figure 4.22 le représente dans une position finale, c'est à dire que le robot a effectué un déplacement selon une trajectoire (la trajectoire polynomiale cubique).

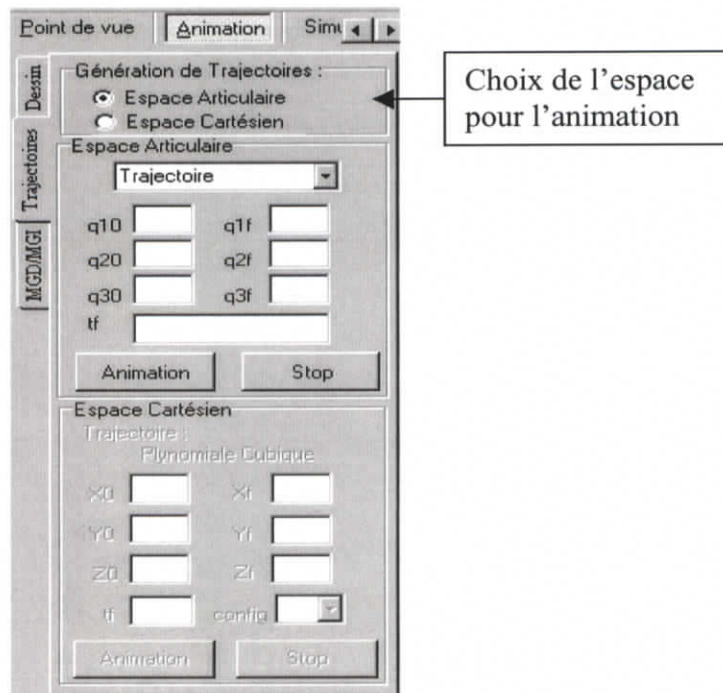


Figure 4.20 : La palette Trajectoires avec la sélection de l'espace articulaire.

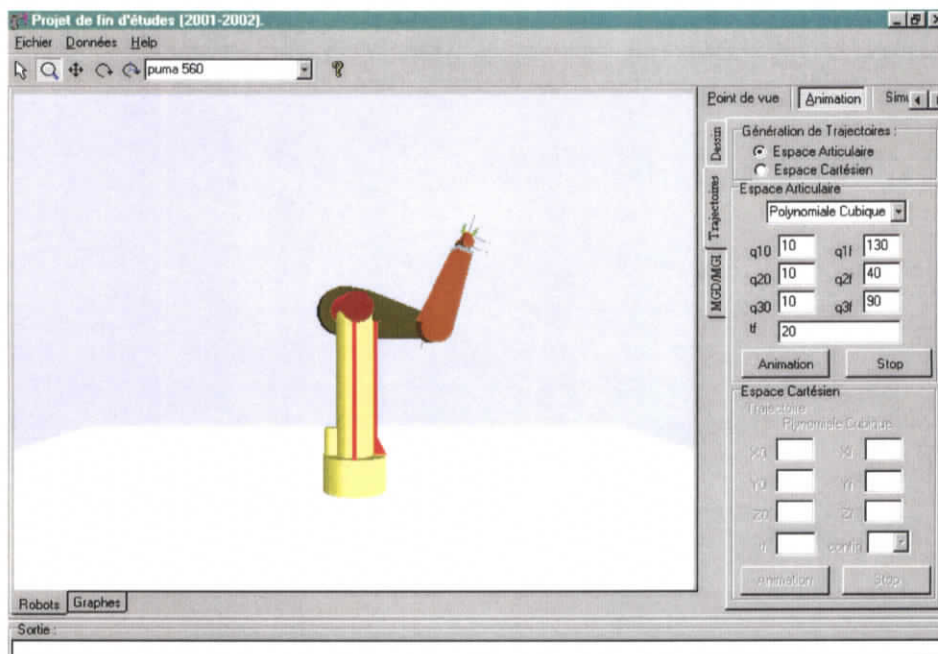


Figure 4.21 : Le robot Puma 560 dans une position initiale
 ($q_{10} = q_{20} = q_{30} = 10 \text{ deg}$).

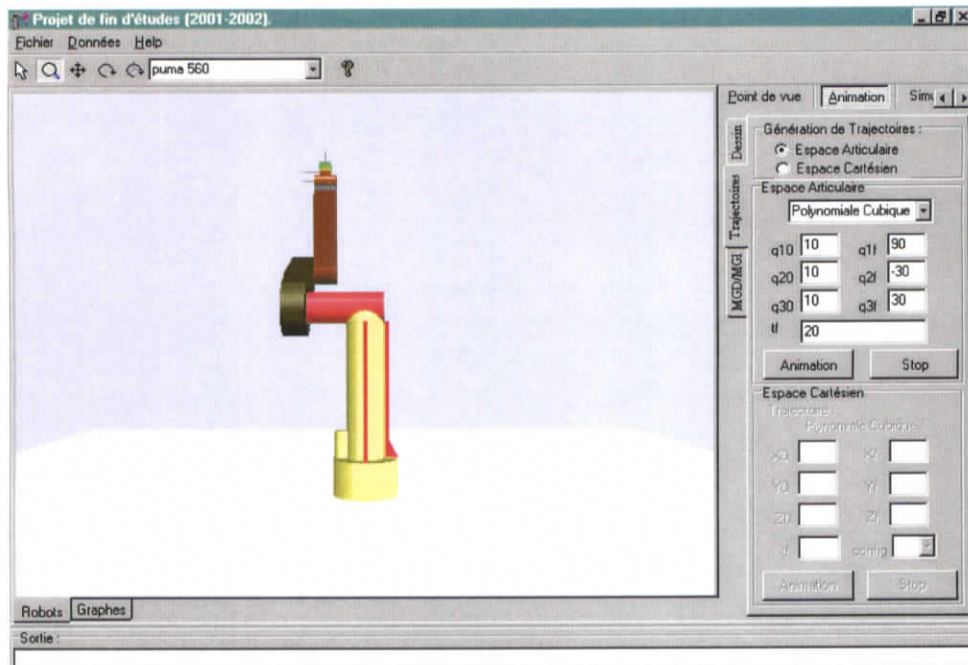


Figure 4.22 : Le robot Puma 560 dans une position finale
 $(q_{1f} = 90 \text{ deg}, q_{2f} = -30 \text{ deg}, q_{3f} = 30 \text{ deg})$.

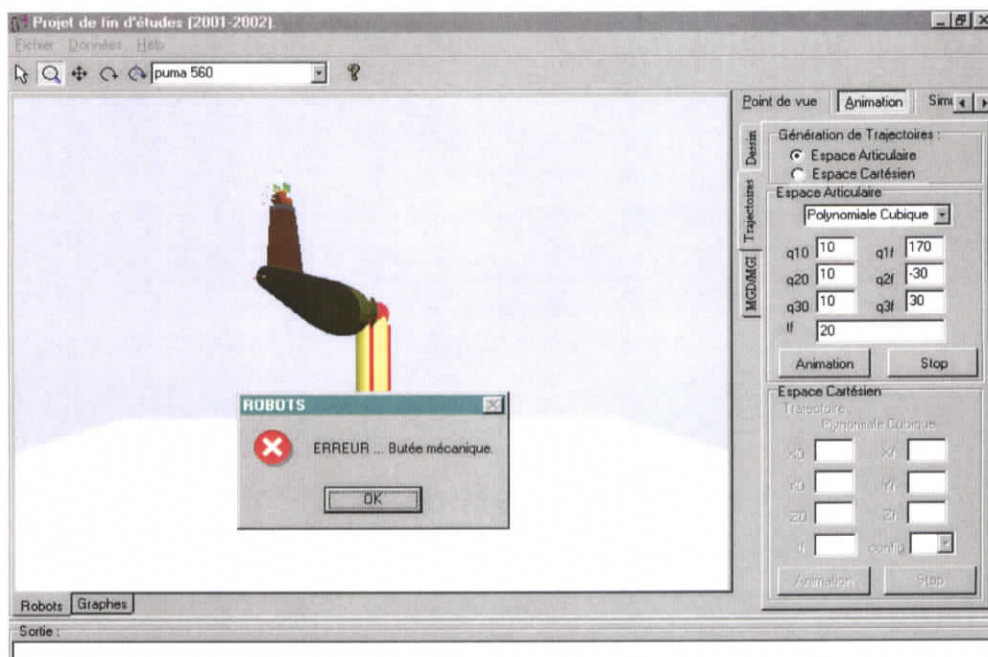


Figure 4.23 : Message d'erreur

Pour éviter le message d'erreur de la figure 4.23, l'utilisateur doit tenir compte des butées mécaniques pour chaque robot, ainsi il doit respecter les restrictions suivantes :

- **Pour le robot PUMA 560 [20],[21]**

$$\begin{aligned} -160 < q_1 < 160, \\ -225 < q_2 < 45, \\ -45 < q_3 < 225, \\ -170 < q_4 < 170, \\ -100 < q_5 < 100, \\ -266 < q_6 < 266. \end{aligned}$$

- **Pour le robot SCARA [22]**

$$\begin{aligned} -180 < q_1 < 180, \\ -150 < q_2 < 150, \\ -180 < q_3 < 180, \\ -29.24 < Z < -14. \end{aligned}$$

En ce qui concerne l'animation dans l'espace cartésien, l'utilisateur doit tout d'abord introduire les positions initiales et finales désirées pour chaque articulation du robot, préciser le temps du mouvement (la base de temps que nous avons choisi est de l'ordre de 0.1s) et choisir la configuration avec laquelle le robot atteindra sa position finale, enfin cliquer sur le bouton Animation pour animer le robot, si l'utilisateur introduit des paramètres qui représentent des butées mécaniques (c'est à dire des positions que le robot ne peut pas atteindre) un message d'erreur apparaît sur l'écran afin de le prévenir. Ceci stoppera l'animation du robot, mais l'utilisateur peut stopper l'animation du robot à tout instant en cliquant sur le bouton stop.

Le principe de la génération de trajectoire dans l'espace cartésien, repose sur l'échantillonnage de la trajectoire polynomiale cubique (avec un pas d'échantillonnage égal à 0.1s), puis nous utilisons le MGI pour le calcul des angles de rotation correspondant, et nous affectons les valeurs obtenues aux coordonnées généralisées du robot.

La figure 4.25 représente le robot Scara dans une position initiale, et la figure 4.26 le représente dans une position finale, c'est à dire que le robot a effectué un déplacement selon la trajectoire polynomiale cubique.

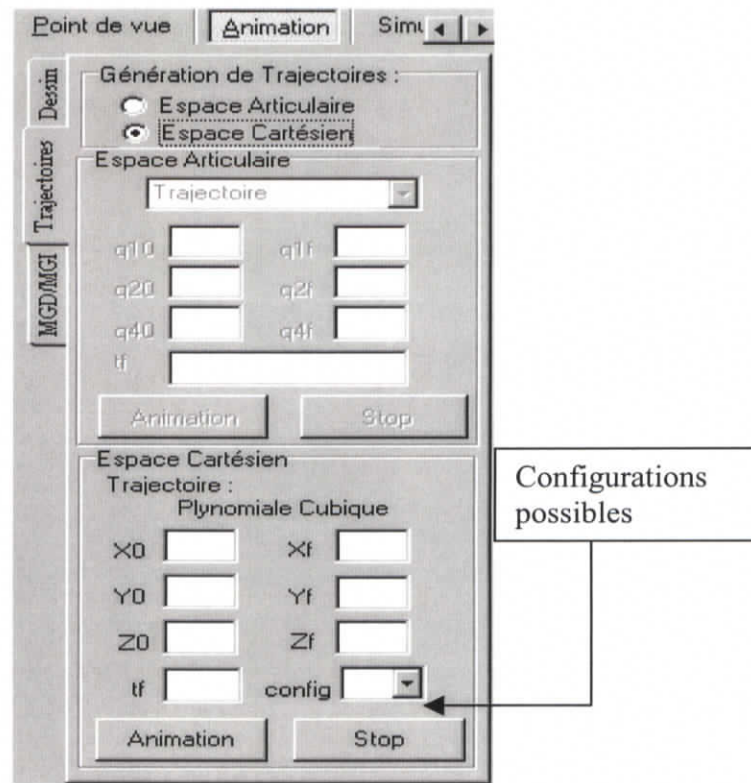


Figure 4.24 : Sélection de l'espace cartésien.

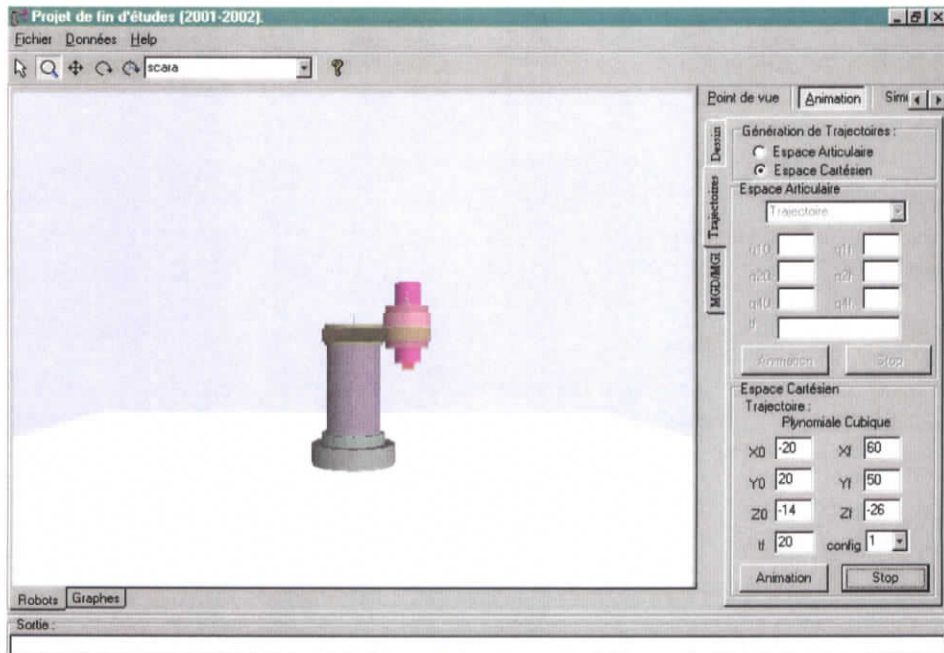


Figure 4.25 : Le robot Scara dans une position initiale
 $(X_{10} = -20, Y_{20} = 20, Z_{30} = -14)$.

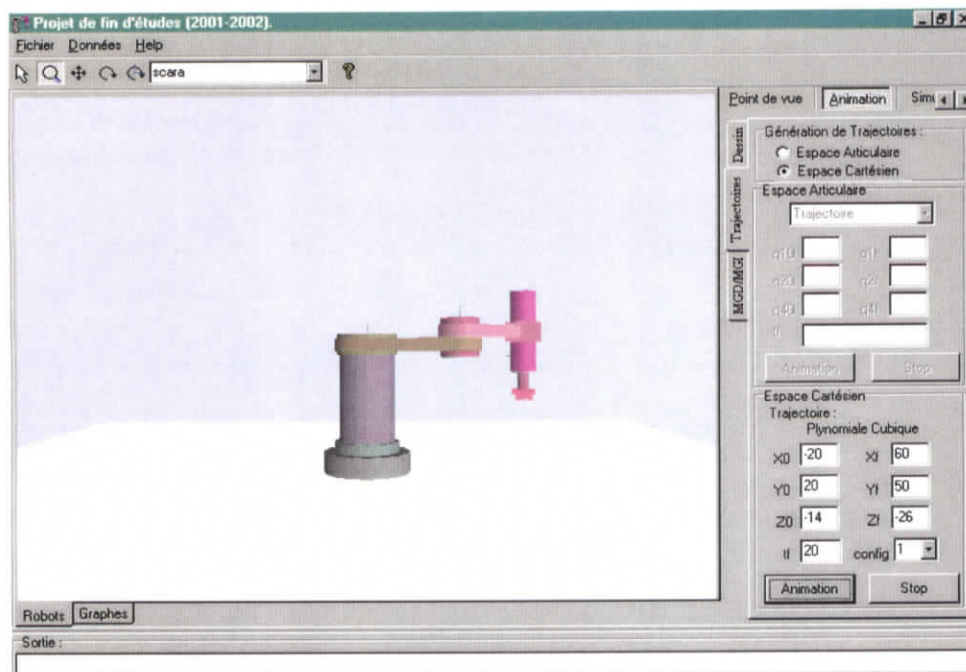


Figure 4.26 : Le robot Scara dans une position finale
 $(X_{1f} = 60, Y_{2f} = 50, Z_{3f} = -26)$.

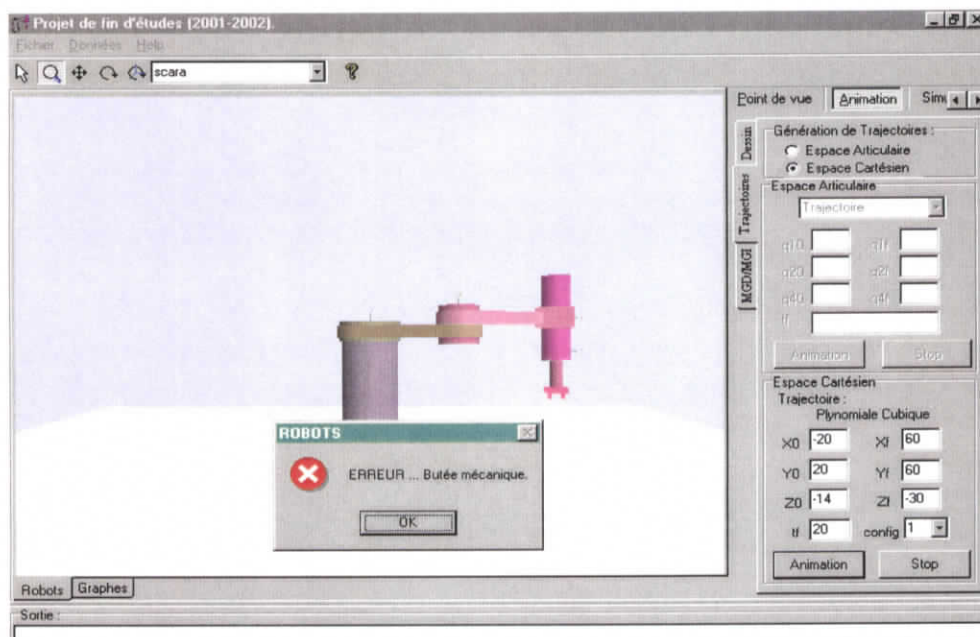


Figure 4.27 : Message d'erreur

Passons maintenant à la palette **MGD/MGI**, qui représente la palette secondaire numéro 3 de la palette **Animation**, elle permet à l'utilisateur de calculer la position de l'effecteur dans l'espace cartésien à partir des coordonnées articulaires et vice versa, mais aussi à travers cette palette l'utilisateur peut voir les différentes configurations possibles qui existent pour une même position de l'effecteur.

La figure 4.28 représente la palette **MGD/MGI**, et les figures 4.29, 4.30, 4.31 permettent de voir ce qu'offre cette palette.

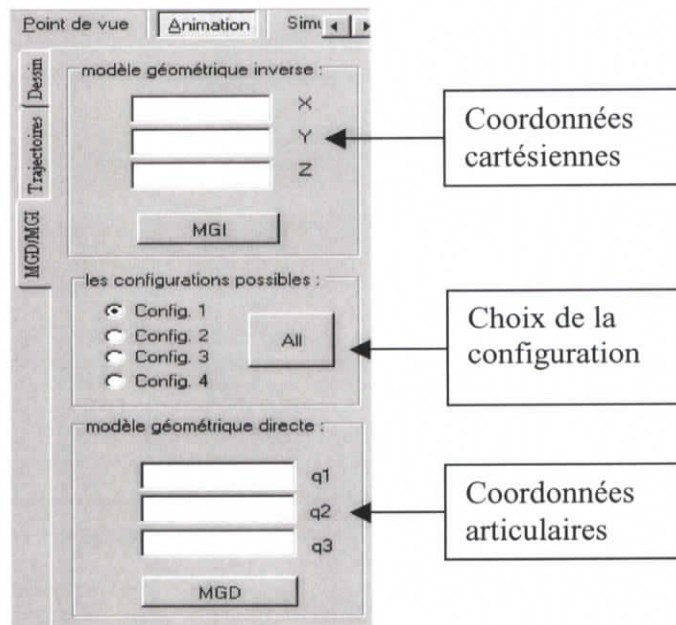


Figure 4.28 : La palette MGD/MGI.

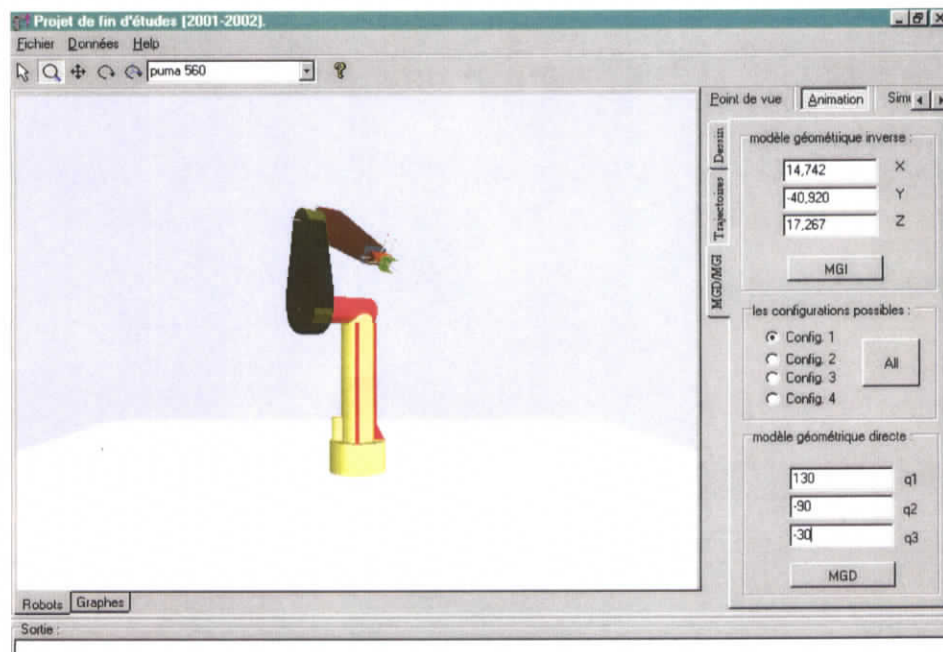


Figure 4.29 : Calcul des coordonnées cartésiennes en utilisant le MGD.

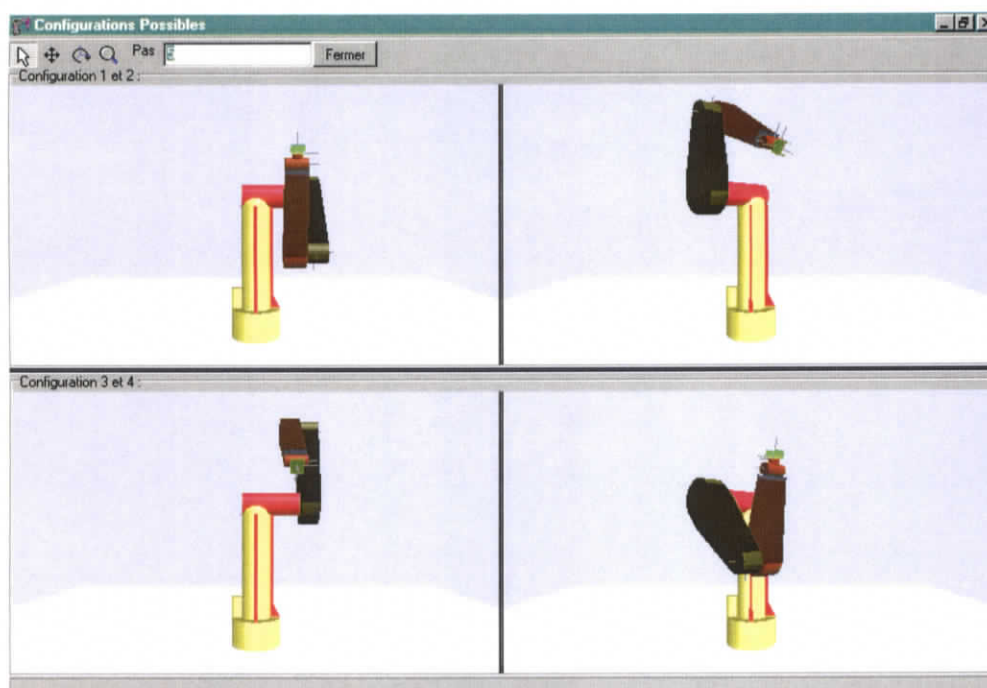


Figure 4.30 : les configurations possibles pour les même coordonnées cartésiennes.

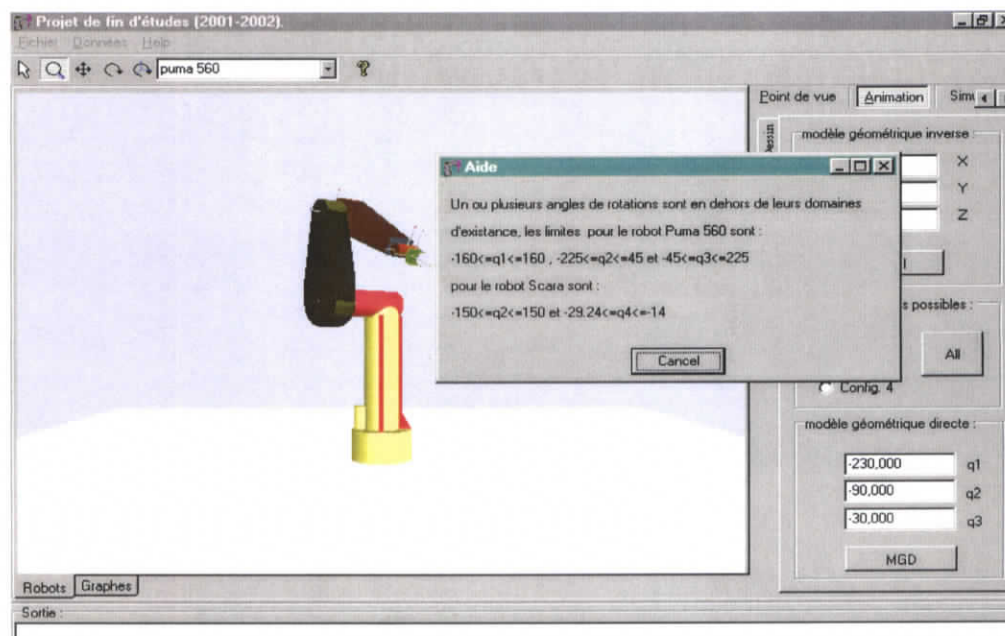


Figure 4.31 : Message spécifiant les configurations erronées.

Le principe de cette palette est simple, l'utilisateur introduit les coordonnées cartésiennes pour obtenir les coordonnées articulaires et ceci en choisissant une configuration et en appuyant sur le bouton MGI, comme il peut faire l'inverse, c'est à dire introduire les coordonnées articulaires pour obtenir les coordonnées cartésiennes et ceci en appuyant sur le bouton MGD.

En ce qui concerne le bouton **All**, ce dernier nous permet d'accéder à la fenêtre « **Configurations Possibles** » dans laquelle sont représentées toutes les configurations qui existent mais certaines d'entre elles peuvent être erronées.

Les figures 4.32, 4.33, 4.34 et 4.35 permettent de voir ce qu'offre la palette **MGD/MGI** mais cette fois pour le robot Scara et le robot Hybride.

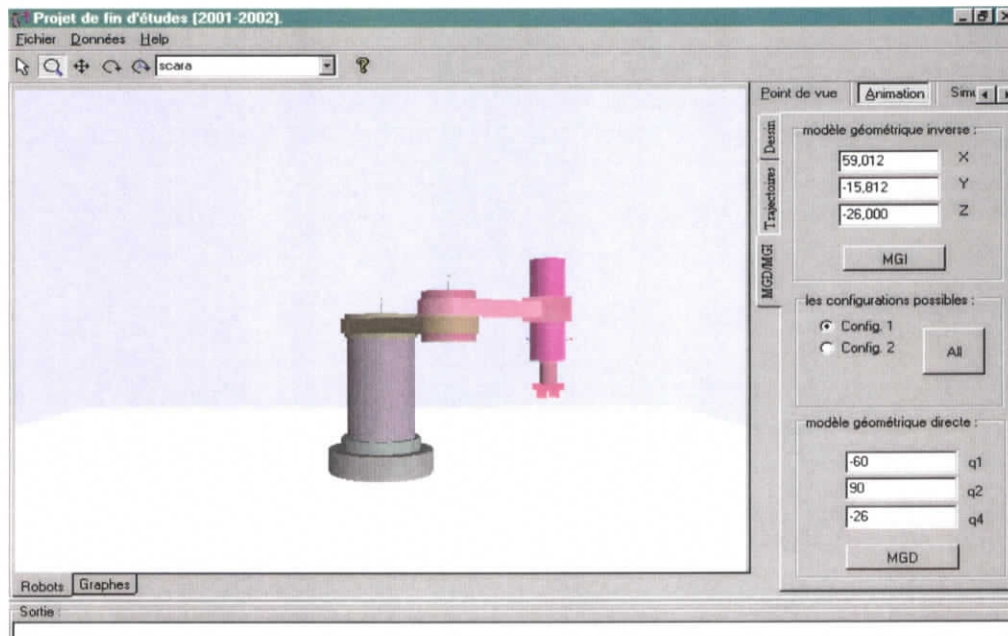


Figure 4.32 : Calcul des coordonnées cartésiennes en utilisant le MGD.

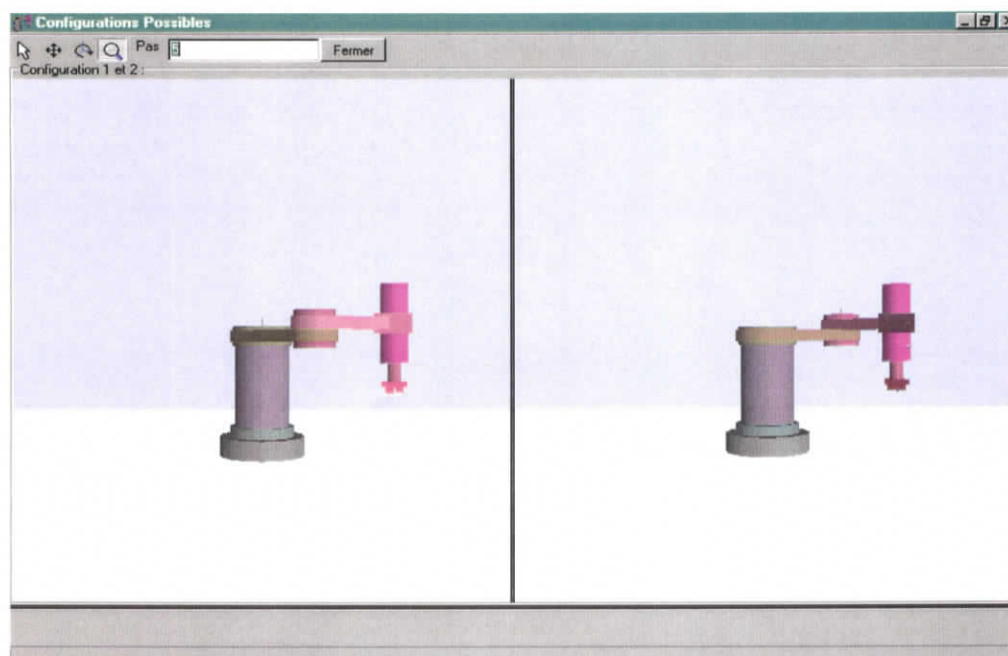


Figure 4.33 : les configurations possibles pour les même coordonnées cartésiennes.

Nous précisons que l'utilisateur dispose dans la fenêtre « **Configurations Possibles** » d'un menu , ce dernier comporte les différents types de curseurs déjà cités , alors l'utilisateur peut déplacer, faire tourner, pivoter le robot dans tous les sens.

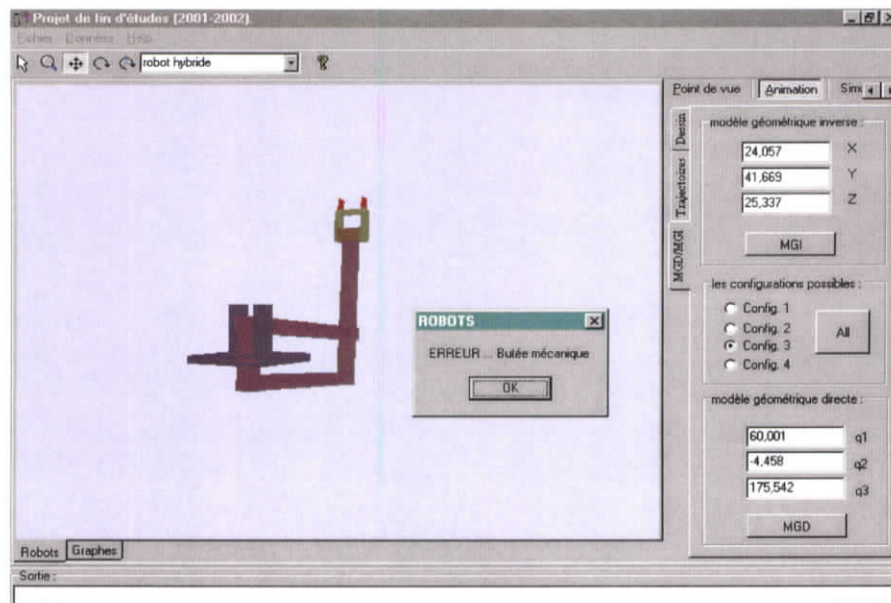


Figure 4.34 : Message spécifiant les configurations erronées.

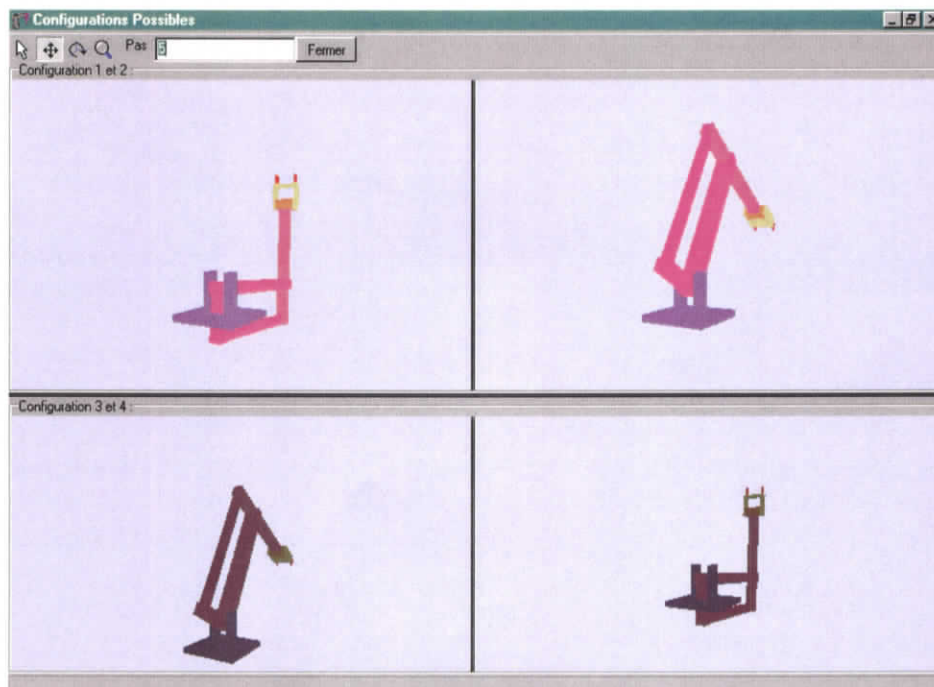


Figure 4.35 : les configurations possibles pour les même coordonnées cartésiennes.

Dans ce qui suit, nous allons parler de la palette **Simulation** à travers laquelle l'utilisateur a la possibilité de choisir le type d'actionneur et le type de trajectoire de référence (consigne). En effet cette palette comprend deux palettes secondaires : la palette **Actionneur** et la palette **Control**.

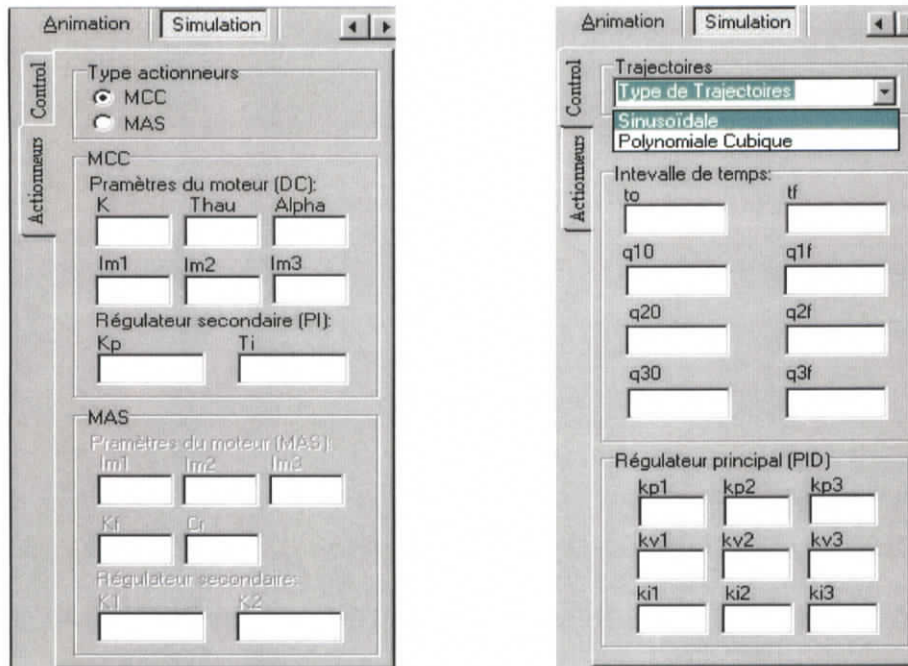


Figure 4.36 : Les palettes Actionneur et Control.

Avant d'appuyer sur le bouton **calcul**, l'utilisateur doit choisir le type d'actionneur (si un type d'actionneur est choisi l'autre est immédiatement désactivé), puis introduire les paramètres de ce dernier (si un des paramètres de l'actionneur manque lors de l'étape de calcul, un message apparaît sur l'écran précisant le paramètre manquant), ensuite il fera le choix de la trajectoire (si c'est la trajectoire polynomiale cubique, il doit faire introduire en plus de l'intervalle du temps, les conditions initiales et finales pour chaque articulation) et enfin les paramètres du régulateur PID.

Le bouton **anime**, sert à animer le robot en utilisant les valeurs des coordonnées généralisées calculées dans l'étape de calcul (quand nous appuyons sur le bouton **calcul**) car les résultats sont stockés dans un fichier. Durant cette étape (étape de l'animation du robot en tenant compte de sa dynamique et de l'influence de l'actionneur), l'utilisateur peut visualiser les résultats obtenus dans l'étape de calcul et ceci tout en faisant l'animation du robot, pour cela l'utilisateur doit utiliser soit la touche de raccourci F12 ou y accéder à travers le menu **Données**.

En ce qui concerne le bouton **pause**, son utilisation peut être à n'importe quel moment lors de l'animation, l'effet d'un clic sur ce bouton est de stopper l'animation du robot pendant un moment (durant ce laps de temps l'utilisateur peut faire bouger le robot ou utiliser le zoom s'il le souhaite afin de voir une configuration particulière du robot), et pour redémarrer l'animation du robot, l'utilisateur n'a qu'à appuyer de nouveau sur le bouton **anime**, ceci permet au robot de terminer sa trajectoire en partant de la dernière position où il s'est arrêté. En ce qui concerne le bouton **stop**, il sert à arrêter définitivement l'animation du robot.

Dans notre interface, nous présentons les résultats de calcul sous forme de graphes, l'interprétation des graphes a été faite dans le chapitre 3. Aussi, nous offrons à l'utilisateur la possibilité d'enregistrer les graphes dans un fichier sous le format « bitmat », les copier pour les utiliser dans un fichier quelconque ainsi que de les imprimer directement après sélection.

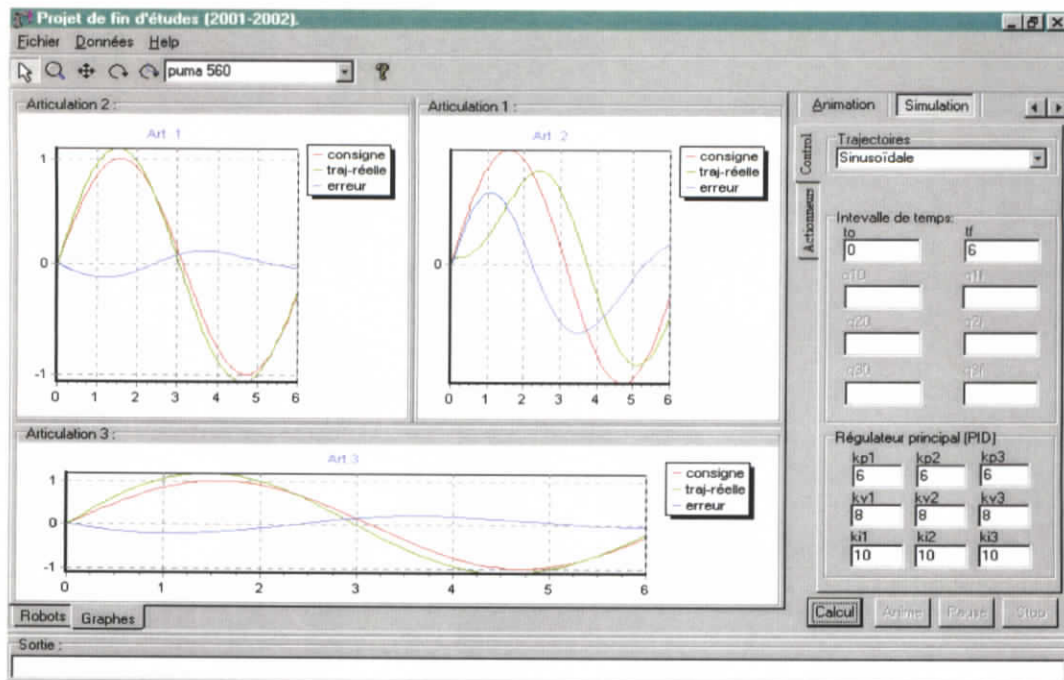


Figure 4.37 : Exemple de simulation du robot Puma 560 avec un actionneur de type moteur asynchrone (pour une trajectoire sinusoïdale).

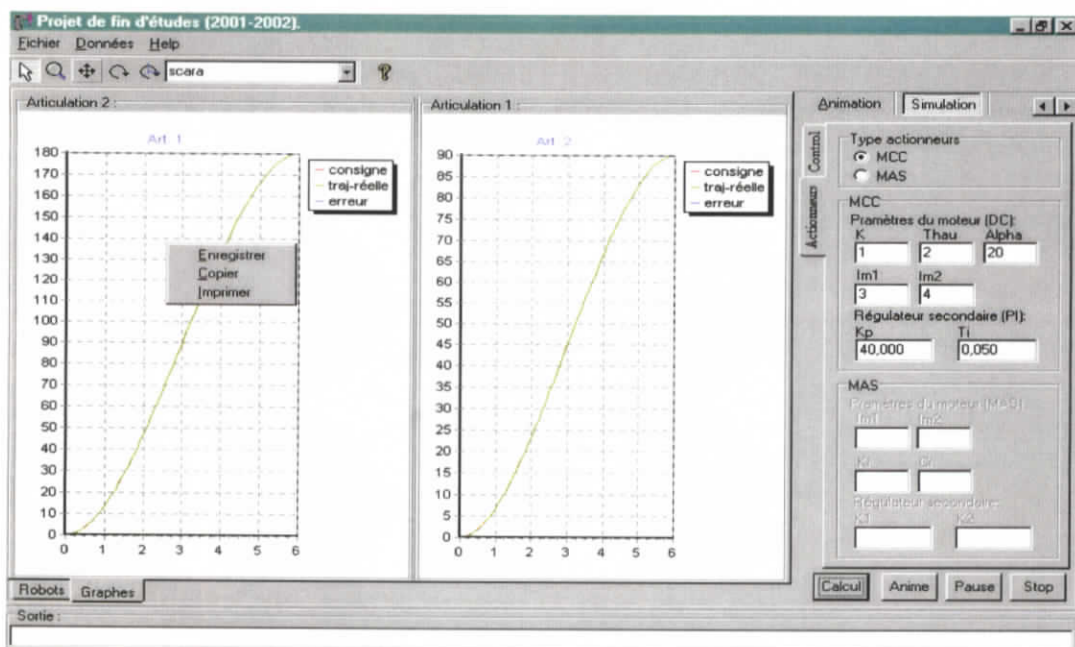


Figure 4.38 : Exemple de simulation du robot Scara avec un actionneur de type moteur à courant continu (pour une trajectoire polynomiale cubique).

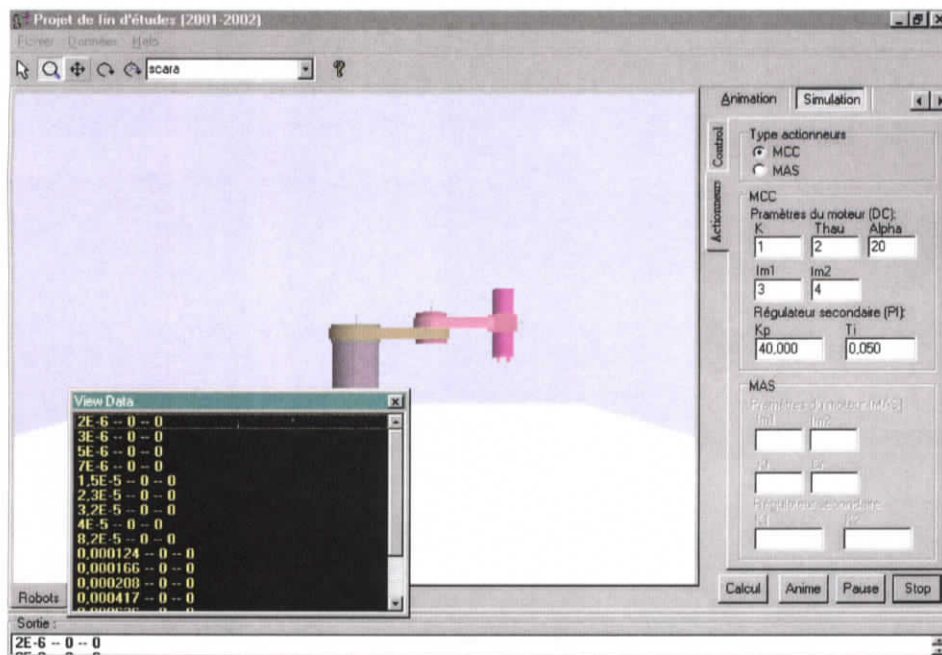


Figure 4.39 : Affichage des résultats de calcul lors de l'animation du robot Scara.

Enfin, à travers le menu Help nous pouvons faire appel au help de quelques fonctions utilisées ainsi qu'à la fenêtre About de notre interface.

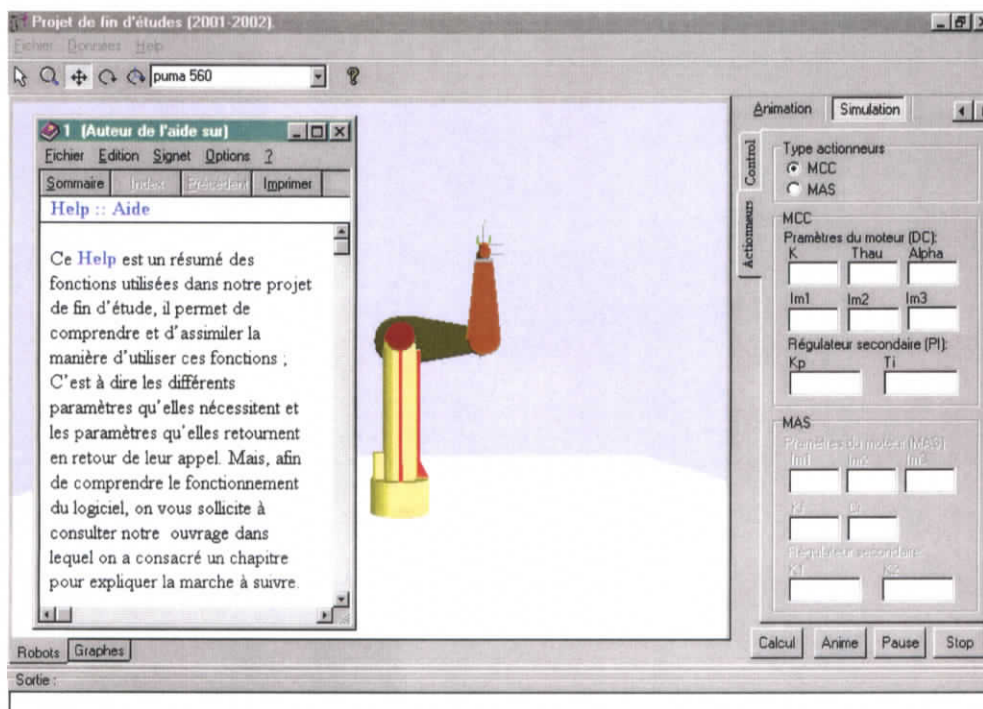


Figure 4.40 : Appel de l'Aide

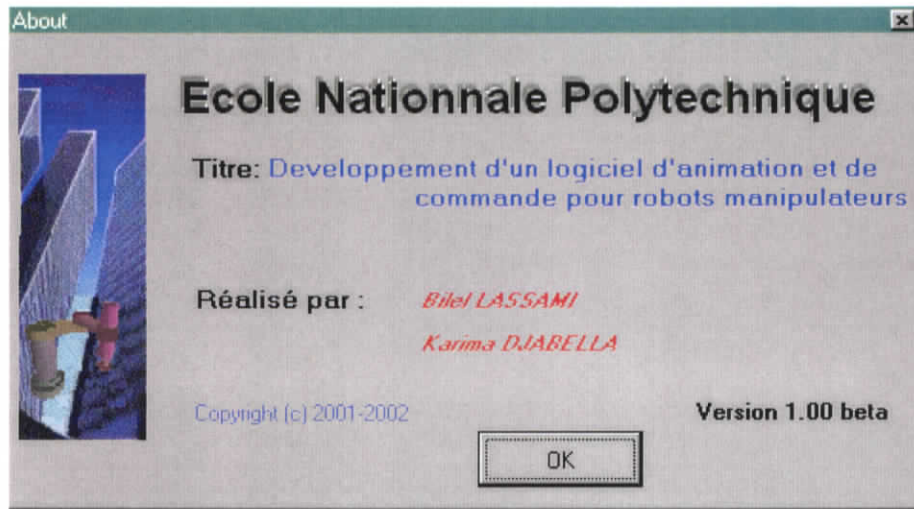


Figure 4.41 : About de l'interface.

Conclusion :

Dans ce chapitre, nous avons présenté notre interface en exposant les différentes palettes qu'elle comporte, et en expliquant les différentes possibilités qu'elle offre à l'utilisateur.

Cette interface est une interface graphique, qui permet de visualiser sur l'écran trois différents robots ainsi que leurs mouvement dans un espace 3D. Elle nous permet aussi le calcul des modèles géométriques directs et inverses pour les robots programmés, aussi elle nous offre deux types d'animation (animation dans l'espace articulaire et cartésien) en utilisant différents types de trajectoires. Et pour finir, cette interface permet la simulation de la commande Couple Calculé (la commande la plus utilisée dans l'industrie) en tenant compte de l'influence de deux types d'actionneurs (le moteur asynchrone et le moteur à courant continu), elle nous permet aussi la représentation des résultats de simulations sous forme de graphes et l'animation des robots en tenant compte de leurs dynamiques pour deux types de trajectoires (une trajectoire sinusoïdale de test et une autre polynomiale cubique).

Par conséquent, cette interface comporte trois volets importants qui sont: l'animation des robots programmés dans un espace 3D, le calcul des différents modèles géométriques et enfin la simulation de la commande Couple Calculé en tenant compte de l'influence de deux types d'actionneurs.

Chapitre 5

Interface de Commande

5.1 Introduction :

Les axes d'un système robotisé sont bien entendu mis en mouvement par des dispositifs d'entraînement qui convertissent en déplacement les signaux de commandes qui lui ont été fournis, parmi ces dispositifs, les moteurs asservis à courant continu, qui assurent une excellente régulation de vitesse, acceptent des couples élevés, et présentent un rendement excellent, c'est pour quoi ils conviennent particulièrement bien aux applications robotiques. Les moteurs à courant continu, peuvent être dimensionnés de façon à satisfaire une large gamme de puissances utiles, nous les trouvons en application dans la plupart des robots de petites et moyennes dimensions.

Dans ce chapitre, nous proposons un circuit d'interfaçage qui permet la commande et le contrôle d'un moteur à courant continu, et ceci à l'aide d'un microcontrôleur programmable de la firme Microchip.

5.2 Présentation de l'interface [51],[52],[53]:

Le microcontrôleur Picmicro® représente un choix idéal pour l'interfaçage des moteurs à courant continu. La famille du Picmicro® comporte une série d'interfaces, dispositifs et systèmes possédants une multitudes d'options qui permettent au concepteur un choix facile et de qualité.

De plus, la compatibilité entre plusieurs microcontrôleurs de cette famille tel que: les **PIC16Cxxx** et les **PIC18Cxxx**, offre une souplesse d'utilisation de plusieurs circuits (microcontrôleurs) pour la même interface, ce qui permet une migration facile et une aptitude pour d'autres performances ainsi que des changements futurs.

En particulier, nous proposons un schéma d'interfaçage du microcontrôleur **PIC16F877**, qui est aussi valable pour le **PIC18C452**, avec un moteur à courant continu. Nous allons détailler les circuits indispensables et les différentes ressources dont le microcontrôleur a besoin pour cette application.

L'unité centrale du microcontrôleur Picmicro offre une multitude de fonctions concernant la commande des moteurs à courent continu telle que :

- Contrôle de l'interface.
- Mesure de la position du moteur.
- Calcule du profil du mouvement (générateur de trajectoires).
- Calcule du signal d'erreur et implémentation de l'algorithme d'un régulateur PID.
- Génération du signal de commande du moteur.
- Communication avec une mémoire EEPROM non volatile.

Le schéma de l'interface est représenté par la figure 5.1. Ce dernier est extensible pour d'autres fonctions que le concepteur peut ajouter ou modifier selon les tâches et les performances désirées, il nécessite un moteur à courant continu doté d'un encodeur optique.

5.3 Partie Matériel (Hardware) [51],[52],[53]:

Le moteur à courant continu utilisé dans cet interface fait partie de la série 9200 de la firme **PITTMAN**, il est conçu pour une tension continu de 24 [volts], sa vitesse sans charge est de 6000 [RPM] (résolution par minute). La constante du couple de ce moteur est égale à 5.17 [oz-in/A] et la constante de sa force contre électromotrice est de 3.82 [V/KRPM]. Ce moteur comporte un encodeur optique interne d'une résolution égale à 500 [CPR] (incréments par tour).

Dans la pratique, cet interface est compatible avec tous les moteurs à courant continu munis d'un encodeur optique du même type.

Le schéma de cette application est représenté par la figure 5.1. Le moteur à courant continu est commandé par un circuit L6203 de la firme **SGS-THOMSON** (pont de commande en H) en utilisant des sorties DMOS qui peuvent délivrer un courant supérieur à 3 [Ampères] pour une alimentation de 52 [volts].

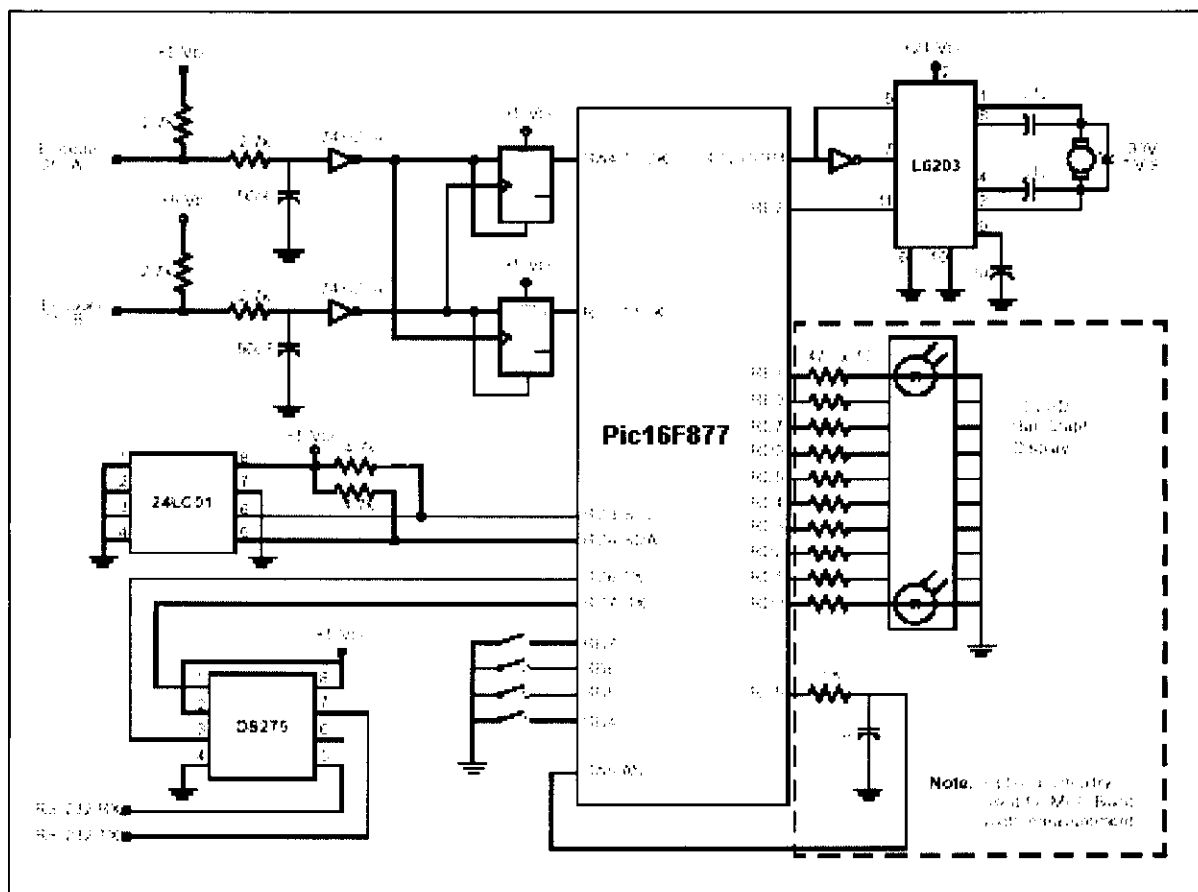


Figure 5.1 : schéma global du circuit de commande du moteur à courant continu.

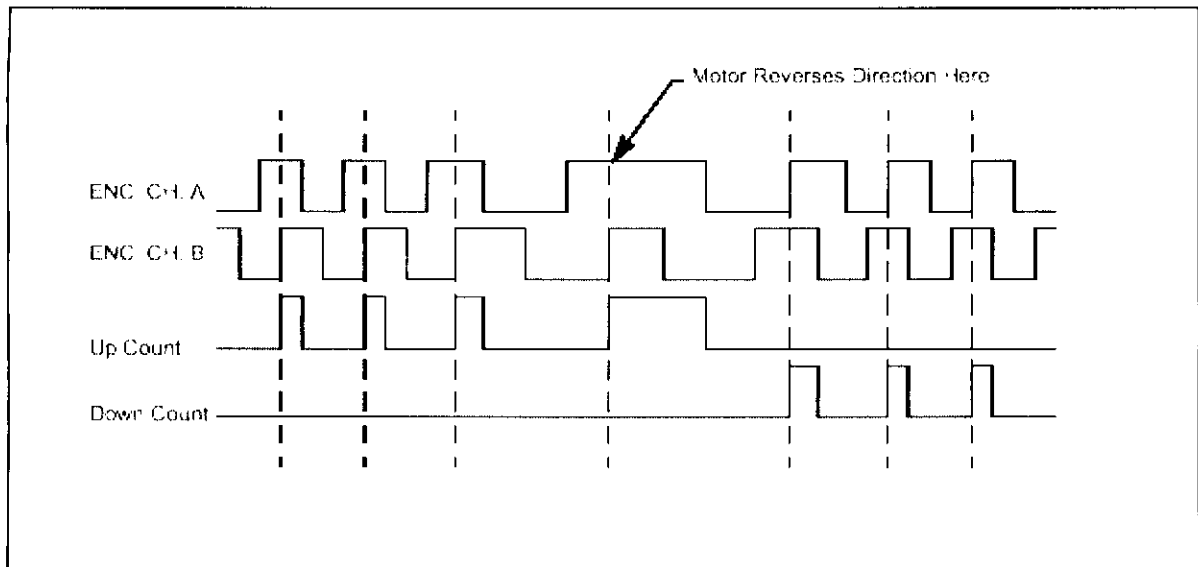


Figure 5.2 : Chronogramme du circuit de décodage.

Le circuit L6203 possède une pompe de charge interne « charge pump » pour commander les transistors du niveau haut « high-side transistor » et les circuits à temps mort. Chaque côté du pont H peut être commandé indépendamment avec des entrées compatibles à la technologie TTL.

Une entrée de permission « Enabled input » et un arrêt thermique sont intégrés dans ce circuit ainsi qu'un filtre des tensions transitoires placé aux bornes du moteur pour empêcher des dommages au circuit intégré L6203.

La broche « pin » en sortie notée PWM1 (pulse width modulation 1) de l'unité centrale du microcontrôleur PIC16F877, est connectée aux deux bornes (borne 5 et 7) du pont de commande en H. L'une de ces deux bornes reçoit le signal PWM inversé (borne 7). Le fonctionnement de ce circuit de commande avec cette connexion nous permet d'avoir moins de pertes de charges lors de la commutation des 4 interrupteurs au niveau du pont en H, car chacun de ces 4 interrupteurs est commuté pour chaque transition du signal PWM. Toute fois, cet arrangement fournit une méthode facile de commande bidirectionnelle à partir d'un seul signal d'entrée simple. Le dernier signal de contrôle qui reste, c'est un signal d'entrée (borne 11) de permission (Enabled input) qui met les sorties du circuit L6203 à 1 (activée) ou bien à 0 (désactivée).

Les deux signaux impulsionnels en sortie de l'encodeur sont en quadrature de phase et ils sont connectés à travers un circuit de résistances « pull up resistor », les sorties sont ensuite filtrées et décodées dans un train d'impulsion haut et bas avec le circuit 74HC74 à double bascule D « dual D flip-flop ». La figure 5.2 montre un chronogramme indiquant la sortie du circuit de décodage pour chaque direction du moteur.

Le circuit de décodage permet de générer une impulsion pour chaque transition des signaux de sorties de l'encodeur. Il est possible d'améliorer la précision de la position du moteur en utilisant un décodage qui fournit 4 impulsions pour une transition des signaux de l'encodeur. Cependant, le circuit de décodage d'une résolution pour une transition équivalente est implémentée dans cette application dans le but de simplifier le circuit (figure 5.1).

Les trains d'impulsions haut et bas à la sortie des bascules D sont connectées aux «Timer 0 » et « Timer 1 » respectivement (plus exactement aux bornes RA4/T0CK1 et RC0/T1CK1). Cette méthode de détermination de la position du moteur est bénéfique, parce qu'elle ne nécessite pas une lourde programmation. Les positions progressives ou régressives du moteur seront maintenues (mémorisées) à chaque instant par les «Timers 0/1 » pendant que l'unité centrale du microcontrôleur effectuera d'autres tâches.

La jonction RS-232 permet la connexion de cette interface au PC. L'émetteur-récepteur DS275 de la firme **Dallas Semi-Conductor** est utilisé dans cette conception, ce circuit englobe une communication de type half-duplex et il est alimenté par le PC pour générer la tension de transmission nécessaire pour la jonction RS232.

Quatre micro-interrupteurs « Dip switches » sont inclus dans le circuit, ils sont connectés au port B aux broches (RB4 –RB7) voir figure 5.1, ces micro-interrupteurs sont optionnels pour notre application et ils sont utilisés dans le programme pour assurer la génération de trajectoire quand l'interface n'est pas connectée au PC.

Les leds représentées dans le schéma sont utilisées pour l'implémentation d'un barre-graphe utilisé dans la microprogrammation pour indiquer le pourcentage de la bande passante utilisé par l'unité de calcul du microcontrôleur. Quand l'unité de calcul débute ces opérations la pin d'entrées/sorties RC5 est activée et elle est désactivée lorsque les calculs seront finis, le résultat en sortie de la pin RC5 est filtré pour créer une tension continue proportionnelle à la bande passante utilisée et il est transmis au canal 0 « channel 0 » du convertisseur analogique numérique (A/ D).

Les leds et le circuit filtre ne sont pas essentiels, et peuvent être supprimés de l'application si c'est désiré.

5.4 Partie Programmation (Software) [51],[52],[53]:

La programmation de l'interface pour la commande du moteur à courant continu permet le calcul de la position du moteur et elle fournit une interruption de commande pour créer et contrôler la génération d'une trajectoire donnée.

5.4.1 Calcul de la position du moteur :

Toutes les fonctions de contrôle du moteur sont implémentées dans la routine d'interruption « ISR » qui accomplit les tâches suivantes:

- Obtention de la position actuelle du moteur.
- Obtention de la position désirée du moteur.
- Calcul de l'erreur en position.
- Détermination du nouveau rapport cyclique du signal de commande MLI.

Le Timer 2, qui représente la base de temps du module CCP1 (générateur du signal MLI), est utilisé pour générer des interruptions à chaque calcul de la position du moteur. Cette propriété assure une synchronisation entre les variations du rapport cyclique et la période du signal MLI.

La fréquence du signal MLI qui commande le moteur doit être assez élevée afin que la valeur minimale du courant ondulé soit induite dans le bobinage du moteur. La valeur de ce courant dépend de la fréquence du signal MLI, de la résistance des enroulements et de l'inductance du moteur. Dans cette application, il est important de choisir la fréquence du signal MLI juste en dehors de la gamme des fréquences audibles (contrainte d'utilisation), c'est pour cela que nous avons choisi la plage de 15 à 20 [Khz] qui représente une bonne plage de travail.

Il n'est pas recommandé de travailler avec une fréquence très élevée du signal MLI car ça augmente les pertes de commutation au niveau du circuit L6203. Pour notre application, l'unité centrale du microcontrôleur fonctionne avec une vitesse d'horloge de 20 [Mhz] et un signal MLI d'une fréquence égale à 19.53 [KHz], à cette fréquence du signal MLI, une interruption provient du Timer 2 chaque 51 [μ secondes], cette interruption n'active pas la fonction de calcul de la position du moteur. Le « postscaler » du Timer 2 est utilisé pour gérer la vitesse des interruptions, il permet aussi de générer des interruptions à une fréquence variable de 1/2 à 1/16 de la fréquence du signal MLI.

5.4.2 La fonction Position « Updates » : (réinitialisation de la position)

La première tâche que le microcontrôleur doit effectuer est de déterminer exactement la position actuelle du moteur. La fonction UpdPos() est appelée pour retourner la nouvelle position du moteur (comme il a déjà été mentionner, le Timer 0 et le Timer 1 sont utilisés pour accumuler les impulsions hautes et basses produites par les 2 signaux en sorties de l'encodeur). Les contenus des compteurs ne sont jamais effacés pour éviter la possibilité de perte d'informations, parce que les valeurs des registres Timer 0 et Timer 1 sauvegardées pendant la précédente période d'échantillonnage seront retranchées aux présentes valeurs, en utilisant une opération arithmétique (complément à 2). Ce calcul nous fourni le nombre total des impulsions (hautes et basses) accumulées durant une nouvelle période de calcul de position du moteur.

Une opération arithmétique (complément à 2) ainsi qu'un test de débordement des Timers sont utilisés après chaque lecture de ces derniers. Le compteur de pulsations basses « DnCount » est soustrait au compteur de pulsations hautes « UpCount » ce qui produit un résultat signé indiquant la direction et la distance parcourue pendant une période de calculs. Ainsi, cette valeur représente la vitesse mesurée du moteur avec une unité : incrémentation par période de calcul du moteur et elle est stockée dans la variable « mvelocity ».

La position mesurée du moteur est sauvegardée dans une variable appelée « mposition ». Les 24 bits de poids fort du registre « mposition » fournissent la position du moteur en nombres

d'incréments de l'encodeur. Les 8 bits de poids faible de la variable « mposition » représentent la fraction du nombre d'incréments de l'encodeur.

La valeur du registre « mvelocity » est additionnée à celle du registre « mposition » pour trouver la nouvelle position du moteur.

Si nous voulons couvrir des grandes distances avec le moteur, la taille de la variable « mposition » doit être augmentée selon nos besoins.

5.4.3 La fonction « Trajectory Updates » : (rafraîchissement de la trajectoire)

Maintenant que nous connaissons la position du moteur, nous allons déterminer la position où il est supposé être, la position de commande (cosigne) est stockée dans une variable notée « position ». La taille de cette variable est de 32 bits tel que les 8 bits de poids faible représentent la partie fractionnelle des incréments de l'encodeur. Lorsque la valeur de la variable « position » est constante, l'arbre du moteur reste fixe et ne tourne pas.

Le microcontrôleur PIC16F877 permet de faire fonctionner le moteur pour une vitesse désirée, ceci se fait en additionnant une constante à la variable « position » à chaque mise à jour du moteur (calcul d'une nouvelle position). La partie fractionnelle de la variable position permet au moteur un fonctionnement à très petites vitesses.

Dans le but de générer un mouvement continu du moteur, nous devons formuler un algorithme qui contrôle la vitesse et l'accélération. Dans le contexte de cette application, nous allons contrôler la variation de la variable « position ». La fonction UpdTraj() est chargée de la génération de trajectoire et nous retourne les paramètres du profil de mouvement à chaque instant.

La fonction UpdTraj() nécessite la connaissance au préalable de la distance, la vitesse limite et l'accélération afin d'exécuter la trajectoire, et à partir des données qu'elle retourne l'arbre du moteur produira une courbe de vitesse sous forme trapézoïdale.

La figure 5.3 représente l'organigramme de la fonction UpdTraj(). Si la génération de trajectoires est activée et la sortie du signal MLI n'est pas saturée (ceci est indiqué par les indicateurs « stat.run » et « stat.saturated »), le générateur de trajectoire calculera la prochaine valeur de la variable « position ».

La figure 5.4 illustre l'algorithme de génération de trajectoires

FIGURE5.3: Organigramme de la fonction UpdTraj()

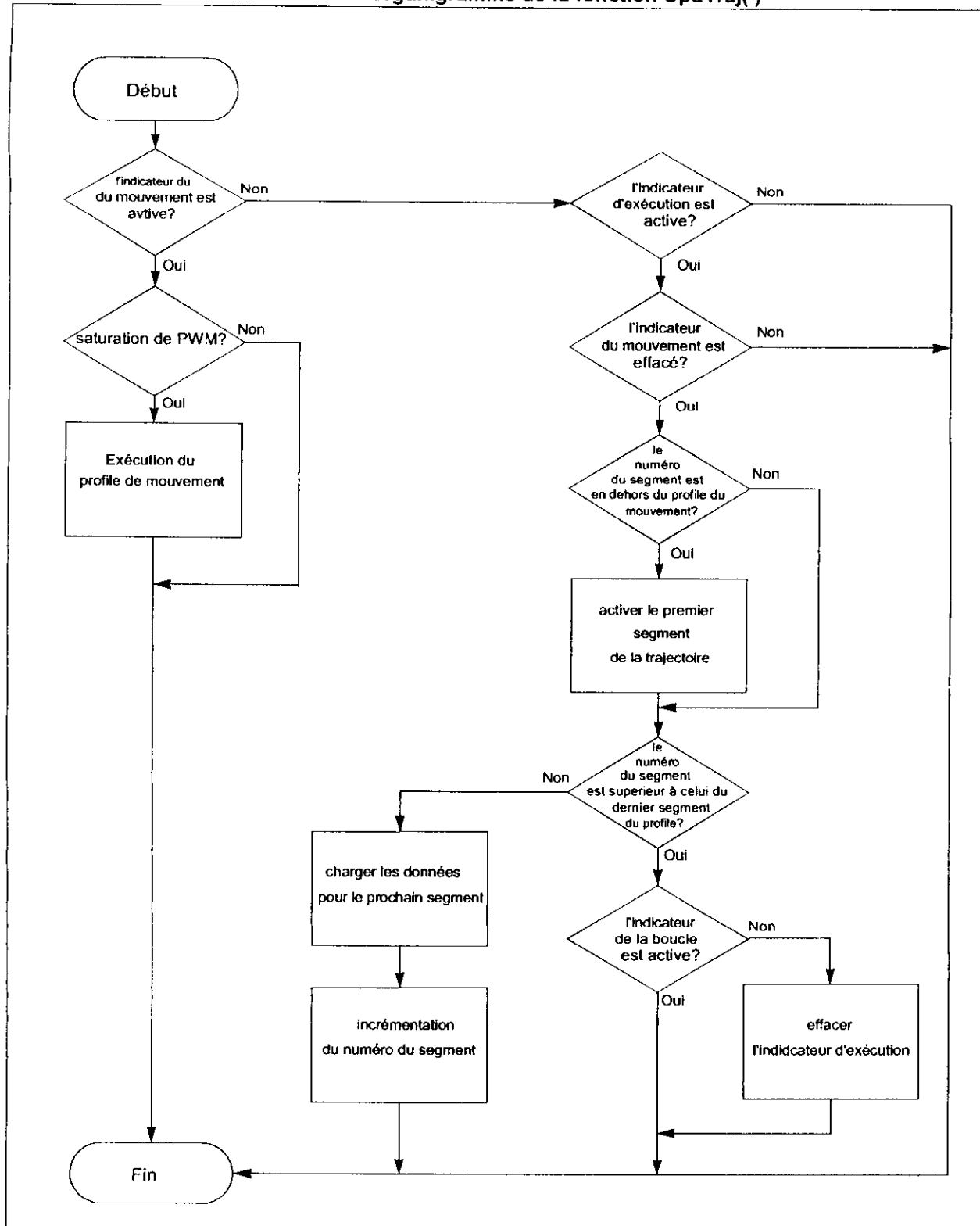
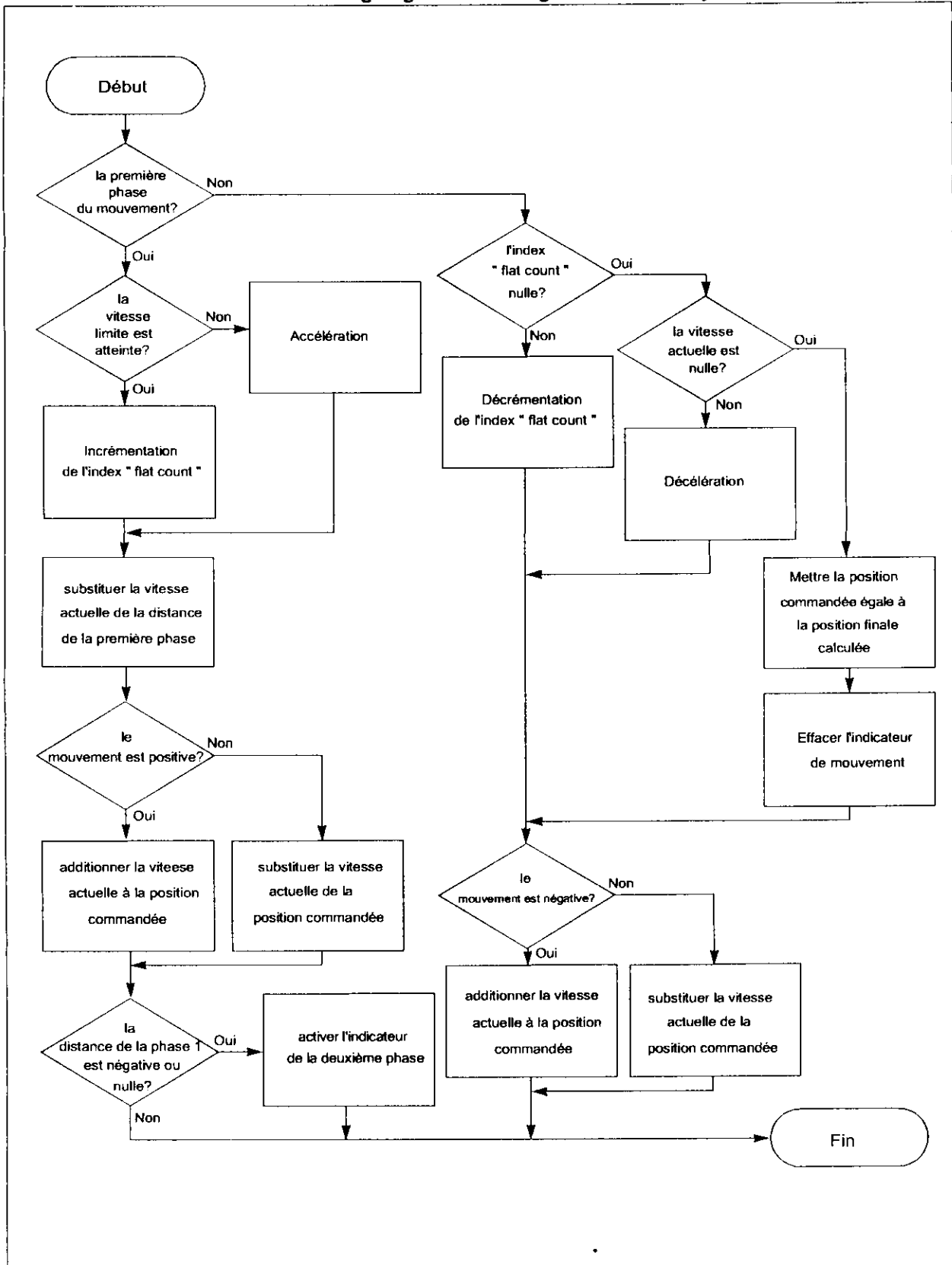


FIGURE 5.4: Organigramme de la génération de trajectoires



La génération de trajectoires se fait en deux phases ; dans la première phase la moitié de la trajectoire est parcourue, le reste durant la seconde. Les indicateurs ou paramètres nécessaires à la génération de mouvement sont les suivants :

- L'indicateur « stat.phase » ; indique la phase actuelle du segment de mouvement.
- La variable « phase1dist » ; indique la moitié de la distance à parcourir dans la phase 1.
- La variable « fposition » ; indique la position finale du moteur.
- La variable « vlim » ; contient la valeur de la vitesse limite lors de la génération de trajectoire.
- La variable « accel » ; la valeur de l'accélération pendant le mouvement y est sauvegardée.
- La variable « dtime » ; contient le paramètre temps de retard de la génération de trajectoires, cette variable représente le nombre de périodes pendant lesquelles le microcontrôleur s'arrête entre chaque exécution d'un segment de trajectoire.
- L'indicateur « stat.neg-move » ; indique la direction du mouvement.

Une fois que les paramètres de la génération de trajectoire sont chargés, l'indicateur « stat.motion » est activé et le mouvement commence pour un nouveau calcul de position du moteur, cet indicateur est effacé après la fin de la génération de trajectoire.

Le moteur peut être mis en marche à n'importe quelle vitesse désirée, par l'addition d'une valeur constante à la variable « position » à chaque étape de calculs, ceci forcera le moteur à prendre une nouvelle position.

La valeur ajoutée à la variable « position » est stockée dans une variable notée « velact », si on additionne ou on soustrait une valeur à la variable « velact » le moteur accélérera ou décélérera à chaque instant de calcul des paramètres du moteur (position, vitesse,...). La valeur d'accélération pendant la trajectoire est stockée dans une variable « accel », cette dernière est additionnée ou soustraite de la valeur de la position à commander « position » selon l'état de l'indicateur « stat.neg.move ». La valeur de « velact » est aussi soustraite à la variable « phase1dist » pour garder la trace de la distance parcourue dans la première partie du mouvement. Le moteur stoppera l'accélération quand « $velact \geq Vlim$ », et au moment où la vitesse limite est atteinte, la variable « flatCount » s'incrémente à chaque période de calcul pour compter le nombre de périodes dans lesquelles aucune accélération ne peut avoir lieu.

La première partie de la distance est complétée lorsque « phase1.dist » devient négative ou égale à zéro. A ce moment, l'indicateur « stat.phase » est mis à 1, la variable « flatcount » est ensuite décrémentée à chaque période. Quand la variable « flatcount=0 », le moteur commence la décélération. Le mouvement est généré complètement lorsque la vitesse de commande « velact » est nulle. La génération de trajectoire s'arrêtera pour une durée donnée par « dtime » (nombre de périodes échantillonnées) et à l'instant où $dtime=0$, la valeur de la variable fposition s'affichera pour la position commandée du moteur « position » et l'indicateur stat.motion est mis à zéro indiquant ainsi la fin de la génération de trajectoire.

Une que fois la génération de trajectoire est finie, la fonction UpdTraj() vérifie la présente valeur du segment du mouvement dans la variable « segnum » pour s'assurer s'il y a d'autres segments de mouvements à exécuter.

Le premier et le dernier segment à exécuter sont stockés dans « firstseg » et « lastseg » respectivement.

Si la valeur de « segnum » n'est pas égale à la valeur de « lastseg » alors « segum » est incrémentée et la fonction SetupMove() est appelée pour charger les nouveaux paramètres dans les variables de générateur de trajectoire.

5.4.4 Calcul des erreurs :

La fonction CalcError() soustrait la position mesurée du moteur « mposition » de la position à commander « position » pour donner la valeur de l'erreur en position.

L'erreur en position (sur 32 bits) résultant est décalée à droite et les 8 bits de poids faible qui représentaient la partie fractionnelle sont ignorés. Les 24 bits de l'erreur en position qui restent sont stockés dans la variable « u_0 » qui sont tronqués à une valeur signée de 16 bits pour les calculs ultérieurs.

5.4.5 Calcul du rapport cyclique :

La fonction CalcPID() implémente l'algorithme du compensateur proportionnel- intégral- dérivé (PID) et elle utilise les 16 bits de l'erreur en position stockés dans u_0 pour déterminer la prochaine valeur du rapport cyclique du signal MLI . Les constantes de gains k_p , k_i et k_d sont stockés dans des registres de 16 bits.

Le terme proportionnel de l'algorithme PID fournit une action directement liée à l'erreur u_0 (un terme proportionnel à l'erreur) pour la réponse du système.

Le terme intégral accumule les erreurs en positions successives calculées à chaque itération de la boucle de réglage, et améliore le gain de la boucle ouverte en basse fréquence. L'effet du terme intégral est d'annuler l'erreur en position du système.

Le terme différentiel de l'algorithme PID est fonction de la vitesse mesurée du moteur « mvelocity » son rôle est d'améliorer la réponse du système en boucle fermée dans la plage des hautes fréquences.

Après la sommation des trois termes de l'algorithme de réglage PID, le résultat est mémorisé dans une variable « ypid » ayant une taille de 32 bits, seuls les 24 bits de poids faible seront pris en compte. Les 16 bits de poids fort de la variable « ypid » sont utilisés pour régler le rapport cyclique du signal MLI qui divise la sortie de l'algorithme PID par 256. Puisque le module PWM possède 10 bits de résolution, la valeur des 16 bits de poids fort de la variable « ypid » est comparée avec 511 et -512, lorsque cette valeur dépasse ces limites, le rapport cyclique du signal PWM est pris égale à la limite maximale ou minimale et l'indicateur « stat.saturated » est mis à 1.

Lorsque la sortie du signal PWM devient saturée, la consigne « position » ne sera pas rafraîchie par la fonction UpdTraj() et l'accumulation du terme intégral dans l'algorithme PID est contournée. Ceci permet au moteur de terminer le mouvement d'une manière lisse une fois la saturation finie. Dans le cas où le terme intégral de l'erreur et la génération de trajectoire continueraient leurs évolutions, le moteur produira un mouvement irrégulier dû aux surcharges mécaniques.

5.4.6 Interrupteur de commande :

La partie programmation de cette application possède un interrupteur de commande qui permet de faire entrer les données des segments de la trajectoire à générer, exécuter une génération de trajectoire et changer les différents gains du régulateur PID.

Après l'initialisation des différents périphériques et les données dans la mémoire, la boucle principale du programme sélectionne l'USART par l'intermédiaire d'un indicateur pour la récupération des données sous le code ASCII. Chaque octet de donnée récupéré est mémorisé dans le vecteur « inbuf[] » sans aucune modification.

Une virgule ou un retour chariot «CR» est utilisé pour délimiter chaque séquence de commande. La fonction Do Command () est appelée dans les 2 cas pour déterminer la réponse correcte à ces deux chaînes de caractères (virgule / retour chariot). En outre, elle peut retourner la partie de la commande qui est au niveau du buffer ; représentée par un vecteur noté « inbuf[] » de taille 8 dans le programme, en utilisant la commande « comcount ». Le vecteur « inbuf[] » contient le nombre de virgules reçues à partir du dernier retour chariot « CR ». Les organigrammes de la commande par interruptions et la fonction Do Command () sont donnés par les figures 5.5 et 5.6 respectivement.

Le segment de trajectoire est mémorisé dans 2 vecteurs de valeurs entières dans la mémoire de donnée. Les données pour chaque segment du mouvement sont :

- La distance du déplacement.
- La valeur de l'accélération.
- La vitesse limite.
- Le temps de retard.

Le programme écrit permet l'introduction et la mémorisation de plus de 24 segments de trajectoire. Cependant, le nombre de segments peut être augmenté ou diminué selon la mémoire disponible dans l'application.

FIGURE 5.5: Organigramme de la commande par interruption

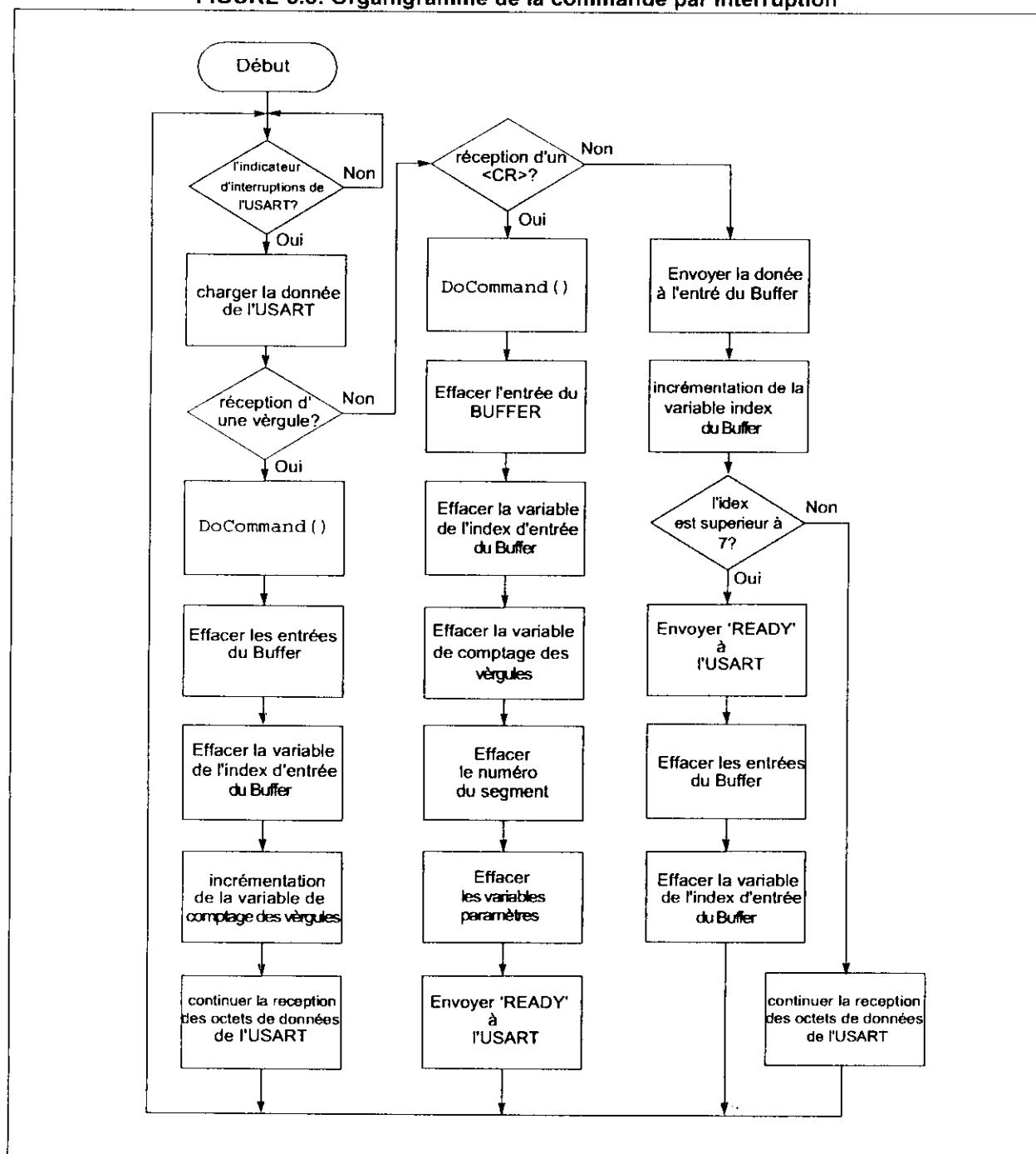
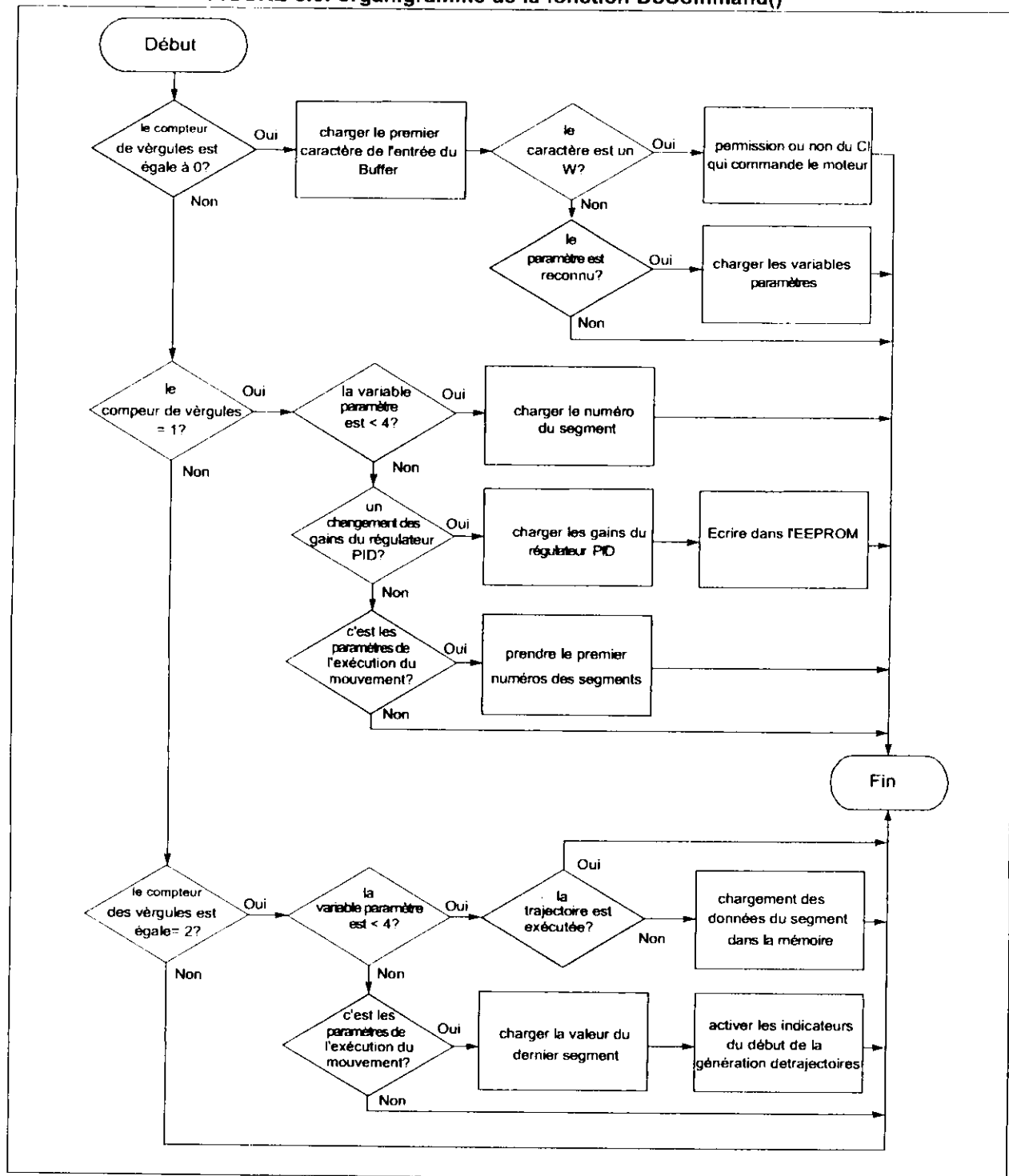


FIGURE 5.6: organigramme de la fonction DoCommand()



La chaîne de caractère ASCII qui permet le changement du profil de mouvement (les segments de mouvement) doit comporter les paramètres suivants :

- Le numéro du segment.
- La donnée à transférer.

Par exemple, supposons que nous voulons changer la valeur de l'accélération du deuxième segment à 1000. Pour se faire, il nous faut envoyer le code ASCII correspondant à la commande suivante :

A,1,1000 <CR>.

Cette syntaxe peut être utilisée pour changer tous les paramètres des segments de trajectoires.

Pour exécuter un ou plusieurs segments d'une trajectoire, nous utilisons l'une des commande « G » et « L ». Par exemple, l'exécution de segments de 1 à 4 pour une trajectoire se fait en écrivant le code ASCII correspondant à la commande suivante :

G,1,4 <CR> .

Si nous voulons exécuter un seul segment de mouvement, le nombre de segments désirés est entré 2 fois comme suit :

G,1,1 <CR>

La commande « L » est utilisée de la même manière que la commande « G » à l'exception de la répétition du mouvement pour chaque combinaison de segment, cette commande est utilisée pour créer des mouvement répétitifs du moteur.

La commande « S » stoppera la génération de trajectoire après la fin de l'exécution du segment de mouvement en cours.

Trois commandes sont disponibles pour changer les constantes de gain du régulateur PID. Avec ces commandes, nous pouvons manuellement fixer les paramètres de l'algorithme PID pour obtenir les meilleures performances du moteur.

La commande « W » permet de mettre le contrôleur du moteur (circuit L6203) en marche ou en arrêt. L'utilisation de cette commande est recommandée lors du réglage du moteur, dans le cas d'instabilité due au mauvais choix des paramètres k_p, k_v, k_i du régulateur PID.

La liste des commandes du servomoteur et leurs syntaxes sont données par la table 5.1.

La commande :	X, seg# , data <CR>
Cette commande permet de fixer la distance à parcourir pour chaque segment de la trajectoire. La donnée « data » est fournie sous forme d'un nombre d'incrément de l'encodeur relativement à la position actuelle. $0 \leq \text{seg\#} \leq 23$ $-32768 \leq \text{data} \leq 32767$	
La commande :	A, seg# , data <CR>
Cette commande permet de fixer l'accélération du moteur pour un segment donné de la trajectoire. La donnée « data » est fournie en (nombre d'incrément / (période d'échantillonnage) ² /65536). $0 \leq \text{seg\#} \leq 23$ $1 \leq \text{data} \leq 32767$	
La commande :	V, seg# , data <CR>
Cette commande permet de fixer la vitesse limite du moteur pour un segment donné de la trajectoire. La donnée « data » est fournie en (nombre d'incrément / période d'échantillonnage/256). $0 \leq \text{seg\#} \leq 23$ $1 \leq \text{data} \leq 32767$	
La commande :	T, seg# , data <CR>
Cette commande spécifie la durée de temps pendant laquelle le microcontrôleur s'arrête entre l'exécution de deux segments de trajectoire. La donnée « data » est un multiple de la période d'échantillonnage. $0 \leq \text{seg\#} \leq 23$ $0 \leq \text{data} \leq 32767$	
La commande :	G, startseg , stopseg <CR>
Cette commande permet l'exécution d'une série de segments par la spécification des segments initial et final $0 \leq \text{startseg} \leq 23$ $0 \leq \text{stopseg} \leq 23$	
La commande :	S , <CR>
Cette commande permet de stopper la génération de trajectoires	
La commande :	P , data <CR>
Cette commande permet de changer le gain proportionnel du régulateur PID. $-32768 \leq \text{data} \leq 32767$	
La commande :	I , data <CR>
Cette commande permet de changer le gain intégral du régulateur PID. $-32768 \leq \text{data} \leq 32767$	
La commande :	D , data <CR>
Cette commande permet de changer le gain différentiel du régulateur PID. $-32768 \leq \text{data} \leq 32767$	
La commande :	W <CR>
Cette commande permet d'activer ou désactiver l'étage pilote MLI	

Tableau 5.1: Sommaire des commandes du servomoteur.

Nous notons que deux indicateurs ; « stat.run » et « stat.loop » sont utilisés pour exécuter le profil du mouvement. Lorsque la commande « G » est utilisée pour exécuter une série de segments, l'indicateur « stat.run » est activé.

Lorsque la commande « L » est utilisée, les 2 indicateurs « stat.run » et « stat.loop » sont activés et quand la fonction Setup Move () détecte que le dernier segment dans le mouvement (la trajectoire désirée) a été exécuté, l'indicateur « stat.loop » est désactivé.

Si l'indicateur « stat.loop » est activé, les données du premier segment de la séquence sont chargées et exécutées continuellement et lorsque cet indicateur est mis à zéro (désactivé), l'indicateur « stat.run » est effacé et le mouvement s'arrête.

5.4.7 Manipulation avec le terminal ASCII :

Nous pouvons utiliser un micro-ordinateur équipé d'un logiciel tel que : PROCOMM ou HYPER Terminal pour assurer la programmation et la communication avec l'interface de commande du moteur à courant continu. La communication à travers le port série doit être configurée comme suit :

- Une vitesse de transmission de 19.2 k bauds.
- Aucune parité.
- 8 bits de données et un bit stop.

Lorsque le servomoteur n'est pas activé, nous devons avoir instantanément un message « Ready », dans ce cas, nous pouvons exécuter n'importe quelle commande du tableau 5.1.

5.5 Conclusion :

A travers ce chapitre, nous avons proposé une interface qui permet la commande d'un moteur à courant continu, cette interface nous permet d'établir la communication avec un ordinateur à travers la jonction RS 232. Elle est bâtie autour du microcontrôleur PIC16F877, que nous avons programmé avec le langage évolué C++ afin de commander et de positionner le moteur, ce programme (Annexe 4) est compilé par la suite avec le logiciel « HiTech PICC », le code source obtenu est transféré au microcontrôleur par l'intermédiaire d'une console de programmation.

Conclusion Générale

Conclusion Générale

Dans ce mémoire, nous nous sommes attachés à l'étude de quatre points fondamentaux de l'automatique, à savoir :

- La modélisation.
- La synthèse d'une commande.
- La simulation.
- Expérimentation.

Le premier point a été abordé en appliquant la convention de Devavit-Hartenberg pour la modélisation géométrique des trois robots. Le calcul du modèle géométrique inverse nous a permis la localisation de l'effecteur à n'importe quel point de l'espace de travail. Le formalisme d'Euler-Lagrange pour la modélisation dynamique des robots manipulateurs a permis de mettre en évidence la difficulté de synthétiser des commandes en robotique, étant donné les fortes non-linéarités qui régissent le fonctionnement dynamique de ceux-ci.

Nous avons appliqué une commande par découplage non-linéaire (couple calculé) qui a permis d'obtenir une simplification énorme dans la commande et des résultats assez satisfaisants. La prise en compte de la dynamique des actionneurs a été implémentée pour deux types de moteurs :

- Moteur à courant continu.
- Moteur asynchrone.

La partie simulation de ce travail a été faite sur un logiciel de simulation en robotique. Nous avons conçu ce logiciel en utilisant le langage de programmation Builder C++, nous avons aussi programmé plusieurs applications pour chaque type de moteur.

Le logiciel comprend des fonctions qui permettent le calcul des modèles géométriques et la génération de mouvements, que nous pouvons visualiser sur une page du logiciel dite « Robots ». La structure globale du logiciel est faite d'une manière modulaire pour permettre des changements éventuels faciles et rapides.

1- Modélisation du moteur asynchrone [36]:

La modélisation du moteur asynchrone est faite en admettant que soient vérifiées :

- L'additivité du flux ;
- La constance des inductances propres ;
- Les inductances mutuelles entre enroulements statoriques et rotoriques varient selon une loi sinusoïdale en fonction de l'angle électrique entre leurs axes magnétiques.

Ceci n'étant possible qu'en conséquences et moyennant certaines hypothèses.

On citera :

- L'entrefer constant ;
- L'influence de peau et de l'échauffement sur les caractéristiques sont négligeables ;
- Le circuit magnétique est non-saturé.

1-1 Equations électriques de la machine asynchrone :

En tenant compte de ces hypothèses, la machine est alors représentée par les équations des phases statoriques et rotoriques suivantes :

$$\begin{cases} [V_{sabc}] = [R_s] [i_{sabc}] + \frac{d[\phi_{sabc}]}{dt} \\ [V_{rabc}] = [R_r] [i_{rabc}] + \frac{d[\phi_{rabc}]}{dt} \end{cases} \quad (\text{An2-1})$$

Les expressions des flux en fonction des courants statoriques et rotoriques sont données par :

$$\begin{cases} [\phi_{sabc}] = [l_s] [i_{sabc}] + [M_{sr}] [i_{rabc}] \\ [\phi_{rabc}] = [l_r] [i_{rabc}] + [M_{sr}]^T [i_{sabc}] \end{cases} \quad (\text{An2-2})$$

avec :

$$R_x = \begin{bmatrix} R_x & 0 & 0 \\ 0 & R_x & 0 \\ 0 & 0 & R_x \end{bmatrix}, \quad l_x = \begin{bmatrix} l_x & M_x & M_x \\ M_x & l_x & M_x \\ M_x & M_x & l_x \end{bmatrix}, \quad x=s, r. \quad (\text{An2-3})$$

et

$$[M_{sr}] = \begin{bmatrix} \cos\theta & \cos(\theta + 2\pi/3) & \cos(\theta - 2\pi/3) \\ \cos(\theta - 2\pi/3) & \cos\theta & \cos(\theta + 2\pi/3) \\ \cos(\theta + 2\pi/3) & \cos(\theta - 2\pi/3) & \cos\theta \end{bmatrix} \quad (\text{An2-4})$$

Il est clair que l'écriture de $[V_s], [V_r]$ en fonction des courants conduit à un système d'équations dont les coefficients sont variables dans le temps, d'où la complexité de leur résolution.

Pour résoudre ce problème, nous avons eu recours à la transformation de Park qui s'impose alors comme alternative dans le but d'obtenir un modèle équivalent plus simple à manipuler.

1-2 Transformation de Park :

La transformation de Park fait projeter les enroulements d'axes a,b et c sur deux axes fictifs d et q en quadrature. On obtient une machine diphasée équivalente à la machine triphasée équilibrée. Cette transformation est définie par la matrice de Park :

$$P(\theta_x) = \sqrt{2/3} \begin{bmatrix} \cos(\theta_x) & \cos(\theta_x - 2\pi/3) & \cos(\theta_x + 2\pi/3) \\ -\sin(\theta_x) & -\sin(\theta_x - 2\pi/3) & -\sin(\theta_x + 2\pi/3) \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \quad (\text{An2-5})$$

Les composantes relatives et réelles sont liées par :

$$\begin{bmatrix} d & q & 0 \end{bmatrix}_x^T = P(\theta_x) \begin{bmatrix} a & b & c \end{bmatrix}_x^T, \quad x=s, r \quad (\text{An2-6})$$

1-3 Application de la transformation de Park à la machine asynchrone:

Une transformation de Park $P(\theta_s)$ et $P(\theta_r)$ appliquée au courant, tension et au flux, intervenant dans les équations (An2-1) et (An2-2) conduit au système d'équations à coefficients constants suivant :

$$\begin{cases} V_{dz} = R_z i_{dz} + \frac{d\phi_{dz}}{dt} - \frac{d\theta_z}{dt} \phi_{qz} \\ V_{dz} = R_z i_{dz} + \frac{d\phi_{dz}}{dt} - \frac{d\theta_z}{dt} \phi_{qz} \end{cases}, \quad z = s, r \quad (\text{An2-7})$$

$$\begin{cases} \phi_{ds} = L_s i_{ds} + M i_{dr} \\ \phi_{qs} = L_s i_{qs} + M i_{qr} \end{cases}; \quad \text{avec} \quad \begin{cases} L_s = l_s - M_s \\ M = 3M_{sr}/2 \end{cases} \quad (\text{An2-8})$$

$$\begin{cases} \phi_{dr} = L_r i_{dr} + M i_{ds} \\ \phi_{qr} = L_r i_{qr} + M i_{qs} \end{cases}; \quad \text{avec} \quad L_r = l_r + M_r \quad (\text{An2-9})$$

On pose : $\omega_s = \frac{d\theta_s}{dt}$, $\omega_r = \frac{d\theta_r}{dt}$ et $\omega = \frac{d\theta}{dt}$, de plus : $\omega_s - \omega_r = \omega = p\Omega$

La dernière relation montre que les vitesses angulaires du repère d-q par rapport aux repères statorique et rotorique liés rigidement à la vitesse angulaire du rotor Ω , ne sont pas des grandeurs indépendantes.

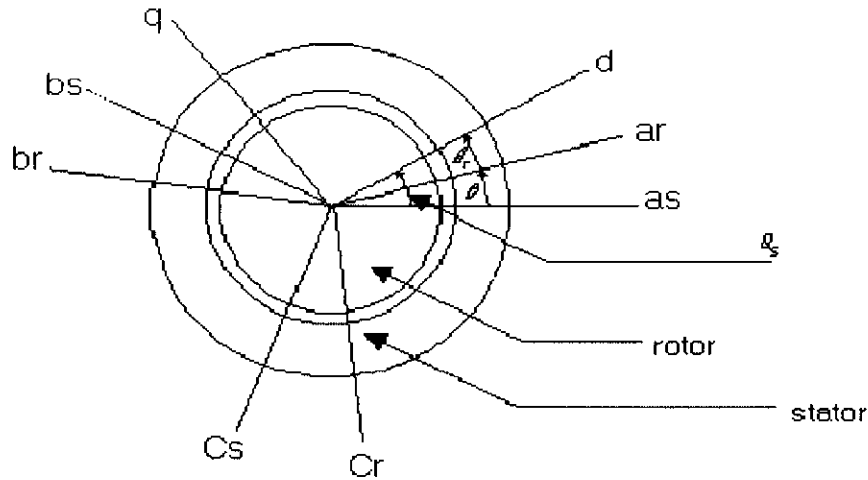


Figure A2.1 : Vue en coupe de la machine.

1-4 Objectifs de la commande et choix du vecteur d'état :

Le choix des sorties est d'une grande importance du fait qu'il est lié aux objectifs de la commande. Pour le modèle complet, la vitesse mécanique $\Omega = \omega / p$ est une variable d'état. Pour les quatre variables électriques, les plus courants sont :

$$x = (i_{ds}, i_{qs}, \phi_{dr}, \phi_{qr}, \omega)$$

$$x = (i_{ds}, i_{qs}, \phi_{ds}, \phi_{qs}, \omega)$$

$$x = (\phi_{dr}, \phi_{qr}, \phi_{ds}, \phi_{qs}, \omega)$$

$$x = (i_{ds}, i_{qs}, i_{dr}, i_{qr}, \omega)$$

Parmi ces choix possibles des variables d'état, nous optons pour les courants statoriques et le flux rotorique du fait qu'ils sont mesurables. Les tensions statoriques (V_{ds}, V_{qs}) et les vitesses du champ tournant (ω_s) sont les commandes du système.

1-5 Modèle de la machine dans le repère (d-q) :

En manipulant les équations précédentes et avec le choix du vecteur d'état $x = [i_{ds} \ i_{qs} \ \phi_{dr} \ \phi_{qr} \ \omega]^T$ et du vecteur de commande $[V_{ds} \ V_{qs}]^T$, nous aboutissons au modèle de la machine asynchrone alimentée en tension dans un repère (d-q) tournant à la vitesse synchrone ω_s .

$$\begin{cases} \frac{di_{ds}}{dt} = -\left(\frac{1}{\sigma \times T_s} + \frac{1-\sigma}{\sigma \times T_r}\right) \times i_{ds} + \omega_s \times i_{qs} + \frac{1-\sigma}{\sigma \times M \times T_r} \phi_{dr} + \frac{1-\sigma}{\sigma \times M} \omega \times \phi_{qr} + \frac{1}{\sigma \times L_s} V_{ds} \\ \frac{di_{qs}}{dt} = -\left(\frac{1}{\sigma \times T_s} + \frac{1-\sigma}{\sigma \times T_r}\right) \times i_{qs} + \omega_s \times i_{ds} + \frac{1-\sigma}{\sigma \times M \times T_r} \phi_{qr} + \frac{1-\sigma}{\sigma \times M} \omega \times \phi_{dr} + \frac{1}{\sigma \times L_s} V_{qs} \end{cases} \quad (\text{An2-10})$$

$$\begin{cases} \frac{d\phi_{dr}}{dt} = \frac{M}{T_r} i_{ds} - \frac{1}{T_r} \phi_{dr} + (\omega_s - \omega) \phi_{qr} \\ \frac{d\phi_{qr}}{dt} = \frac{M}{T_r} i_{qs} - \frac{1}{T_r} \phi_{qr} + (\omega_s - \omega) \phi_{dr} \end{cases} \quad (\text{An2-11})$$

avec : $T_r = L_r / R_r$, $T_s = L_s / R_s$ et $\sigma = 1 - (M^2 / L_s L_r)$.

qui peut être sous forme condensée :

$$x = f(x) + g(x)U$$

en prenant comme vecteur d'état :

$$x = [i_{ds} \quad i_{qs} \quad \phi_{dr} \quad \phi_{qr}]^T \quad \text{et} \quad U = [V_{ds} \quad V_{qs}]^T$$

1-6 Equation mécanique :

Afin d'obtenir le modèle complet de la machine, il faut tenir compte de l'équation mécanique. Sur la base développement de l'équation de la puissance instantanée, on obtient l'expression du couple électromécanique :

$$C_{em} = \frac{pM}{L_r} (\phi_{dr} \times i_{qs} - \phi_{qr} \times i_{ds}) \quad (\text{An2-12})$$

d'autre part on dispose de l'expression :

$$C_{em} = J \frac{d(\omega / p)}{dt} + K_f (\omega / p) + C_r \quad (\text{An2-13})$$

on obtient la cinquième équation d'état suivante :

$$\frac{d\omega}{dt} = \frac{p^2 M}{J L_r} (\phi_{dr} \times i_{qs} - \phi_{qr} \times i_{ds}) - \frac{K_f}{J} \omega - \frac{p}{J} C_r \quad (\text{An2-14})$$

Où C_r représente le couple résistant.

On remarque, qu'un couplage apparaît entre les courants statoriques dans l'expression du couple électromagnétique. Par conséquent, le couple ne peut être contrôlé séparément du flux.

Référentiel lié au champ tournant avec orientation du flux rotorique :

Le choix convenable d'un référentiel mène à une expression du couple électromagnétique similaire à celle de la machine à courant continu à excitation séparée. Ce choix est celui lié au champ tournant. Il permet de contrôler le couple séparément du flux. Le flux peut être le flux statorique, rotorique ou magnétique.

Un référentiel lié au champ tournant se traduit par la relation :

- Orientation du flux statorique

$$\phi_{ds} = \phi_s, \phi_{qs} = 0 ;$$

- Orientation du flux magnétisant

$$\phi_{dM} = \phi_M, \phi_{qM} = 0 ;$$

- Orientation du flux rotorique

$$\phi_{dr} = \phi_r, \phi_{qr} = 0$$

Si le référentiel choisi est lié au champ tournant avec orientation du flux rotorique, le contrôle du flux est fait par le courant i_{ds} séparément de la vitesse par i_{qs} .

Une orientation parfaite du flux permet d'avoir :

$$\begin{cases} \frac{d\phi_{dr}}{dt} = \frac{M}{T_r} i_{ds} - \frac{1}{T_r} \phi_{dr} \\ 0 = \frac{M}{T_r} i_{qs} - (\omega_s - \omega) \phi_{dr} \end{cases} \quad (\text{An2-15})$$

L'expression de ω_s déduite est :

$$\omega_s = \omega + \frac{M}{T_r} \frac{i_{qs}}{\phi_r} \quad (\text{An2-16})$$

une meilleure orientation du flux est obtenue en utilisant l'expression suivante [36] :

$$\omega_s(2) = \omega + \alpha \times i_{qs} + \beta \times \phi_{qr} \quad (\text{An2-17})$$

où : α et β sont des constantes à déterminer.

1-7 Equation d'état :

A partir des équations (An2-10), (An2-11) et (An2-14) et avec la notation suivante :

$$x = (x_1, x_2, x_3, x_4, x_5) = (i_{ds}, i_{qs}, \phi_{dr}, \phi_{qr}, \omega)$$

nous aboutissons au système d'équations avec grandeurs réduites suivant :

$$\begin{cases} \dot{x}_1 = -a_1 x_1 + b_1 x_3 + c_1 \omega_s x_2 + d_1 x_4 x_5 + e V_{ds} \\ \dot{x}_2 = -a_1 x_2 + b_1 x_4 - c_1 \omega_s x_1 - d_1 x_3 x_5 + e V_{qs} \\ \dot{x}_3 = -a_2 x_3 + b_2 x_1 + c_1 \omega_s x_4 - c_1 x_4 x_5 \\ \dot{x}_4 = -a_2 x_4 + b_2 x_2 - c_1 \omega_s x_3 + c_1 x_3 x_5 \\ \dot{x}_5 = -a_3 x_5 + d_3 x_2 x_3 - d_3 x_1 x_4 - f \cdot C_r \end{cases} \quad (\text{An2-18})$$

où :

$$\omega_s = \omega + \frac{M i_{qs}}{T_r \phi_r}$$

$$a_1 = (1/(\sigma \times T_s) + (1 - \sigma)/(\sigma \times T_r))$$

$$, a_2 = 1/T_r, \quad a_3 = K_f / j.$$

$$b_1 = ((1 - \sigma)/(\sigma \times M \times T_r)) \times (\phi_n / I_m)$$

$$, b_2 = M \times I_m / (T_r \times \phi_n)$$

$$d_1 = ((1 - \sigma)/(\sigma \times M)) \times (\phi_n \times \omega_n / I_m)$$

$$, d_3 = ((p \times p \times M)/(j \times L_r)) \times (I_m \times \phi_n / \omega_n)$$

$$c_1 = \omega_n$$

$$, T_r = L_r / R_r$$

$$e = V_m / (\sigma \times L_s \times I_m).$$

$$f = p \times C_n / (j \times \omega_n).$$

Annexe 3

Linéarisation par Backsteeping

Linéarisation entrée-sortie (Backsteeping) :**Cas monovariable :**

Afin d'expliciter la notion de bouclage non linéaire, on considère d'abord le cas monovariable. Partant d'un système non linéaire appartenant à la classe des systèmes définis par :

$$\Sigma = \begin{cases} \dot{x} = f(x) + g(x)u \\ y = h(x) \end{cases} \quad \text{où : } \begin{cases} x : \text{l'état} \in \mathbb{R}^n; \\ u : \text{l'entrée} \in \mathbb{R}; \\ y : \text{sortie} \in \mathbb{R}. \end{cases} \quad (\text{An3-1})$$

avec : f, g et h sont des fonctions lisses.

Le bouclage non linéaire consiste à trouver un retour d'état de la forme :

$$u = \alpha(x) + \beta(x)v \quad (\text{An3-2})$$

qui puisse ramener le comportement entrée-sortie du système défini par l'expression (An3-1) à celui d'un système linéaire. La première dérivée de y peut être représentée à l'aide de la dérivée directionnelle de Lie suivant un champ de vecteur [36] :

$$\dot{y} = \frac{\partial h}{\partial x} [f(x) + g(x)u] = L_f h(x) + L_g h(x)u \quad (\text{An3-3})$$

Soit x_0 un état d'équilibre du système en boucle ouverte.

Si : $L_g h(x_0) \neq 0$, y est de degré relatif égal à 1 à x_0 (puisque la fonction est lisse, $\frac{\partial h(x_0)}{\partial x} g(x_0) \neq 0$, implique qu'il existe un voisinage V_0 de x_0 tel que $\frac{\partial h(x)}{\partial x} g(x) \neq 0$) et la commande est donnée par :

$$u = \frac{1}{L_g h(x)} [-L_f h(x) + v] \quad (\text{An3-4})$$

Cette expression de u permet d'obtenir un système linéaire où la sortie est séparée de l'entrée par un simple intégrateur. Pour cela, il suffit de substituer (An3-4) dans (An3-3). Nous obtenons :

$$\dot{y} = v \quad (\text{An3-5})$$

Par contre si $L_g h(x_0) \neq 0$, la commande n'apparaît pas. Deux cas se présentent :

- S'il existe un point arbitraire x proche de x_0 tel que : $\frac{\partial h(x)}{\partial x} g(x) \neq 0$, on ne peut pas définir un degré relatif à x_0 .
- S'il existe un voisinage V_0 de x_0 tel que : $\frac{\partial h(x)}{\partial x} g(x) = 0$, pour tout $x \in V_0$, alors le degré relatif de y à x_0 peut être défini et on continue la dérivation de y jusqu'à obtenir :

$$y^{(i)} = L_f^i h(x) + L_g L_f^{i-1} h(x)u \quad (\text{An3-6})$$

tel que : $L_g L_f^{i-1} h(x) \neq 0$.

Si cette condition est vérifiée pour tout $x \in \mathfrak{R}^n$, ce degré relatif est défini globalement. Ainsi, l'idée est de trouver le degré de dérivation δ (degré relatif de $h(x)$) qui est le nombre de fois qu'il faut dériver y tel que u apparaisse.

$$y^{(\delta)} = L_f^\delta h(x) + L_g L_f^{\delta-1} h(x)u \quad (\text{An3-7})$$

La commande est ainsi donnée par :

$$u = \frac{1}{L_g L_f^{\delta-1} h(x)} [-L_f^\delta h(x) + v] \quad (\text{An3-8})$$

Cette expression conduit à un système linéaire équivalent à une chaîne de δ intégrateurs comme le montre la figure A3.1.

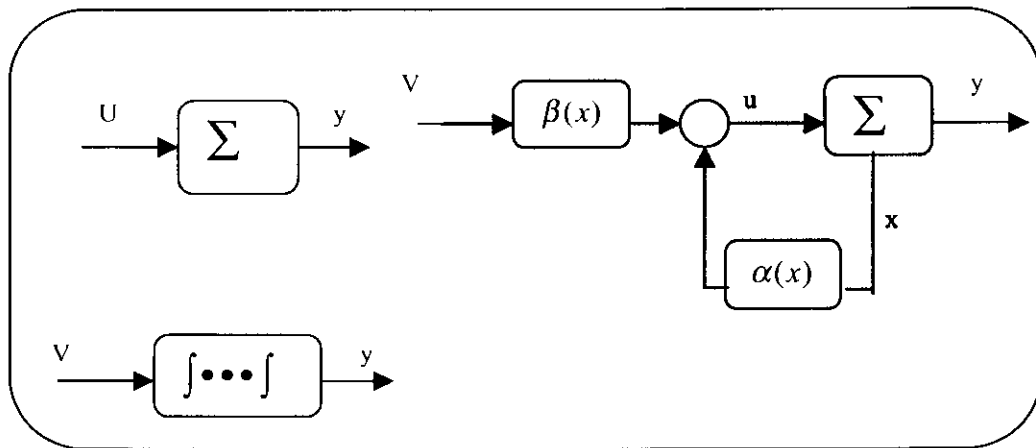


Figure A3.1 : Linéarisation entrée-sortie (cas monovariante)

On construit le difféomorphisme (changement de coordonnées) suivant :

$$\begin{cases} z_1 = F_1(x) = h(x) \\ z_2 = F_2(x) = L_f h(x) \\ \vdots \\ z_\delta = F_\delta(x) = L_f^{\delta-1} h(x) \end{cases} \quad (\text{An3-9})$$

En utilisant les outils de la géométrie différentielle [36], on peut montrer que si le degré relatif est inférieur au degré du système ($\delta < n$), il est possible de compléter le difféomorphisme (An3-9) avec $(n - \delta)$ fonctions : $\xi = [F_{\delta+1}(x), \dots, F_n(x)]$, mais avec des dynamiques ne dépendant pas directement de u . On obtient le nouveau système d'équations :

$$\begin{cases} \dot{z}_1 = z_2 \\ \dot{z}_2 = z_3 \\ \vdots \\ \dot{z}_{\delta-1} = z_\delta \\ \dot{z}_\delta = L_f^\delta h(x) + L_g L_f^{\delta-1} h(x) = f_1(z) + g_1(z) u \\ \dot{\xi} = \psi(z_1, \dots, z_\delta, \xi) \end{cases} \quad (\text{An3-10})$$

En écriture matricielle :

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \vdots \\ \dot{z}_{\delta-1} \\ \dot{z}_\delta \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{\delta-1} \\ z_\delta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} v \quad (\text{An3-11})$$

Remarque :

Si le degré relatif δ de $h(x)$ est strictement inférieur au degré du système n , le retour linéarisant rend certaines dynamiques non observables et de ce fait apparaît ce qu'on appelle la dynamique des zéros. Si cette dynamique est asymptotiquement stable, la commande linéarisante assurera la stabilité interne du système en boucle fermée [36].

Cas multivariable :

Le cas multivariable n'est qu'une extension du cas monovariable. Considérant un système à p entrées et p sorties représenté par :

$$\left\{ \begin{array}{l} \dot{x} = f(x) + \sum_{i=1}^p g_i(x)u_i \\ y_1 = h_1(x) \\ \vdots \\ y_p = h_p(x) \end{array} \right\} \left\{ \begin{array}{l} \text{les sorties que nous souhaitons découplées ;} \\ y_i \in \mathfrak{R} \end{array} \right. \quad (\text{An3-12})$$

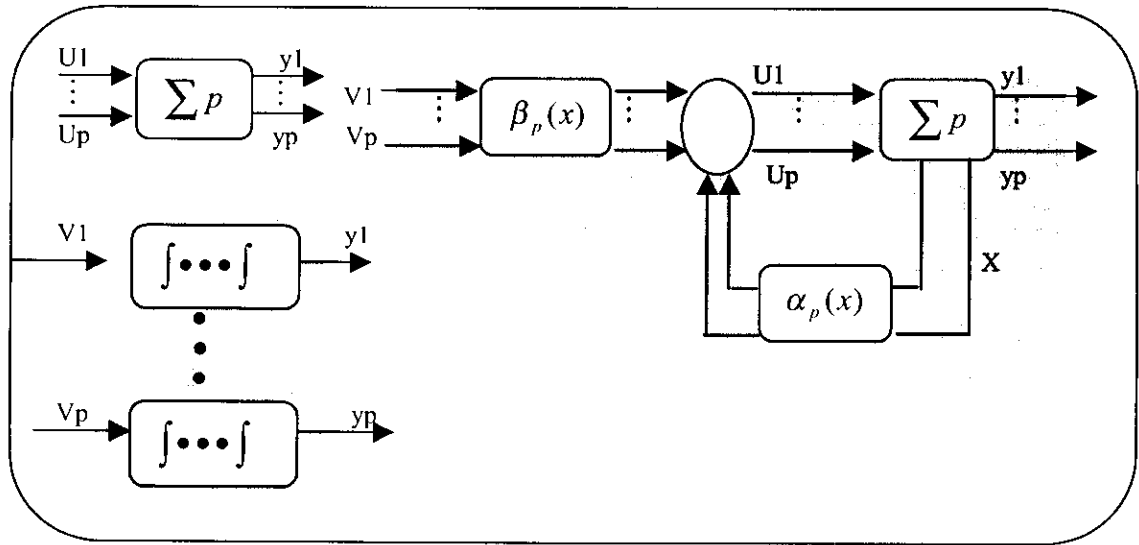


Figure A3.2 : Linéarisation entrée-sortie (cas multivariable).

Il existe des conditions permettant de vérifier si oui ou non un système est découplable [36]. Chaque sortie y_i est dérivée δ_i fois tel que $L_{g_i} L_f^{\delta_i-1} h_i$ soit différente de zéro $x \in \mathfrak{R}^n$.

$$y_j^{(\delta_j)} = L_f^{\delta_j} h_j(x) + \sum_{i=1}^p \left(L_{g_i} L_f^{\delta_j-1} h_j(x) \right) u_i \quad (\text{An3-13})$$

on définit une matrice $A(x) \in \mathfrak{R}^p \times \mathfrak{R}^p$ de découplage :

$$A(x) = \begin{bmatrix} L_{g_1} L_f^{\delta_1-1} h_1 & \cdot & \cdot & \cdot & L_{g_p} L_f^{\delta_1-1} h_1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ L_{g_1} L_f^{\delta_p-1} h_p & \cdot & \cdot & \cdot & L_{g_p} L_f^{\delta_p-1} h_p \end{bmatrix} \quad (\text{An3-14})$$

Le système devient :

$$\begin{bmatrix} y_1^{(\delta_1)} \\ \vdots \\ y_p^{(\delta_p)} \end{bmatrix} = \begin{bmatrix} L_f^{\delta_1} h_1 \\ \vdots \\ L_f^{\delta_p} h_p \end{bmatrix} + A(x) \begin{bmatrix} u_1 \\ \vdots \\ u_p \end{bmatrix} \quad (\text{An3-15})$$

si $A(x)$ est non singulière, le retour d'état statique qui linéarise le système est donné par :

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_p \end{bmatrix} = -A^{-1}(x) \begin{bmatrix} L_f^{\delta_1} h_1 \\ \vdots \\ L_f^{\delta_p} h_p \end{bmatrix} + A(x) \begin{bmatrix} v_1 \\ \vdots \\ v_p \end{bmatrix} \quad (\text{An3-16})$$

Remarque :

Dans le cas où la matrice A est singulière, un découplage dynamique est nécessaire [36]. On construit le difféomorphisme suivant :

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_p \end{bmatrix}, \text{ avec } z_i = \begin{bmatrix} z_{i,1} \\ z_{i,2} \\ \vdots \\ z_{i,\delta_i} \end{bmatrix} = \begin{bmatrix} h_i(x) \\ L_f h_i(x) \\ \vdots \\ L_f^{\delta_i-1} h_i(x) \end{bmatrix}, i = 1, p \quad (\text{An3-17})$$

si $m = (\delta_1 + \dots + \delta_p) < n$ (la somme des degrés relatifs est inférieure à l'ordre du système), nous complétons le difféomorphisme avec $n-m$ fonctions : $\xi = [F_1(x), \dots, F_{n-m}(x)]$.

Dans ces nouvelles coordonnées, le système devient :

$$\dot{z}_i = \begin{bmatrix} \dot{z}_{i,1} \\ \dot{z}_{i,2} \\ \vdots \\ \dot{z}_{i,\delta_i-1} \\ \dot{z}_{i,\delta_i} \end{bmatrix} = \begin{bmatrix} z_{i,2} \\ z_{i,3} \\ \vdots \\ z_{i,\delta_i} \\ L_f^{\delta_i} h_i(x) + L_g L_f^{\delta_i-1} h_i(x) u \end{bmatrix}, i = 1, p \text{ et } \begin{cases} y_1 = z_{1,1} \\ y_2 = z_{2,1} \\ \vdots \\ y_p = z_{p,1} \end{cases} \quad (\text{An3-18})$$

en notation matricielle, on a :

$$\begin{bmatrix} \dot{z}_1 \\ \vdots \\ \dot{z}_m \end{bmatrix} = A \begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix} + B \begin{bmatrix} v_1 \\ \vdots \\ v_p \end{bmatrix} \quad (\text{An3-19})$$

$$\dot{\xi} = \Psi(z) + \Theta(z)u(z) \quad (\text{An3-20})$$

avec :

$$A = \text{diag}(A_1, \dots, A_p), B = \text{diag}(B_1, \dots, B_p); \quad (\text{An3-21})$$

et

$$A_i = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}_{\delta_i \times \delta_i}, \quad B_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}_{\delta_i \times 1}, \quad i = 1, \dots, p.$$

Si $(x_0 = \bar{0}) \in \mathcal{R}$ est un point d'équilibre, les dynamiques des zéros sont données par :

$$\dot{\xi} = \Psi(\bar{0}, \xi) + \Theta(\bar{0}, \xi) \cdot u(\bar{0}, \xi), \quad (\text{An3-22})$$

Il faut prouver la stabilité de la dynamique des zéros afin d'appliquer la linéarisation [36].

Les états $F_j, j = 1, n - m$ du sous système non linéaire ont été rendu non observables à la sortie y par l'expression (An3-16) de u . C'est l'équivalent non linéaire du placement de δ pôles du système à l'origine et l'élimination de $n - (m = \sum \delta_i)$ zéros par les $n - (m = \sum \delta_i)$ pôles restants. Reste à prouver que la dynamique des zéros est stable. L'expression (An3-16) de u ne garantie pas la stabilité interne du système néanmoins, elle garantie la stabilité des sorties, à moins d'établir séparément la stabilité du sous-système non linéaire.

Validité de l'application de la commande linéarisante au modèle de la MAS :

Il faut avant tout vérifier la validité de l'application de la commande par bouclage non linéaire sur le modèle de la machine asynchrone présenté au chapitre 3.

Rappelons-en les équations :

$$\begin{cases} \dot{x}_1 = -a_1 x_1 + b_1 x_3 + c_1 \omega_s x_2 + d_1 x_4 x_5 + e V_{ds} \\ \dot{x}_2 = -a_1 x_2 + b_1 x_4 - c_1 \omega_s x_1 - d_1 x_3 x_5 + e V_{qs} \\ \dot{x}_3 = -a_2 x_3 + b_2 x_1 + c_1 \omega_s x_4 - c_1 x_4 x_5 \\ \dot{x}_4 = -a_2 x_4 + b_2 x_2 - c_1 \omega_s x_3 + c_1 x_3 x_5 \\ \dot{x}_5 = -a_3 x_5 + d_3 x_2 x_3 - d_3 x_1 x_4 - f.C_r \end{cases} \quad (\text{An3-23})$$

On choisit les grandeurs de sorties comme indiquées ci-après :

$$\begin{cases} y_1 = h_1(x) = \frac{1}{2}(x_3^2 + x_4^2) = \frac{1}{2}\Phi_r^2 \\ y_2 = h_2(x) = x_5 = \omega \end{cases} \quad (\text{An3-24})$$

Ainsi, le système (entrées – sorties – commandes) appartient à la classe des systèmes représentés par des équations non linéaires de la forme :

$$\begin{cases} \dot{x} = f(x) + g(x)u \\ \begin{cases} y_1 = h_1(x) \\ y_2 = h_2(x) \end{cases} \end{cases} \quad \text{les sorties que nous désirons découpler et commander.}$$

Avec : $x = [x_1, x_2, x_3, x_4, x_5]^T = [i_{ds}, i_{qs}, \phi_{dr}, \phi_{qr}, \omega]^T$,
 $u = [u_1, u_2]^T = [V_{ds}, V_{qs}]^T$,
 $f(x) = [f_1(x), f_2(x), f_3(x), f_4(x), f_5(x)]^T$,
 $g(x) = [g_1(x), g_2(x)]^T$ où : $g_1(x) = [e, 0, 0, 0, 0]^T$, $g_2(x) = [0, e, 0, 0, 0]^T$.

Afin de vérifier les conditions d'application de la commande, on construit les distributions suivantes [36] :

$$\begin{cases} G_0 = \text{span} \{g_1, g_2\} \\ G_1 = \text{span} \{g_1, g_2, \text{ad}_f g_1, \text{ad}_f g_2\} \end{cases}$$

Dans notre cas, G_0 est introduite et de rang constant égal à 2. G_1 est de rang 4 mais n'est pas involutive.

On remarquera que $\text{rang}(G_1) < 5$ (inférieur au degré du système), par conséquent, le système n'est pas entièrement linéarisable. Néanmoins le plus grand sous-système linéarisable est de dimension 4. Notons aussi que la dimension du sous-système linéarisé dépendra des sorties, car il n'est autre que la somme des degrés relatifs associés aux sorties.

Calcul des degrés relatifs :

Soit la première sortie :

$$y_1 = h_1(x) = \frac{1}{2}(x_3^2 + x_4^2) = \frac{1}{2}\Phi_r^2 \quad (\text{An3-25})$$

Le calcul de sa dérivée donne :

$$\dot{y}_1(x) = -2a_2 y_1(x) + b_2(x_1 x_3 + x_2 x_4) \quad (\text{An3-26})$$

Il est clair que dans l'expression de $\dot{y}_1$ aucune commande n'apparaît. Par conséquent, le degré relatif étant différent de 1 et on continue la dérivation jusqu'à faire apparaître au moins une commande.

Le calcul de la seconde dérivée donne :

$$y_1^{(2)}(x) = -a_2 \dot{y}_1(x) + b_2(x_1 f_3 + x_3 f_1 + x_2 f_4 + x_4 f_2) + b_2 e(x_3 V_{ds} + x_4 V_{qs}) \quad (\text{An3-27})$$

Les commandes V_{ds} et V_{qs} apparaissent dans l'expression de la deuxième dérivée. Donc le degré relatif δ_1 de y_1 est 2. Afin d'éviter de traîner cette expression, on pose :

$$V_1 = y_1^{(2)}(x) = B_1(x) + b_2 e(x_3 V_{ds} + x_4 V_{qs}) \quad (\text{An3-28})$$

Soit la deuxième sortie :

$$y_2 = h_2(x) = x_5 = \omega \quad (\text{An3-29})$$

Aucune commande n'apparaît dans l'expression de la première dérivée. Le calcul de la deuxième dérivée donne :

$$y_2^{(2)} = -a_3 \dot{y}_2 + d_3(x_3 f_2 + x_2 f_3 - x_1 f_4 - x_4 f_1) + d_3 e(x_3 V_{qs} - x_4 V_{ds}) \quad (\text{An3-30})$$

on pose :

$$V_2 = y_2^{(2)}(x) = B_2(x) + e(x_3 V_{qs} - x_4 V_{ds}) \quad (\text{An3-31})$$

Ainsi, comme pour le flux, le degré relatif δ_2 associé à y_2 est égal à 2.

Comme $[(\delta_1 + \delta_2) = 4 < 5]$, on obtient une dynamique non observable d'ordre 1. On peut effectuer le changement de coordonnées suivant :

$$\begin{cases} Z_1 = h_1(x) \\ Z_2 = L_f h_1(x) = \dot{Z}_1 \\ Z_3 = h_2(x) \\ Z_4 = L_f h_2(x) = \dot{Z}_3 \end{cases} \quad (\text{An3-32})$$

Pour compléter le difféomorphisme, on introduit une cinquième variable. On prendra Z_5 l'angle du flux rotorique dans l'axe (d-q). Le système (An3-23) s'exprime dans les nouvelles coordonnées de la façon suivante :

$$\begin{cases} \dot{Z}_1 = Z_2 = L_f h_1(x) \\ \dot{Z}_2 = L_f^2 h_1(x) + L_g L_f h_1(x)u \\ \dot{Z}_3 = Z_4 = L_f h_2(x) \\ \dot{Z}_4 = L_f^2 h_2(x) + L_g L_f h_2(x)u \\ \dot{Z}_5 = \frac{x_4 f_3 - x_3 f_4}{x_4 \Phi_r} \end{cases} \quad (\text{An3-33})$$

avec : $Z_5 = \arctg(x_3 / x_4)$.

On aboutit, à partir de (An3-9) et (An3-15), à l'expression des commandes suivantes :

$$\begin{bmatrix} V_{ds} \\ V_{qs} \end{bmatrix} = \begin{bmatrix} b_2 e x_3 & b_2 e x_4 \\ -d_3 e x_4 & d_3 e x_3 \end{bmatrix}^{-1} \left(\begin{bmatrix} -B_1(x) \\ -B_2(x) \end{bmatrix} + \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} \right) \quad (\text{An3-34})$$

En écriture abrégée, on pose :

$$V_s = (A(x))^{-1} (B(x) + V) \quad (\text{An3-35})$$

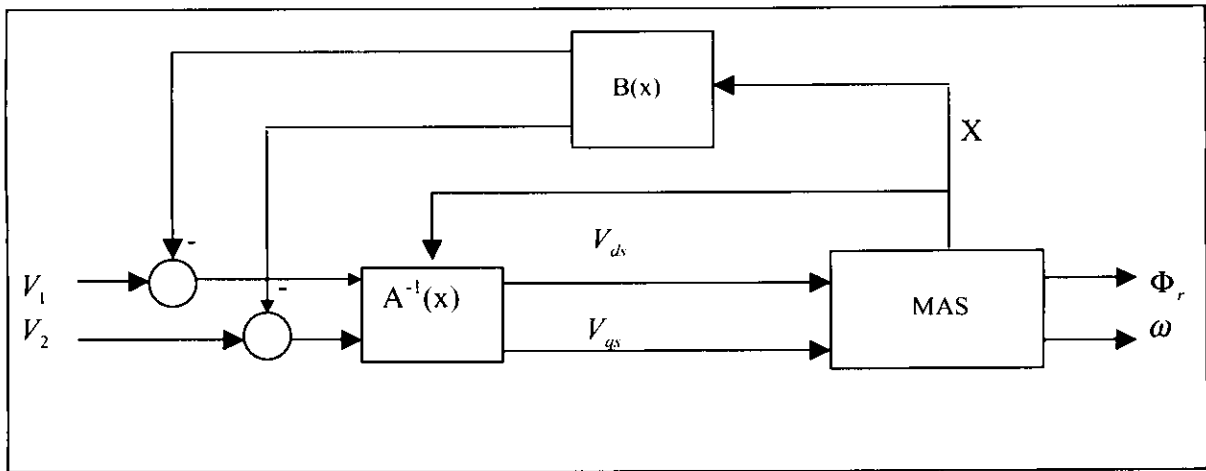


Figure A3.3 : Structure de la commande par bouclage non linéaire (cas de la MAS).

Cela n'étant possible que si $A(x)$ est inversible, ceci est vérifié si : $(x_3^2 + x_4^2 \neq 0)$. Donc il apparaît une singularité uniquement à l'instant initial.

Enfin, on obtient le système linéarisé suivant :

$$\begin{cases} \dot{Z}_1 = Z_2 \\ \dot{Z}_2 = V_1 \\ \dot{Z}_3 = Z_4 \\ \dot{Z}_4 = V_2 \\ \dot{Z}_5 = \frac{x_3 f_4 - x_4 f_3}{x_3 \Phi_r} \end{cases} \quad (\text{An3-36})$$

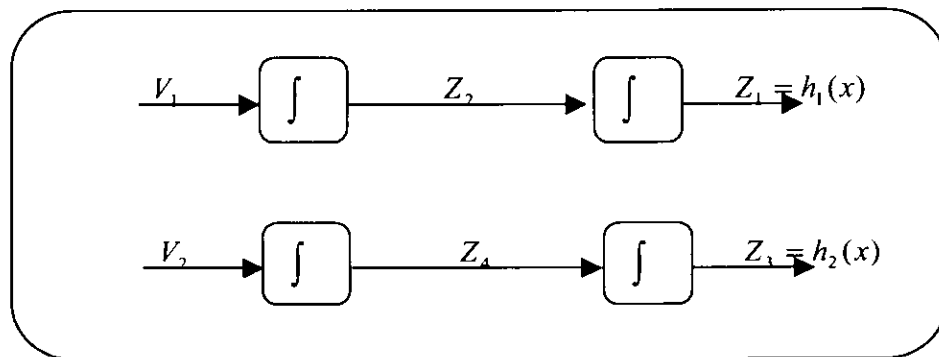


Figure A3.4 : Linéarisation entrée-sortie et découplage.

Annexe 4

Code Source du Programme du Microcontrôleur 16F877

```

=====LE CODE SOURCE DU SERVOMOTOR PIC16F877=====
//-----
//
// File:16mot877.c
//
//
// This code implements a brush-DC servomotor using the PIC18F877
// MCU.
// The code was compiled using the HiTech PICC compiler ver. 7.85.
//
// The following files should be included in the MPLAB project:
//
// 16mot877.c-- Main source code file
//
//-----
#include <pic.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
//-----
//Constant Definitions
//-----
#define DIST 0           // Array index for segment distance
#define VEL 1            // Array index for segment vel. limit
#define ACCEL 2          // Array index for segment accel.
#define TIME 3           // Array index for segment delay time
#define INDEX RB0        // Input for encoder index pulse
#define NLIM RB1         // Input for negative limit switch
#define PLIM RB2         // Input for positive limit switch
#define GPI RB3          // General purpose input
#define MODE1 !RB4       // DIP switch #1
#define MODE2 !RB5       // DIP switch #2
#define MODE3 !RB6       // DIP switch #3
#define MODE4 !RB7       // DIP switch #4
#define SPULSE RC5       // Software timing pulse output
#define ADRES ADRESH     // Redefine for 10-bit A/D converter
#define KP 250
#define KI 252
#define KD 254
//-----
// Variable declarations
//-----
const char ready[] = "\n\rREADY>";
const char error[] = "\n\rERROR!";
char inbuf[8];
char tempch,UpCount;
unsigned char
eadr,
firstseg,
lastseg,
segnum,
parameter,
i,           // index to ASCII buffer
comcount,   // index to input string

```

```

udata                                // received character from USART;

struct {                              // holds status bits for servo
unsigned phase:1;                    // first half/ second half of profile
unsigned neg_move:1;                 // backwards relative move
unsigned motion:1;                   //
unsigned saturated:1;                // servo output is saturated
unsigned bit4:1;
unsigned bit5:1;
unsigned run:1;
unsigned loop:1;
} stat ;
union INTVAL
{
int i;
char b[2];
};
union LNG
{
long l;
unsigned long ul;
int i[2];
unsigned int ui[2];
char b[4];
unsigned char ub[4];
};
bank1 union LNG
accel,
temp,
u0,
ypid,
velact,
phaseldist
;
bank1 int
dtime,                                // Segment delay time
integral,
kp,ki,kd,                             // PID gain constants
vlim, velcom,
mvelocity,
DnCount
;
bank1 long
position,    // Commanded position.
mposition,   // Actual measured position.
fposition,   // Originally commanded position.
flatcount;   // Holds # of sample periods for which the
               // velocity limit was reached in first half of the move.

bank2 int segment1[12][4]; // array in bank2 for 12 motion segments
bank3int segment2[12][4];  // array in bank3 for 12 motion segments
// Function Prototypes-----
interrupt void servo_isr( void);
void putsUSART(const char *data);
void DoCommand(void);
void Setup(void);

```

```

void UpdPos(void);
void UpdTraj(void);
void CalcError(void);
void CalcPID(void);
void SetupMove(void);
void EEDatWrite(unsigned char address, int data);
int EEDatRead(unsigned char address);
//-----
// servo_isr()
// Performs the servo calculations
//-----
interrupt void servo_isr(void)
{
    SPULSE = 1;           // Toggle output pin for ISR code timing
    UpdTraj();            // Get new commanded position
    UpdPos();             // Get new measured position
    CalcError();          // Calculate new position error
    CalcPID();
    TMR2IF = 0;           // Clear Timer2 Interrupt Flag.
    SPULSE = 0;           // Toggle output pin for ISR code timing
}
//-----
// UpdTraj()
// Computes the next required value for the next commanded motor
// position based on the current motion profile variables.
// Trapezoidal
// motion profiles are produced.
//-----
void UpdTraj(void)
{
    if(stat.motion && !stat.saturated)
    {
        if(!stat.phase)           // If in the first half of the move.
        {
            if(velact.ui[1] < vlim) // If still below the velocity limit
            {
                velact.ul += accel.ul; // Accelerate
            }
            else // If velocity limit has been reached,
            {
                flatcount++; // increment flatcount.
                temp.ul = velact.ul; // Put velocity value into temp
                // and round to 16 bits
                if(velact.ui[0] == 0x8000)
                {
                    if(!(velact.ub[2] & 0x01))
                    {
                        temp.ui[1]++;
                    }
                    else
                    {
                        if(velact.ui[0] > 0x8000) temp.ui[1]++;
                    }
                }
                phaseldist.ul -= (unsigned long)temp.ui[1];
                if(stat.neg_move)
                {
                    position -= (unsigned long)temp.ui[1];
                }
                else
                {
                    position += (unsigned long)temp.ui[1];
                }
                if(phaseldist.l <= 0) // If phaseldist is negative
                { // first half of the move has

```

```

stat.phase = 1;                // completed.
}
else                          // If in the second half of the move,
{                             // Decrement flatcount if not 0.
if(flatcount) flatcount--;
else
if(velact.ul)                 // If commanded velocity not 0,
{
velact.ul -= accel.ul;       // Decelerate
if(velact.i[1] < 0)
velact.l = 0;
}
else                          // else
{
if(dtime) dtime--;          // Decrement delay time if not 0.
else
{
stat.motion = 0;            // Move is done, clear motion flag
position = fposition;
}
}
temp.ul = velact.ul;          // Put velocity value into temp
// and round to 16 bits
if(velact.ui[0] == 0x8000)
{
if(!(velact.ub[2] & 0x01))
temp.ui[1]++;
else;
}
else
if(velact.ui[0] > 0x8000) temp.ui[1]++;
else;
if(stat.neg_move)             // Update commanded position
position -= (unsigned long)temp.ui[1];
else
position += (unsigned long)temp.ui[1];
}
}                             // END if (stat.motion)
else
{
if(stat.run && !stat.motion) // If motion stopped and profile
{                             // running, get next segment number
if(segnum < firstseg) segnum = firstseg;
if(segnum > lastseg)
{
segnum = firstseg;          // Clear run flag if loop flag not set.
if(!stat.loop) stat.run = 0;
}
}
else
{
SetupMove();                // Get data for next motion segment.
segnum++;                    // Increment segment number.
}
}
else;
}

```

```

}
//-----
// SetupMove()
// Gets data for next motion segment to be executed
//-----
void SetupMove(void)
{
    if(segnum < 12)                // Get profile segment data from
    { // data memory.
        phaseldist.i[0] = segment1[segnum][DIST];
        vlim = segment1[segnum][VEL];
        accel.i[0] = segment1[segnum][ACCEL];
        dtime = segment1[segnum][TIME];
    }
    else if(segnum < 24)
    {
        phaseldist.i[0] = segment2[segnum - 12][DIST];
        vlim = segment2[segnum - 12][VEL];
        accel.i[0] = segment2[segnum - 12][ACCEL];
        dtime = segment2[segnum - 12][TIME];
    }
    phaseldist.b[2] = phaseldist.b[1]; // Rotate phaseldist one byte
    phaseldist.b[1] = phaseldist.b[0]; // to the left.
    phaseldist.b[0] = 0;
    if(phaseldist.b[2] & 0x80)          // Sign-extend value
        phaseldist.b[3] = 0xff;
    else
        phaseldist.b[3] = 0;
    accel.b[3] = 0;                    // Rotate accel one byte to
    accel.b[2] = accel.b[1];           // the left
    accel.b[1] = accel.b[0];
    accel.b[0] = 0;
    temp.l = position;
    if(temp.ub[0] > 0x7f)               // A fractional value is left
        temp.l += 0x100;              // over in the 8 LSbits of
    temp.ub[0] = 0;                    // position, so round position
    // variable to an integer value
    position = temp.l;                 // before computing final move
    // position.
    fposition = position + phaseldist.l; // Compute final position for
    // the move
    if(phaseldist.b[3] & 0x80)          // If the move is negative,
    {
        stat.neg_move = 1;             // Set flag to indicate negative
        phaseldist.l = -phaseldist.l; // move.
    }
    else stat.neg_move = 0;            // Clear flag for positive move
    phaseldist.l >>= 1;                // phaseldist holds total
    // move distance, so divide by 2
    velact.l = 0;                     // Clear commanded velocity
    flatcount = 0;                    // Clear flatcount
    stat.phase = 0;                   // Clear flag: first half of move
    if(accel.l && vlim)
        stat.motion = 1;              // Enable motion
}

```

```

//-----
// UpdPos()
//-----
void UpdPos(void)
{
    mvelocity = DnCount;           // Get old DnCount value
    temp.b[0] = TMR1L;             // Read Timer1
    temp.b[1] = TMR1H;
    if(TMR1L < temp.b[0])           // If a rollover occurred, read
    {                               // Timer1 again.
        temp.b[0] = TMR1L;
        temp.b[1] = TMR1H;
    }
    DnCount = temp.i[0];           // Store timer value in DnCount
    mvelocity -= DnCount;          // Subtract new value from
                                // measured velocity
    tempch = -UpCount;             // Put old UpCount value in
                                // temporary variable.
    UpCount = TMR0;               // Read Timer0
    tempch += UpCount;
    if(tempch > 0)
        mvelocity += (int)tempch;
    else
        mvelocity -= (int)(tempch);
    mposition += (long)(mvelocity << 8); // Update measured position
}
//-----
// CalcError()
// Calculates position error and limits to 16 bit result
//-----
void CalcError(void)
{
    u0.l = mposition - position;   // Get error
    u0.b[0] = u0.b[1];             //
    u0.b[1] = u0.b[2];             // shift to the right to discard
    u0.b[2] = u0.b[3];             // lower 8 bits.
    if (u0.b[2] & 0x80)            // If error is negative.
    {
        u0.b[3] = 0xff;           // Sign-extend to 32 bits.
        if((u0.i[1] != 0xffff) || !(u0.b[1] & 0x80))
        {
            u0.ui[1] = 0xffff;     // Limit error to 16-bits.
            u0.ui[0] = 0x8000;
        }
    }
    else;
}
else
    // If error is positive.
    {
        u0.b[3] = 0x00;
        if((u0.i[1] != 0x0000) || (u0.b[1] & 0x80))
        {
            u0.ui[1] = 0x0000;     // Limit error to 16-bits.
            u0.ui[0] = 0x7fff;
        }
    }
    else;
}

```

```

}
//-----
// CalcPID()
// Calculates PID compensator algorithm and determines new value for
// PWM duty cycle
//-----
void CalcPID(void)
{
ypid.l = (long)u0.i[0]*(long)kp; // If proportional gain is not 0,
                                // calculate proportional term.
if(!stat.saturated)             // If output is not saturated,
integral += u0.i[0];            // add present error to integral
                                // value.
if(ki)                           // If integral value is not 0,
ypid.l += (long)integral*(long)ki; // calculate integral term.
if(kd)                           // If differential term is not 0,
ypid.l += (long)mvelocity*(long)kd; // calculate differential term.
if(ypid.b[3] & 0x80)              // If PID result is negative
{
if((ypid.b[3] < 0xff) || !(ypid.b[2] & 0x80))
{
ypid.ui[1] = 0xff80;                // Limit result to 24-bit value
ypid.ui[0] = 0x0000;
}
else;
}
else                             // If PID result is positive
{
if(ypid.b[3] || (ypid.b[2] > 0x7f))
{
ypid.ui[1] = 0x007f;                // Limit result to 24-bit value
ypid.ui[0] = 0xffff;
}
else;
}
ypid.b[0] = ypid.b[1];            // Shift PID result right to
ypid.b[1] = ypid.b[2];            // get upper 16 bits.
stat.saturated = 0;               // Clear saturation flag and see
if(ypid.i[0] > 500)                // if present duty cycle output
{                                  // exceeds limits.
ypid.i[0] = 500;
stat.saturated = 1;
}
if(ypid.i[0] < -500)
{
ypid.i[0] = -500;
stat.saturated = 1;
}
ypid.i[0] += 512;                 // Add offset to get positive
ypid.i[0] <= 6;                  // duty cycle and shift left to
                                // get 8 Msb's in upper byte.
CCPR1L = ypid.b[1];              // Write upper byte to CCP register
                                // to set duty cycle.
                                // Set 2 LSB's of duty cycle in
                                // CCP1CON register.

if(ypid.b[0] & 0x80) CCP1X = 1;

```

```

else CCP1X = 0;
if(ypid.b[0] & 0x40) CCP1Y = 1;
else CCP1Y = 0;
TMR2IF = 0; // Clear Timer2 Interrupt Flag.
SPULSE = 0;
}
//-----
// Setup() initializes program variables and peripheral registers
//-----
void Setup(void)
{
    firstseg = 0; // Initialize motion segment
    lastseg = 0; // variables
    segnum = 0;
    parameter = 0; // Motion segment parameter#
    i = 0; // Receive buffer index
    comcount = 0; // Input command index
    udata = 0; // Holds USART received data
    stat.phase = 0; // Initialize flags and variables
    stat.saturated = 0; // used for servo calculations.
    stat.motion = 0;
    stat.run = 0;
    stat.loop = 0;
    stat.neg_move = 0;
    dtime = 0;
    integral = 0;
    vlim = 0;
    velcom = 0;
    mvelocity = 0;
    DnCount = 0;
    UpCount = 0;
    temp.l = 0;
    accel.l = 0;
    u0.l = 0;
    ypid.l = 0;
    velact.l = 0;
    phaseldist.l = 0;
    position = 0;
    mposition = position;
    fposition = position;
    flatcount = 0;
    memset(inpbuf,0,8); // clear the input buffer
    udata = RCREG;
    udata = RCREG;
    RCREG = 0;
    udata = 0;
    SPBRG = 15; // 19200 baud @ 20MHz
    TXSTA = 0x20; // setup USART transmit
    RCSTA = 0x90; // setup USART receive
    SSPADD = 49; // Setup MSSP for master I2C
    SSPSTAT = 0; // 100Khz @ 20MHz
    SSPCON2 = 0; //
    SSPCON = 0x28; //
    PR2 = 0xff; // Setup Timer2 and CCP1 for
    T2CON = 0x4c; // 19.53 Khz PWM @ 20MHz
    CCPR1L = 0x7f; // 50% initial duty cycle

```

```

CCP1CON = 0x0f;           //
TMR2IF = 0;               //
TMR2IE = 1;               // Enable Timer2 interrupts
TMR1H = 0;               //
TMR1L = 0;               //
T1CON = 0x07;            // Enable Timer1, ext. sync. clock
TMR0 = 0;                //
OPTION = 0x6f;           // Enable Timer0
ADCON1 = 0x04;           // Setup A/D converter
ADCON0 = 0x81;           //
PORTC = 0;               // Clear PORTC
PORTD = 0;               // Clear PORTD
PORTE = 0x00;            // Clear PORTE
TRISC = 0xdb;            //
TRISD = 0;               // PORTD all outputs
TRISE = 0;               // PORTE all outputs
kp = EEDatRead(KP);       // Get PID gain values from
ki = 0;                  // data EEPROM
kd = EEDatRead(KD);       //
putsUSART("\r\nPIC16F877 DC Servomotor");
putsUSART(ready);
PEIE = 1;                // Enable interrupts.
GIE = 1;                 //
}
//-----
// main()
//-----
main()
{
Setup();
for(segnum=0;segnum < 12;segnum++) // Reads profile data into RAM for
{                                  // from data EEPROM.
for(parameter=0;parameter < 4;parameter++)
{
eeadr = (segnum << 3) + (parameter << 1);
segment1[segnum][parameter] = EEDatRead(eeadr);
}
}
segment2[0][DIST] = 29500; // Initialize last 12 motion segments in
segment2[0][VEL] = 4096;   // RAM to pre-determined values for test
segment2[0][ACCEL] = 2048; // purposes. This profile data is used
segment2[0][TIME] = 1200;  // when a profile is selected via the DIP
// switches on the motor PCB.
segment2[1][DIST] = -29500;
segment2[1][VEL] = 1024;
segment2[1][ACCEL] = 512;
segment2[1][TIME] = 1200;

segment2[2][DIST] = 737;
segment2[2][VEL] = 4096;
segment2[2][ACCEL] = 2048;
segment2[2][TIME] = 1200;

segment2[3][DIST] = 737;
segment2[3][VEL] = 4096;
segment2[3][ACCEL] = 2048;

```

```
segment2[3][TIME] = 1200;

segment2[4][DIST] = 738;
segment2[4][VEL] = 4096;
segment2[4][ACCEL] = 2048;
segment2[4][TIME] = 1200;

segment2[5][DIST] = 738;
segment2[5][VEL] = 4096;
segment2[5][ACCEL] = 2048;
segment2[5][TIME] = 1200;

segment2[6][DIST] = -2950;
segment2[6][VEL] = 1024;
segment2[6][ACCEL] = 128;
segment2[6][TIME] = 1200;

segment2[7][DIST] = 2950;
segment2[7][VEL] = 256;
segment2[7][ACCEL] = 64;
segment2[7][TIME] = 1200;

segment2[8][DIST] = -2950;
segment2[8][VEL] = 4096;
segment2[8][ACCEL] = 512;
segment2[8][TIME] = 1200;

segment2[9][DIST] = 29500;
segment2[9][VEL] = 1024;
segment2[9][ACCEL] = 512;
segment2[9][TIME] = 1200;

segment2[10][DIST] = 29500;
segment2[10][VEL] = 2048;
segment2[10][ACCEL] = 512;
segment2[10][TIME] = 1200;

segment2[11][DIST] = 29500;
segment2[11][VEL] = 4096;
segment2[11][ACCEL] = 1024;
segment2[11][TIME] = 1200;

if(MODE1)          // If DIP switch #1 is on, set for looping mode.
stat.loop = 1;
if(MODE2)
{
    // If DIP switches #2,#3, or #4 turned on,
    firstseg = 12;    // execute predetermined profile from above.
    lastseg = 13;
    segnum = 12;
    stat.run = 1;
}
else if(MODE3)
{
    firstseg = 14;
    lastseg = 18;
    segnum = 14;
```

```

stat.run = 1;
}
else if(MODE4)
{
firstseg = 18;
lastseg = 19;
segnum = 18;
stat.run = 1;
}
else;
while(1)
{
CLRWDT();
ADGO = 1;           // Start an A/D conversion
while(ADGO);       // Wait for the conversion to complete
PORTD = 0;         // Clear the LED bargraph display.
PORTE &= 0x04;     //
if(ADRES > 225)
{
PORTE |= 0x03;     // Turn on 10 LEDS
PORTD = 0xff;
}
if(ADRES > 200)
{
PORTE |= 0x01;     // Turn on 9 LEDS
PORTD = 0xff;
}
else if(ADRES > 175) PORTD = 0xff; // Turn on 8 LEDS
else if(ADRES > 150) PORTD = 0x7f; // 7 LEDS
else if(ADRES > 125) PORTD = 0x3f; // 6 LEDS
else if(ADRES > 100) PORTD = 0x1f; // 5 LEDS
else if(ADRES > 75)  PORTD = 0x0f; // 4 LEDS
else if(ADRES > 50)  PORTD = 0x07; // 3 LEDS
else if(ADRES > 25)  PORTD = 0x03; // 2 LEDS
else if(ADRES > 0)   PORTD = 0x01; // 1 LED
else;
if(RCIF)
{
switch(udata = RCREG)
{
case ',': DoCommand(); // process the string
memset(inpbuf,0,8);    // clear the input buffer
i = 0;                // clear the buffer index
comcount++;           // increment comma count
TXREG = udata;        // echo the character
break;
case 0x0d: DoCommand(); // process the string
memset(inpbuf,0,8);    // clear the input buffer
i = 0;                // clear the buffer index
comcount = 0;         // clear comma count
segnum = 0;           // clear segment number
parameter = 0;        // clear paramater
putsUSART(ready);     // put prompt to USART
break;
default: inpbuf[i] = udata; // get received char
i++;                  // increment buffer index
}
}
}

```

```

if(i > 7)                                // If more than 8 chars
{                                          // received before getting
putrsUSART(ready);                      // a <CR>, clear input
memset(inpbuf,0,8);                     // buffer
i = 0;                                  // the buffer index
}
else TXREG = udata;                      // echo character
break;                                  //
}                                        //end switch(udata)
}                                        //end if(RCIF)
}                                        //end while(1)
}

//-----
// DoCommand()
// Processes incoming USART data.
//-----
void DoCommand(void)
{
if(comcount == 0)                        // If this is the first parameter of the input
{                                        // command...
switch(inpbuf[0])
{
case 'X': parameter = DIST;             // Segment distance change
break;
case 'V': parameter = VEL;             // Segment velocity change
break;
case 'A': parameter = ACCEL;           // Segment acceleration change
break;
case 'T': parameter = TIME;            // Segment delay time change
break;
case 'P': parameter = 'P';             // Change proportional gain
break;
case 'I': parameter = 'I';            // Change integral gain
break;
case 'D': parameter = 'D';            // Change differential gain
break;
case 'L': parameter = 'L';            // Loop a range of segments
break;
case 'S': stat.run = 0;                // Stop execution of segments
break;
case 'G': parameter = 'G';            // Execute a range of segments
break;
case 'W': if(RE2)                      // Enable or disable motor
{                                        // driver IC
putrsUSART("\r\nPWM On");
RE2 = 0;
}
else
{
putrsUSART("\r\nPWM Off");
RE2 = 1;
}
break;
default: if(inpbuf[0] != '\0')
{
putrsUSART(error);

```

```

}
break;
}
}
else if(comcount == 1)           // If this is the second parameter of
                                // the
{                                // input command.
if(parameter < 4) segnum = (char)(atoi(inpbuf));
else
switch(parameter)
{
case 'P': kp = atoi(inpbuf);    // proportional gain change
EEDatWrite(KP, kp);            // Store value in EEPROM
break;
case 'I': ki = atoi(inpbuf);    // integral gain change
EEDatWrite(KI, ki);            // Store value in EEPROM
break;
case 'D': kd = atoi(inpbuf);    // differential gain change
EEDatWrite(KD, kd);            // Store value in EEPROM
break;
case 'G': firstseg = (char)(atoi(inpbuf));
break;

                                // Get the first segment in
                                // the range to be executed.
case 'L': firstseg = (char)(atoi(inpbuf));
break;
default: break;
}
}
else if(comcount == 2)
{
if(!stat.run)                 // If no profile is executing
{
if(parameter < 4) // If this was a segment parameter
{
// change.
if(segnum < 12)
{
// Write the segment parameter into data memory
segment1[segnum][parameter] = atoi(inpbuf);
// Compute EEPROM address and write data to EEPROM
eeadr = (segnum << 3) + (parameter << 1);
EEDatWrite(eeaddr, segment1[segnum][parameter]);
}
else if(segnum < 24)
// Write segment parameter data into data memory
segment2[segnum - 12][parameter] = atoi(inpbuf);
}
else switch(parameter)
{
case 'G': lastseg = (char)(atoi(inpbuf)); // Get value for
segnum = firstseg;                        // last segment.
stat.loop = 0;
stat.run = 1;                             // Start profile.
break;
case 'L': lastseg = (char)(atoi(inpbuf)); // Get value for
segnum = firstseg;                        // last segment.

```

```

stat.loop = 1;                                // Enable looping
stat.run = 1;                                // Start profile
break;
default: break;
}
}
}
else;
}
//-----
// putrsUSART()
//
// Puts a string of characters in program memory to the USART
//-----
void putrsUSART(const char *data)
{
do
{
while(!(TXSTA & 0x02));
TXREG = *data;
} while( *data++ );
}
//-----
// EEDatWrite()
//
// Writes an integer value to the 16F877 data EEPROM memory at
// the specified address.
//-----
void EEDatWrite(unsigned char address, int data)
{
union INTVAL temp;
temp.i = data;
while(WR);                                // If write in progress, wait until done.
EEADR = address;                          // Load address to be written.
EEDATA = temp.b[0];                       // Load data to be written.
EEPGD = 0;                                // Point to data memory.
WREN = 1;                                  // Enable writes.
GIE = 0;                                  // Disable interrupts.
EECON2 = 0x55;                            // Required write sequence.
EECON2 = 0xaa;                            //
WR = 1;                                    // Do the write.
while(WR);                                // If write in progress, wait until done.
EEADR++;                                  // Increment address.
EEDATA = temp.b[1];                       // Load data to be written.
EECON2 = 0x55;                            // Required write sequence.
EECON2 = 0xaa;                            // Required write sequence.
WR = 1;                                    // Do the write.
while(WR);                                // Wait for write to finish.
GIE = 1;                                  // Reenable interrupts.
WREN = 0;                                  // Disable writes.
}

//-----//
EEDatRead()
//

```

```
//Reads an integer value as two bytes from the 16F877 data EEPROM at
//the specified address location
//-----
int EEDatRead(unsigned char address)
{
union INTVAL data;
EEADR = address;           // Load the address.
EPPGD = 0;                 // Point to data memory.
RD = 1;                    // Do the read.
data.b[0] = EEDATA;        // Get the data.
EEADR++;                   // Point to next address.
RD = 1;                    // Do the read.
data.b[1] = EEDATA;        // Get the data.
return data.i;
}
```

Références Bibliographiques

Références Bibliographiques

- [1] **P. Guay** “Modèle dynamique des mécanismes : Formulation à l’aide du groupe de Lie des déplacements”, Thèse de doctorat, INSA de Lyon, France, Mars 1989.

- [2] **S. Charentus** “Modélisation et commande d’un manipulateur redondant composé de plusieurs plate formes de Stewart”, Thèse de doctorat, Université Paul Sabatier, Toulouse, France, Avril 1990.

- [3] **B. Gorla et M. Renaud** “Modèles des robots manipulateurs, application à leur commande”, Cepadues édition, 1984.

- [4] **D. Chablat** “Domaines d’unicité et parcourabilité pour les manipulateurs pleinement parallèles”, Thèse de doctorat, Ecole Centrale de Nantes, France, 1998.

- [5] **C. M. Gosselin** “Kinematic Analysis, Optimisation and Programming of Parallel robotic Manipulators ”, Thèse de doctorat, McGill, 1988.

- [6] **J-P. Merlet** “Les robots parallèles”, 2^e édition, HERMES, Paris, 1997.

- [7] **E. Dombre et W. Khalil** “Modélisation et Commande des Robots”, Hermès, Paris, 1988.

- [8] **F. Pierrot** “Robot Pleinement Parallèle Légers : Conception, Modélisation et Commande”, Thèse de Doctorat, Montpellier, Avril 1991.

- [9] **M. Aït-Ahmed** “Contribution à la modélisation géométrique et dynamique des robots parallèle”, Thèse de doctorat, Toulouse, 1993.

- [10] **Youram Koren** “La Robotique pour Ingénieurs”, Traduit de l’américain par Bernard Loubieres, McGRAW-HILL, Paris, 1986.

- [11] **J.P. Merlet** "Parallel manipulators, Part 2 : Theory singular configuration and Grassmann geometry ", Rapport de recherche no. 791, INRIA, Sophia-Antipolis, France, Février 1988.

- [12] **J.P. Merlet** "Manipulateurs parallèles, 4^{ème} partie : Mode d'assemblage et cinématique directe sous forme polynomiale", Rapport de recherche, INRIA, Sophia-Antipolis, France, Octobre 1989.

- [13] **C. Reboulet** "Modélisation des robots parallèles", Technique de la robotique : Architectures et commandes, Edition : Hermes, 1988.

- [14] **E. F. Fichter** "A Stewart platform-based manipulator : General theory and practical construction", The International Journal of Robotics Research, Vol. 5, No. 2, 1986.

- [15] **R. Bricard** "Mémoire sur la théorie de l'octaèdre articulé", Journal de mathématiques pures et appliquées, Gauthier-Villars et Fils, Paris, Tome 3, 1897.

- [16] **M. Gati et M. Mazouz** "Commande Robuste Décentralisée à Base des Systèmes Flous: Application en Robotique", Projet de fin d'étude, Ecole Nationale Polytechnique, Alger, 2001.

- [17] **T. Madani** "Commande Décentralisée à Structure Variable : Application en Robotique", Projet de fin d'étude, Ecole Nationale Polytechnique, Alger, 1997.

- [18] **T. Madani** "Différentes Approches de Commande Décentralisée à Structure Variables Appliquées en Robotique", Thèse de magistère, Ecole Nationale Polytechnique, Alger, 2000.

- [19] **J.P. Lallemand et S. Zeghloul** "Robotique : Aspects fondamentaux", Modélisation mécanique- CAO Robotique- Commande, Masson, Paris, Millan, Barcelone, 1994.

- [20] **John J. Craig** "Introduction to Robotics", Addison-Wesley Publishing Company, Canada, 1989.

- [21] **P.J. McKerrow** "Introduction to robotics", Addison-Wesley Publishing Company, University of Wollongong Australia, 1991.

- [22] **C. Hennessey, A. Jiang and K. Ko** "Scara robot simulator", Project report, ENSC 488, Simon Fraser University.
<http://www.craig.hennessey.com>.
- [23] **M. Chakir** "Modélisation d'un robot manipulateur comportant une boucle cinématique", Projet de fin d'étude, Ecole Nationale Polytechnique, Alger, 2002.
- [24] **C. Vibet** "Robots principes et contrôle", Edition Ellipses, Paris, 1987.
- [25] **P. Coiffet** "La robotique : principes et applications", Hermes, Paris, 1992.
- [26] **R.P. Paul** "Robots manipulators : Mathematics programming and control of robots manipulators", The mit Press, Combridge, 1986.
- [27] **F.L. Lewis, C.T. Abdallah and D. M. Dawson** "Control of robots manipulators", 1993.
[http:// 155.69.254.10 / users / visc / www / pub.html](http://155.69.254.10/users/visc/www/pub.html).
- [28] **Armstrong, B., O. Khatib and J. Burdick** "The explicit dynamic model and inertial parameters of the Puma 560 arm.", In Proc. 1986 IEEE Int. Conf. On Robotics and Automation, S an Francisco, CA.
- [29] **L. L. Whitcomb** "Notes on the dynamics of rigid-body open kinematic chains", The Jhon Hopkins University, 2001.
<http://robotics.me.jhu.edu/~llw/courses/me530646>.
- [30] **Jörg Mareczek and al.** "Robust global stabilization of underactuated 2-DOF manipulator R2D1", Proceeding of the 1998 IEEE.
<http://rs.tu-berlin.de/wwwrt/papers/icra982.pdf>.
- [31] "Chapiter 4 : An introduction to control theory for lagrangian systems", January 2002.
[http:// mast.queensu.ca / ~math 439 / ps / chapiter 4.pdf](http://mast.queensu.ca/~math439/ps/chapiter4.pdf).
- [32] **T. Koseka** "CS 685-Lecture Notes : Position, control and trajectory tracking".
[http:// cs.guess.edu / ~Kosecka / cs685 / control intro.pdf](http://cs.guess.edu/~Kosecka/cs685/controlintro.pdf).
- [33] **R. Gourdeau and al.** "Computed torque control of robots without joint velocity mesurments", IEEE canadian conference computer engineering (CCECE), Edmonton, Canada, 1999.

<http://chemey.chee.queensu.ca/~blouin/publication.eng.htm>.

- [34] **G. Lacruox** “Actionneurs électriques pour la robotique et les asservissements”, Technique et Documentation, Paris, 1985.

- [35] **J. C. Lecoufle-Senlis** “Actinneurs électriques : Technologies et méthodes de choix”, CITEM, 1987.

- [36] **M. Larabi** “Différentes approches de commandes à structures variables : Application à une machine asynchrone”, Projet de fin d’étude, Ecole Nationale Polytechnique, 1997.

- [37] **C. Delannoy** “Programmer en langage C++”, Edition Eyrolles, Paris, 1993.

- [38] **P. Spoljav** “Borland C++ Builder : Méga poche”, Sybex, 1997.

- [39] **B. Garcia** “Introduction à C++ Builder”, ISIMA, 1999.
 [http:// www.dadacouette. free.fr / cours /année 3F2-F3.pdf](http://www.dadacouette.free.fr/cours/année%203F2-F3.pdf).

- [40] **L. Gilles** “L’assembleur avec C++ Builder”, Mai 2001.
 [http:// www.developpez.com /c /](http://www.developpez.com/c/).

- [41] “Chapiter 1 : C++ Builder 4 Client / Server Foundation, Courseware manual”, 1999
 [http://www.ibm.com / services / learning / be / pdf / mini 2002-fr-ebus & AD.pdf](http://www.ibm.com/services/learning/be/pdf/mini2002-fr-ebus&AD.pdf).

- [42] **B. Garcia** “Introduction à Matlab”, ISIMA, 1999.
 [http:// www.dadacouettes.free.fr/ cours /3F2-F3.pdf](http://www.dadacouettes.free.fr/cours/3F2-F3.pdf).

- [43] “Introduction to Matlab : Basic Matlab command and syntaxe”, Information technology Rice University, 2001.
 <http://www.rice.edu/computer/student.html>.

- [44] **P. I. Corke** “Robotics toolbox for user with Matlab”, 1999.
 <http://www.cat.csiro.au/dmt/programs/autom/pic/matlab.html>.

- [45] “Itroduction to OpenGL”.
 <http://www.mesa3d.org>.

- [46] **E. Boyer** "OpenGL et Glut: Une introduction", Institut d'informatiques et mathématiques appliquées de Grenoble.
[http:// www.info.ucl.ac.be/etudiants/Cours/2760/OpenGL/doc/aux.html](http://www.info.ucl.ac.be/etudiants/Cours/2760/OpenGL/doc/aux.html)
- [47] **M. Woo, J. Neider and T. Davis** "OpenGL programming guide", Second edition, Addison Wesley, 1997.
- [48] "OpenGL 1.2 Specification".
<http://www.sgi.com/Software/opengl/manual.html>.
- [49] "Glut. Graphics Library Utility Toolkit Specification".
<http://reality.sgi.com/mjk/glut.html>.
- [50] **M. J. Kilgard** "The OpenGL Utility Toolkit (GLUT), Programming Interface", API version 3, Silicon Graphics, Inc, 1996.
[http:// www.opengl.org / developers / documentation / glut /spec3 / spec 3.html](http://www.opengl.org/developers/documentation/glut/spec3/spec3.html).
- [51] **S. Bowling** "PIC 18CXXX / PIC 16CXXX DC servomotor application", DS00696A, Microchip technology. Inc. Chandler, AZ, 2000.
[http:// www.microchip.com](http://www.microchip.com).
- [52] **A. Colot** "PIC downloader V1.2", East. EPFL, 2001.
<http://alexlife.fr.fm>.
- [53] "PIC 16F87X data sheet 28/40-Pin 8-bit CMOS Flash Microcontrollers", DS 30292C, microchip technology. Inc, 2001.
<http://www.microchip.com>.

العمل المقدم من خلال هذه المذكرة يمثل تصميم برنامج تظاهري للأذرع الآلية. قمنا في بداية الأمر باستخراج النماذج الثلاثة للأذرع الآلية بإستعمال مواصفاتها الحقيقية. برمجة كل من النموذج الهندسي المباشر و العكسي مكنت من تحديد موقع المفاعل في أي نقطة من فضاء عمل الذراع. النموذج التحريكي استعمل في تمثيل سلوك مختلف المحاور مع الأخذ بعين الإعتبار تأثير حركية أداة الحركة. منظومة التحكم المستخلصة هي "العزم المحسوب" التي تتركز أساساً على المعرفة الحيدة للنموذج الحركي للذراع الآلية. و آخرًا، قمنا بعرض نظام ربط للتحكم للتحكم بمحرك ذي تيار مستمر، الذي صمم من حول منظومة تحكم مصغرة قابلة للبرمجة لغرض التحكم المستقل لمختلف محاور الذراع الآلية.

كلمات مفتاحية:

برنامج تظاهري- الذراع الآلية - النموذج الهندسي المباشر- النموذج الهندسي العكسي- النموذج التحريكي- التحكم بالعزم المحسوب.

Résumé :

Le but du travail présenté dans cette thèse, est la conception d'un logiciel de simulation des bras manipulateurs. Nous avons commencé par la modélisation de trois types de robots en utilisant leurs paramètres réels. L'implémentation des modèles géométriques directs et inverses a permis la localisation de l'effecteur pour chaque point de l'espace du travail du robot. Le modèle dynamique a été utilisé pour simuler le comportement des différents axes du robot en tenant compte de l'influence de la dynamique des actionneurs. La commande synthétisée pour cette application est celle du couple calculé, elle est basée essentiellement sur la bonne connaissance du modèle dynamique du robot. Enfin, une interface de commande du moteur à courant continu a été présentée, elle est bâtie autour d'un microcontrôleur programmable dans le but de commander indépendamment les différents axes du robot manipulateur.

Mots clés :

Logiciel - bras manipulateur - modèle géométrique direct - modèle géométrique inverse - modèle dynamique - commande par couple calculé.

Abstract :

This work, is a conception of software for arms manipulators, we have started by modelling of three kinds of robots using their real parameters. The implementation of direct and inverse kinematics of the robots allowed the localisation of the end effector in each points of workspace. The dynamics have been used to simulate the behaviour of robots axes considering actuators effects. The synthesised control in this software is the computed torque control one based essentially on the good knowledge of the robots dynamics. Finally, a control interface of brush DC motor have been presented, it's built around a programmable microcontroller in order to control robots axes independently.

Key words:

Software - arm manipulator - direct kinematics - inverse kinematics - dynamic model - computed torque control.