SOMMAIRE

Le présent travail se veut le résumer et l'application des nombreux apprentissages reçus lors de ces années d'étude. Il a été réalisé à l'Ecole Polytechnique de Montréal, dans la section automation et systèmes du département de génie électrique, dans le cadre d'un échange. J'ai voulu ici mettre en application les notions apprises de la mécatronique. C'est pourquoi ce projet regroupe des notions d'automatique, de robotique, d'informatique et de mécanique.

Ce projet fait suite à une étude effectuée l'été dernier sur le robot mobile Pioneer 2, par deux étudiants. Leur projet avait pour but d'implanter différents contrôleurs pour réaliser l'asservissement en parcours de ce robot. Il s'est avéré que le comportement du robot ne satisfaisait pas entièrement les attentes.

Il était donc nécessaire de trouver un nouveau robot dont le comportement, bien connu, puisse répondre à des spécifications précises. Ce robot devait servir aux manœuvres de stationnement, ainsi qu'à d'autres applications où l'asservissement en parcours est nécessaire. Il sera notamment utilisé pour concevoir une équipe de robots joueurs de soccer. Un robot mobile, construit par le comité SAE Robotique de l'École Polytechnique de Montréal, a été repris. Une validation des composantes et des tests en boucle fermée ont permis de caractériser son comportement. Par la suite, les algorithmes de suivi de chemin ont été simulés et implantés expérimentalement. Nos résultats montrent une bonne correspondance entre l'expérimentation et la simulation.

TABLE DES MATIÈRES

REMERCIEMENTS

Je voudrais tout d'abord remercier le professeur Raymond Hanus pour avoir accepté de patronner mon travail malgré les distances.

Je remercie également le professeur Richard Hurteau pour m'avoir offert les moyens de réaliser ce projet. Son intérêt face à l'avancement du projet, ainsi que sa disponibilité m'ont été d'une grande aide tout au long de ce travail.

Je voudrais ensuite remercier toutes les personnes qui ont gravité autour de ce travail, et plus particulièrement :

Le professeur Romano M. De Santis, pour ses précieuses explications en robotique mobile.

Marie-Lyne Brisson et Richard Grenier, tous deux techniciens de la section automation et système, pour leur efficacité et leur gentillesse.

Julien Beaudry pour sa patience à m'expliquer les détails de l'architecture du robot, et Francis Mailhot pour son « support technique » assuré via Internet. Merci à eux deux pour leur aide et leurs encouragements.

Tous ceux et celles, qu'il serait trop long de citer ici, et qui ont apporté leur pierre à ce travail, par un conseil, un encouragement, ou un simple moment de détente.

Je voudrais profiter de ce rapport pour remercier l'École Polytechnique de Montréal pour l'excellente année d'échange que j'y ai effectuée, ainsi que toutes les personnes de l'École Polytechnique de Bruxelles qui ont permis cet échange.

Un grand merci également à toute ma famille, et particulièrement ma Mère, pour son soutient moral indispensable durant ces longues années.

Enfin, un merci tout spécial à Benoît, avec qui j'ai pu partager les pires et les meilleurs moments qu'a apportés la réalisation de ce projet.

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SYMBOLES

l : distance entre les roues motrices du robot (en m)
R : rayon des roues motrices du robot (en m)
m : masse du robot (en kg)
F_f : force de frottement du robot sur le sol (en N)
μ_c : coefficient de frottement cinétique des roues du robot sur le sol
K_m : gain du système (en unité de commande/m.s^{-1})
τ_m : constante de temps du système (en s)
K_P : gain proportionnel du contrôleur
K_D : gain dérivé du contrôleur
K_I : gain intégral du contrôleur
K_{vff} : gain anticipatif du contrôleur
θ : orientation du robot par rapport au repère fixe (en rad)
(x_P, y_P) :position du centre de masse du robot, par rapport au référentiel fixe (en m)
V_u : vitesse tangentielle du robot (en m.s^{-1})
V_w : vitesse transversale du robot (en m.s^{-1})
Ω : vitesse angulaire du robot (en rad. s^{-1})
q : vecteur de coordonnées généralisées
α : vecteur de vitesses généralisées
J : Jacobien
M : matrice de masse
W : matrice de vitesse angulaire
F_{u_1} et F_{u_2} :forces de propulsion appliquées par les moteurs aux roues motrices
w_a : vecteur des forces actives
τ : vecteur des forces généralisées
D : matrice d'inertie
C : matrice de Coriolis
l_{os} : erreur latérale de position (en m)
θ_{os} : erreur d'orientation (en rad)
λ_{os} : erreur longitudinale de position (en m)

INTRODUCTION

L'exécution automatique des manœuvres de stationnement a fait l'objet de plusieurs projets au sein de la section automation et systèmes du département de génie électrique. Nombre de simulations ont été réalisées, mais les expérimentations effectuées n'ont jamais été entièrement satisfaisantes. Les derniers essais, réalisés l'été dernier sur le robot Pioneer 2, ont mené aux mêmes conclusions.

L'idée de concevoir un robot et de caractériser entièrement son comportement s'est avérée nécessaire pour pouvoir réaliser des expériences plus poussées. Afin de partir d'une certaine base modulable qui avait déjà fait ses preuves, un robot construit par le comité SAE Robotique de l'École Polytechnique a été repris. L'élément le plus important de sa structure, et le plus spécifique au contrôle du véhicule a été changé : deux nouvelles cartes de contrôle de mouvement (ACS-Tech80 et Ajeco) ont été achetées afin de remplacer la carte qui équipait le prototype.

Ce projet a consisté à valider les composantes du robot, et à étudier les performances des cartes de contrôle. Pour ce faire, les composantes ont été passées en revue, et leur validation a été effectuée. Une modélisation dynamique du système a permis de calculer les paramètres du robot, et de les comparer à ceux obtenus par des tests expérimentaux. Ces derniers ont été utilisés afin de régler le contrôleur. Des tests en boucle fermée ont permis de valider ces gains. Enfin, le suivi de chemin a pu être traité, en simulation et en expérimentation.

Le chapitre I présente la structure électromécanique du robot mobile. Ce chapitre est partagé en quatre paragraphes qui décrivent, par type, les composantes du robot. Le premier concerne la mécanique, et le châssis, les roues et les moteurs sont passés en revue afin d'être validés. Le deuxième paragraphe décrit les parties électronique et électrique du robot. Celles-ci comprennent les circuits permettant de véhiculer l'information de l'utilisateur vers les moteurs, et vice versa (génération et acquisition des signaux, contrôle proprement dit…), ainsi que ceux nécessaires à l'alimentation du robot. Le troisième paragraphe expose la structure informatique implantée sur le robot et qui permet aux utilisateurs de lui communiquer des tâches spécifiques. Enfin, dans le dernier paragraphe, un diagramme bloc récapitule les fonctionnalités du système.

Le chapitre II concerne la modélisation du système. Celle-ci est séparée en deux parties : une partie théorique où les paramètres du système sont calculés, et une partie expérimentale où les paramètres sont mesurés. Dans ce second paragraphe, la procédure et les résultats du réglage du contrôleur sont présentés.

Le chapitre III expose les méthodes et les résultats des tests en boucle fermée, et ce, tant en simulation qu'en expérimentation. Les deux approches sont comparées dans cette seconde partie. L'ajustement et l'influence des gains y sont aussi traités.

Le suivi de chemin constitue le quatrième et dernier chapitre. Il est composé de trois parties : la première décrit les objectifs et les moyens utilisés pour réaliser ce suivi de chemin. Les deux autres paragraphes concernent les méthodes et les résultats obtenus lors de ces tests, en simulation, d'une part, et en expérimentation de l'autre.

CHAPITRE I : DESIGN ÉLECTROMÉCANIQUE

Ce premier chapitre présente les différentes composantes, tant matérielles que logicielles, du robot. Cette description s'étend sur quatre parties : une première partie décrit la mécanique du système : l'architecture globale ainsi que les éléments principaux de la structure du robot y sont esposées. Vient ensuite la description des parties électroniques et électriques. La structure informatique du robot est décrite dans la troisième partie. Pour finir, un diagramme bloc récapitule le mode de fonctionnement du système robotique.

1. PARTIE MÉCANIQUE

1.1. Architecture globale

Le véhicule réalisé pour ce projet est un robot à vitesse différentielle symétrique: il comporte deux moteurs couplés à deux roues motrices. Ces dernières sont situées de part et d'autre du robot, sur l'axe de symétrie transversal du robot. Deux roues libres, situées à l'avant et à l'arrière du robot, permettent de le stabiliser.

L'architecture a été réalisée la plus symétrique possible afin de garder le centre de masse le plus près de l'axe des roues motrices, ce qui permet des performances intéressantes en rotation ainsi que des simplifications du modèle dynamique.

Deux batteries sont situées de part et d'autre des moteurs. Comme elles constituent la majorité du poids total du robot, elles sont placées le plus près du sol afin de placer le centre de masse du robot le plus bas possible pour assurer une meilleure stabilité dans les virages.

L'électronique de contrôle, entièrement modulaire, se trouve sur une plaque amovible au dessus de l'ensemble batteries/moteurs. Leur poids étant négligeable par rapport à celui des batteries, la hauteur de cette plaque n'est pas importante.

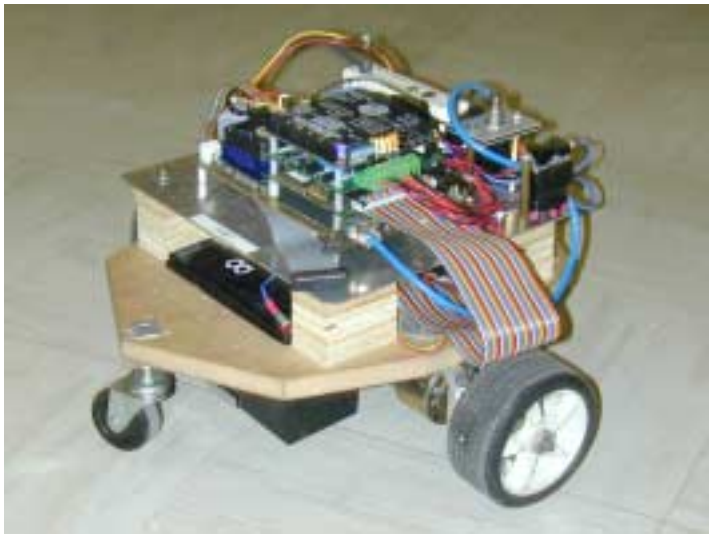


Figure 1.1 *Photo du robot*

1.2. Validation du choix des roues

Le robot possède deux roues motrices et deux roues libres. Les premières sont couplées à deux moteurs, tandis que les autres assurent la stabilité du robot.

1.2.1. Choix des roues motrices

Le choix des roues motrices est important car les performances du robot dépendent de leur dimension. Les roues et les moteurs doivent être choisis conjointement, puisque vitesse, accélération et couple fournis par les moteurs dépendent du rayon des roues motrices. La vitesse est directement proportionnelle au rayon des roues, tandis que l'accélération dépend de l'inverse de celui-ci. Il est donc impossible d'obtenir une vitesse élevée ainsi qu'une accélération élevée, selon le choix des roues motrices. Il est alors nécessaire d'établir un compromis lors de ce choix. Les roues choisies ont un rayon de 5.25 cm, ce qui satisfait les spécifications souhaitées. Ce choix allant de pair avec le choix des moteurs, le lecteur peut se référer au paragraphe suivant pour le détail des calculs ayant mené à cette décision.

Outre la dimension, le matériau de fabrication des roues doit aussi être choisi avec attention: les roues motrices doivent être suffisamment adhérentes au sol, mais leur point de contact avec ce dernier doit être le plus fin possible afin de limiter l'influence de la force de friction.

Nous verrons, dans la validation des moteurs, que des roues d'un plus grand diamètre ont été choisies. De plus, elles sont gonflables, ce qui offre une plus grande souplesse d'utilisation.

La technique de fixation utilisée est aussi à considérer. Les fixations actuelles du prototype (directes à l'axe du moteur), entraînent un cisaillement trop grand subi par cet axe. Il est donc conseillé de répartir le poids du robot sur l'axe de part et d'autre de la roue.

De plus, et nous en avons fait l'expérience, les joints de fixation utilisés n'étaient pas de bonne qualité. Tout ceci a entraîné des rotations irrégulières des roues, tant autour de leur axe que dans le plan des roues. Lors des essais, ces problèmes mécaniques ont créé énormément de bruit sur les signaux, et les courbes tracées s'en sont vues fort entachées. Il est donc de mise, lors de la construction du prochain robot, de veiller à corriger ces différents points.

1.2.2. Choix des roues libres

Les roues motrices se situant au milieu de la structure, les roues libres sont situées à l'avant et à l'arrière du robot. Quelques conditions pratiques ont été respectées afin de déterminer les caractéristiques de ces roues.

Etant donné qu'elles ne servent qu'à stabiliser le robot, aucune de leur caractéristique ne doit influencer les spécifications du robot. Elles ne doivent donc entraîner aucune force de frottement. Elles doivent être petites et fines afin de permettre des rotations aisées autour de leur axe d'attache. Leur fixation à la base mécanique du robot est telle qu'elles peuvent tourner librement autour de cet axe d'attache. Il faut de plus prévoir des possibilités d'ajustement en hauteur afin de prévoir un éventuel changement des roues motrices. Elles ne sont d'ailleurs pas réglées à la même hauteur que les roues motrices. Elles sont moins basses afin d'empêcher la configuration où seules les roues folles touchent le sol.

Toutes ces conditions ayant été respectées, le choix des roues libres a été validé.

1.3. Validation du choix des moteurs

Les moteurs doivent répondre aux spécifications souhaitées pour le robot. Il faut donc tenir compte de plusieurs paramètres, dont, principalement, le diamètre des roues, le rapport des engrenages et les vitesse et accélération maximales désirées.

Les moteurs choisis sont des moteurs à courant continu avec balais. Le modèle est le GM 9236SO15 de la compagnie *Pittman*. Sa fiche technique se trouve à la référence [8] (disponible sur le CD d'annexe).

Le tableau suivant reprend ses principales caractéristiques, tirées de [8], dont nous aurons besoin pour effectuer ladite validation:

Paramètre	Symbole	Valeur
Rapport d'engrenage	n	5,9:1
Couple maximal en régime continu	T_C	0,339 N.m
Vitesse sans charge	$\omega_{roue.max}$	82,7 rad.s^{-1}
Couple maximal absolu	T_{PK}	2,02 N.m
Constante de couple	K_T	0,046 N.m.A^{-1}
Constante de force contre-électromotrice	K_E	0,046 V.rad^{-1}.s^{-1}
Résistance interne	R_T	2,49 Ω
Tension de référence	E	24 V

Tableau 1.1 *Paramètres des moteurs*

Les paramètres du robot sont aussi nécessaires pour effectuer cette validation. Ils sont repris dans le tableau ci-dessous :

Paramètre	Symbole	Valeur
Masse du robot	m	15 kg
Distance entre les roues motrices	l	0,4 m
Rayon des roues motrices	R	5,25 cm
Coefficient de friction des roues au sol	F_f	0,02

Tableau 1.2 ***Paramètres du robot***

Présentons tout d'abord les formules qui seront utiles afin de valider le choix des moteurs.

La vitesse linéaire maximale sans charge est donnée par:

$$V_{max} = \omega_{roue_{max}}.R$$

Le couple nécessaire à l'accélération est exprimé par le produit de la résultante des forces appliquées à la roue par le rayon de celle-ci:

$$T_{acc} = F_{rés}.R$$

La résultante des forces appliquées à chaque roue comporte deux forces : la force de friction due au contact avec le sol, et le poids du robot. Si on considère que chaque roue supporte la moitié de la masse du robot, la force due au poids du robot s'exprime par :

$$P = \frac{ma}{2}$$

Le couple nécessaire à l'accélération s'écrit alors:

$$T_{acc} = R\left(F_f + \frac{ma}{2}\right)$$

F_f , la force de friction, s'écrit :

$$F_f = \mu_c N$$

μ_c est le coefficient de frottement cinétique. Une valeur égale à 0.02 a été choisie, ce qui correspond au coefficient de friction d'une roue sur le sol. N est la force normale appliquée à la roue. C'est, dans le cas présent, la moitié du poids du véhicule.

La friction peut être calculée. Elle vaut :

$$F_f = 0.02\frac{mg}{2} = 0.02\frac{15kg*9.81m.s^{-1}}{2} = 1.47N$$

La valeur de T_{acc} étant donnée dans les spécifications du moteur ($T_{acc} = T_{PK}$ considérée après l'engrenage), l'accélération maximale que peut fournir le moteur est déduite de ce qui précède, et s'écrit:

$$a = \frac{2}{m}\left(\frac{T_{acc}}{R} - F_f\right)$$

En reprenant les valeurs numériques données dans les tableaux 1.1 et 1.2, nous pouvons effectuer l'application numérique pour les moteurs choisis.

Les résultats sont les suivants :

- Une vitesse maximale de :

$$V_{max} = 82.7\ rad.s^{-1}*0.0525\ m$$
$$\Rightarrow \quad V_{max} = 4.34\ m.s^{-1}$$

- Et une accélération maximale de :

$$a_{max} = \frac{2}{15\,kg}.\left(\frac{2.02\,N.m}{0.0525\,m} - 1.47\,N\right)$$

$$\Rightarrow \quad a_{max} = 4.93\ m.s^{-2}$$

Nous constatons donc que les moteurs et les roues choisis satisfont tout à fait les critères de performance recherchés pour le robot, puisque ces derniers sont établis à :

$$V_{max} = 2\,m.s^{-1} \quad \text{et} \quad a_{max} = 2\,m.s^{-2}\,.$$

Néanmoins, il était intéressant de comparer ces résultats à ceux qu'on pourrait obtenir en changeant les dimensions des roues motrices et/ou le rapport d'engrenage des moteurs. D'autres calculs ont alors été effectués pour deux diamètres de roue (5,25 cm et 8,25 cm), ainsi que deux tailles d'engrenage (5.9:1 et 11.5:1). Pour le moteur avec un engrenage 11.5:1, nous avons repris de [8] les spécifications qui nous intéressaient pour effectuer le calcul :

$$\omega_{roue_{max}} = 44.4\,rad.s^{-1} \quad \text{et} \quad T_{acc} = 2.147\,N.m$$

Les résultats sont répertoriés dans le tableau suivant:

Rayon des roues motrices	5,25 cm		8,25 cm	
Rapport d'engrenage	5,9:1	11,5:1	5,9:1	11,5:1
Vitesse linéaire maximale	4,34	2,33	6,82	3,66
Accélération maximale	4,93	5,25	3,07	3,27

Tableau 1.3 *Validation des moteurs*

Après analyse des résultats obtenus, la combinaison retenue ($R = 5.25\,cm$, $n = 5.9:1$) est jugée assez bonne.

La combinaison $R = 5.25\,cm$, $n = 11.5:1$ est un peu limite car la vitesse maximale que peut fournir le moteur vaut seulement 2.33 $m.s^{-1}$. Cette valeur est tout juste supérieure (de 0.33 $m.s^{-1}$) aux critères de performance recherchés, ce qui n'en fait pas une bonne solution.

La combinaison $R = 8.25\,cm$, $n = 5.9:1$ offre, elle, des vitesse et accélération bien supérieures aux valeurs désirées. Une telle performance n'est pas nécessaire pour l'utilisation du robot qui va être faite.

La dernière possibilité ($R = 8.25\,cm$, $n = 11.5:1$) apparaît comme étant la meilleure, car les vitesse et accélération maximales fournies sont suffisamment supérieures, mais pas trop, aux spécifications. Ces roues et ce rapport d'engrenage ont été choisis pour construire le nouveau robot.

2. PARTIES ÉLECTRONIQUE ET ÉLECTRIQUE

2.1. Partie électronique

La structure électronique a été conçue de manière modulaire afin de pouvoir être modifiée et/ou complétée à tout moment. Elle est située sur une plaque au-dessus de l'ensemble batteries/moteurs.

Elle est constituée tout d'abord d'une unité de traitement informatique nécessaire pour implanter les algorithmes. La structure contient aussi des circuits électroniques permettant de lire les encodeurs des moteurs et de générer des signaux de commande appropriés. Afin de transmettre la commande désirée aux moteurs, un circuit amplificateur est intégré à l'électronique de contrôle. Enfin, conformément à la fonctionnalité du robot, un système permettant l'asservissement des moteurs complète cette structure électronique.

Figure 1.2 *Photo de l'électronique de contrôle embarquée*

2.1.1. Ordinateur embarqué et disque dur

L'ordinateur choisi est le modèle Viper 830 de Kontron Communications. Malgré sa taille (moitié de la taille normale), cette carte possède une architecture similaire à celle d'un ordinateur de bureau. Elle incorpore un processeur Intel Celeron 566 MHz, de la mémoire vive (128 M), un processeur graphique, un contrôleur Ethernet…

A cela a été ajouté un micro drive de chez IBM. Ce dernier possède une meilleure résistance aux chocs et aux vibrations mécaniques qu'un disque dur standard de bureau. De plus, il consomme peu d'énergie, occupe très peu de place, et sa capacité de 1G suffit aux besoins.

2.1.2. Signaux de commande

En raisons des nombreux avantages que nous verrons plus loin, nous avons choisi d'utiliser des signaux de commande PWM (Pulse Width Modulation ou Modulation à largeur d'impulsion). La modulation à largeur d'impulsion est une technique puissante pour contrôler des circuits analogiques. En effet, la tension fournie des batteries n'est pas constante. Elle varie en fonction du temps, et peut prendre n'importe quelle valeur. En raison de sa résolution infinie, n'importe quel bruit sur un signal analogique viendra modifier la valeur du courant. C'est en partie pour cette raison que des signaux PWM sont utilisés sur ce robot.

La méthode de modulation de largeur d'impulsion consiste à introduire des commutations à fréquence plus élevée que la fréquence de sortie, transformant la tension en une suite de créneaux d'amplitude fixe et de largeur variable. On contrôle la vitesse du moteur en modulant la largeur de ces impulsions.

La figure suivante illustre ces notions:

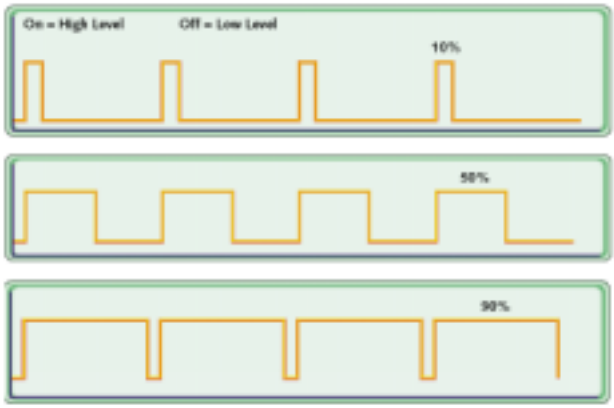


Figure 1.3 *Signaux PWM à largeur d'impulsion variable [10]*

Ces figures montrent des signaux PWM à largeur d'impulsion variable. La première, par exemple, nous montre une sortie PWM à 10% du cycle libre. Le signal est donc « on » pour 10% de la période et « off » pour 90%. Si l'alimentation est de 24V, il résulte un signal analogique de 2.4V.

Les avantages sont les suivants:

- En gardant le signal digital tout au long de son utilisation, les effets du bruit sont limités. Le bruit affectera un signal digital s'il est suffisamment puissant pour transformer un 1 logique en un 0 logique, ou vice versa.
- Passer d'un signal analogique à un PWM peut augmenter la tension d'alimentation fournie, et la plage de vitesses est donc plus étendue. En effet, ces signaux surmontent plus facilement les résistances internes du moteur.
- Les signaux PWM permettent un contrôle très fin du couple et offrent la possibilité de travailler en boucle ouverte.

2.1.3. Circuit amplificateur de signaux

Les signaux de commande, générés par l'électronique afin d'être envoyés aux moteurs, ne sont pas assez puissants pour être exécutés par ceux-ci. Un amplificateur, placé à la sortie de la génération des signaux de commande, permet d'envoyer aux moteurs une commande suffisante.

Voici un schéma de la plateforme de locomotion crée:

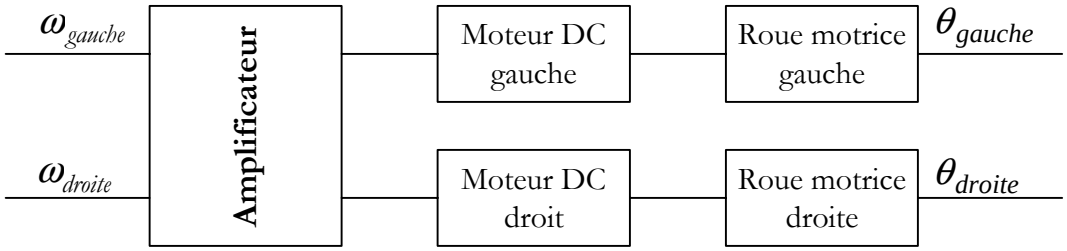


Figure 1.4 *Plateforme de locomotion*

Un circuit amplificateur fabriqué par un étudiant dans le cadre de son projet de fin d'étude, équipait le robot. Nous avons choisi de le changer pour une carte disponible sur le marché. La raison de ce changement est que quelques exemplaires de ce robot doivent être conçus et il était plus facile d'acheter plusieurs circuits que de les construire.

Le circuit choisi est le Servo Booster de la compagnie Ajeco. C'est un amplificateur de puissance PWM à deux canaux, destiné aux moteurs à courant continu. La carte accepte des signaux d'entrée PWM et de direction, et les amplifie à travers un pont en H – MOSFET à haut courant, présenté à la figure suivante. Elle est alimentée par du 24V.

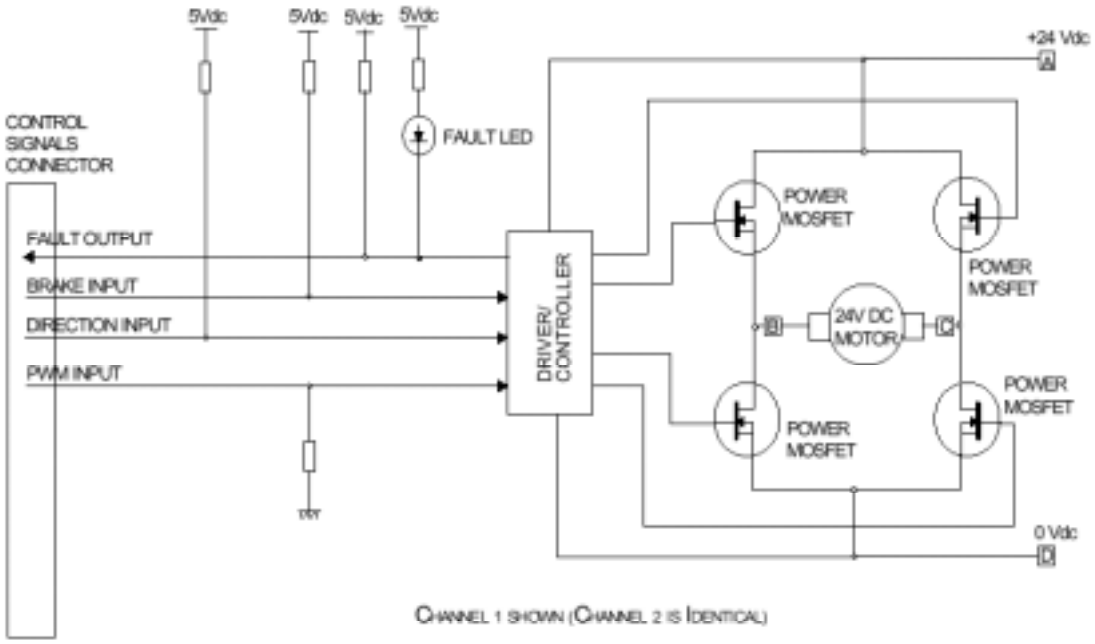


Figure 1.5 *Schéma des ponts en H du Servo Booster de Ajeco [7]*

2.1.4. Liens Rf (radiofréquence)

Un système de radiofréquence est utilisé pour communiquer avec le robot à distance. Il permet aux utilisateurs de se brancher sur un ordinateur de bureau, afin d'avoir accès au nombre de programmes qu'il offre, et de pouvoir se connecter à distance sur le micro drive du robot. Les liens RF offrent aux utilisateurs une connexion sans fil au réseau.

La figure [1.6] présente l'interface utilisateur que permet l'utilisation de ces liens :

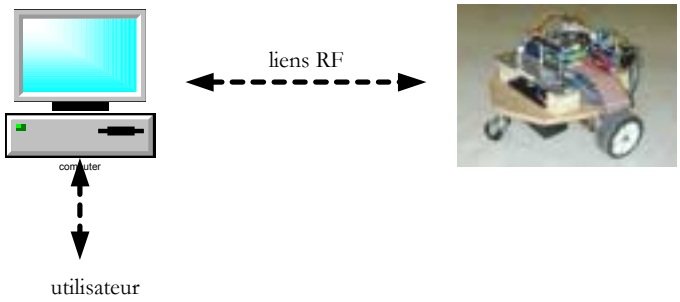


Figure 1.6 *Structure de l'interface robot/utilisateur*

2.1.5. Contrôle des moteurs

Nous arrivons au point clé de la structure informatique : le circuit de contrôle qui fait l'objet des principaux tests de ce travail.

2.1.5.1. Choix du mode de contrôle

Deux possibilités ont été considérées pour réaliser le contrôle des moteurs : un contrôle logiciel, ou une carte de contrôle.

La première solution permet une grande flexibilité car il est alors possible d'ajouter tous les algorithmes de contrôle désirés. Néanmoins, la programmation de ces algorithmes peut rapidement devenir une tâche délicate afin d'obtenir une rapidité de contrôle suffisante. De plus, il faut y intégrer des circuits électroniques pour lire les encodeurs et générer des signaux de commande. Tout ceci doit posséder une rapidité d'exécution qu'il nous est peut-être difficile d'atteindre.

Nous avons choisi la deuxième possibilité : une carte de contrôle. Cette solution est entièrement matérielle. Elle inclut des circuits spécialement conçus pour le contrôle, des circuits électroniques de lecture des encodeurs et de génération de signaux de commande.

Deux cartes de contrôle ont été achetées afin d'être comparées. Dans ce projet, nous nous sommes intéressés à la carte 5950B de ACS-Tech80. Elle permet de gérer quatre axes, tandis que l'autre carte, de la compagnie Ajeco, ne peut en contrôler que deux et n'offre pas de fonctionnement en boucle ouverte.

2.1.5.2. Carte de contrôle Acs-Tech80

Nous allons décrire les principales caractéristiques et les fonctions de cette carte. Pour plus de détail, le lecteur est invité à se référer au document [6].

Comme nous l'avons indiqué, cette carte peut contrôler jusqu'à quatre axes (nous n'en utiliserons que deux, pour les moteurs). Elle est le lien direct entre les moteurs et le processeur de mouvements. Elle peut recevoir ou envoyer des signaux analogiques ou PWM. Ces signaux sont envoyés via un ruban de câble TB50N-S, et son interface est compatible avec le format PC/104.

Nous allons décrire ici le principal élément de cette carte: la puce de contrôle de mouvement, et plus précisément le modèle MC1401A de la compagnie PMD Corp. (par la suite, cette puce est appelée sous le nom de « puce PMD »). La figure ci-dessous, extraite de [6] décrit le fonctionnement de cette puce:

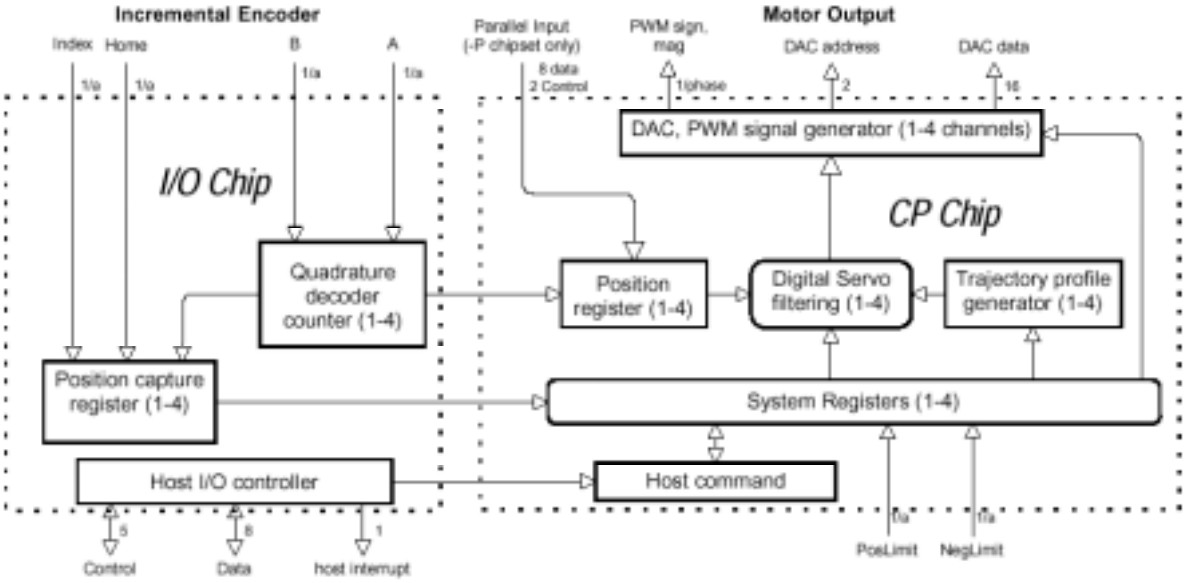


Figure 1.7 *Diagramme interne de la puce PMD*

Cette puce fonctionne selon deux modes:

- Mode « boucle ouverte »: ce mode est utilisé pour des opérations de contrôle directe des axes, et permet de mesurer les différents paramètres du système.
- Mode « boucle fermée »: c'est le mode de fonctionnement « normal » de la puce. Il génère un profil de trajectoire et réalise la fermeture de la boucle d'asservissement.

Ce second mode de fonctionnement, que la figure [1.5] illustre, est davantage décrit dans ce qui suit.

Chaque module rattaché à un axe reçoit la position actuelle de l'axe par l'intermédiaire d'un encodeur incrémental (« Quadrature decoder counter »). Cet encodeur fait l'acquisition de deux signaux A et B en quadrature de phase. En plus de donner la position de l'axe, ils renseignent, dans le cas du robot mobile considéré, du sens de rotation des roues. Ces signaux sont alors envoyés au contrôleur proprement dit (« Digital Servo filtering »), et sont comparés aux signaux émis par le générateur de trajectoire (« Trajectory profile generator ») et correspondants à la trajectoire désirée. La commande à envoyer est générée sous forme d'un signal digital ou PWM (« DAC, PWM signal generator »); elle est amplifiée par un circuit externe, puis elle est envoyée aux axes concernés à chaque mise à jour de la boucle.

Le générateur de trajectoire effectue des calculs pour déterminer à chaque boucle la position, la vitesse et l'accélération du système.

Il existe quatre profiles de trajectoire :

- *S-curve point à point:* pour ce mode, quatre paramètres sont spécifiés: la position désirée, les vitesse et accélération maximales et le « jerk » (secousse).

Le profil S-curve conduit l'axe au jerk désiré jusqu'à atteindre l'accélération maximale (phase I). Le jerk retourne à zéro, et l'accélération garde sa valeur maximale (phase II). Durant la phase III, une valeur opposée au jerk spécifié est appliqué pour atteindre la vitesse maximale avec une accélération nulle. La phase IV, exécutée à vitesse maximale constante permet d'atteindre la position désirée. Les phases V à VIII sont symétriques aux trois premières. Si la position est atteinte trop vite, les phases II et VI n'auront pas lieu.

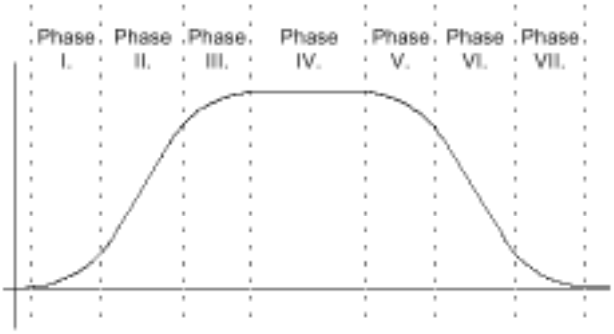


Figure 1.8 *Profil S-Curve [6]*

- *Trapézoïdale point à point:* pour ce mode, trois paramètres sont spécifiés: la position désirée, la vitesse maximale et l'accélération désirée.

La trajectoire est exécutée en appliquant aux axes l'accélération désirée jusqu'à la vitesse maximale où elle reste constante pour atteindre la position désirée. Une décélération se produit quand la position désirée a été atteinte.

Figure 1.9 *Profil trapézoïdal [6]*

- *Velocity contouring:* pour ce mode, deux paramètres seulement sont spécifiés: la vitesse maximale et l'accélération désirée.

C'est un contrôle en vitesse proprement dit. Le profil trapézoïdal est utilisé, mais ici, le système demeure à vitesse constante jusqu'à être interrompu. C'est ce mode qui est sélectionné lors de l'utilisation de la carte en boucle fermée dans les prochains tests.

Figure 1.10 *Profil de vitesse [6]*

- *Electronic gear:* pour ce profil, l'utilisateur spécifie un seul paramètre: le rapport d'engrenage. La position désirée est générée en appliquant ledit rapport à la position courant de l'axe.

Le contrôleur utilisé est un contrôleur PID (proportionnel, intégral, dérivé) avec gain anticipatif. Son schéma, repris de [6], est présenté ci-dessous:

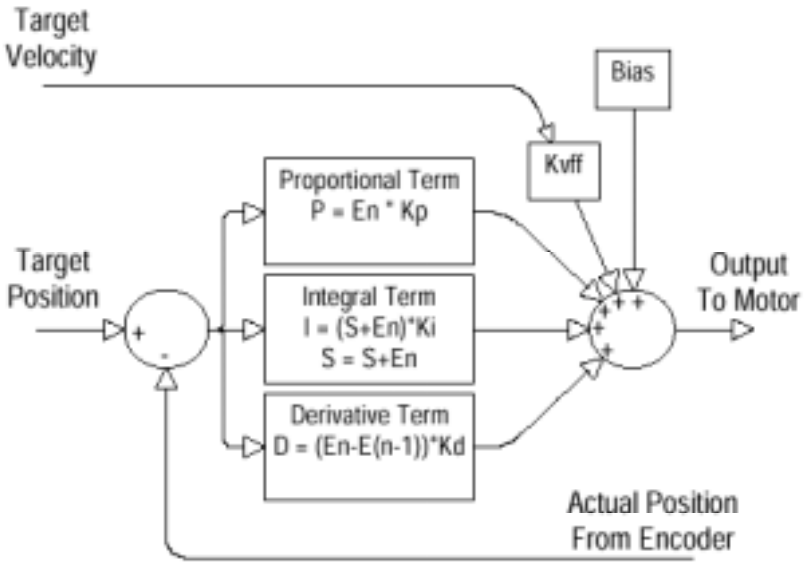


Figure 1.11 *Structure du contrôleur de la carte de contrôle [6]*

La position actuelle des encodeurs recueillie par la carte est comparée à la position désirée introduite par l'utilisateur ou générée selon le profil de trajectoire choisi. L'erreur de position passe à travers un PID classique comportant trois termes : proportionnel, dérivé et intégral. À sa sortie, le signal est ajouté au produit de la vitesse désirée par le gain anticipatif « velocity feed forward ». Il est également ajouté à une valeur « Bias », utilisée pour ajuster l'offset. Remarquons de suite que ce dernier gain n'est pas utilisé dans les essais qui suivent.

La connaissance de l'influence de chacun des gains va permettre de mieux les initialiser:

- Gain proportionnel K_P : plus ce gain est élevé, plus le système est sensible aux erreurs instantanées de position et plus le système sera « agressif » pour les corriger. En d'autres termes, c'est la « proportion » avec laquelle le système répondra instantanément aux erreurs de position.
- Gain dérivé K_D : plus ce gain est élevé, plus le système répondra de manière rapide à l'erreur instantanée de position. Il contrôle la pente de l'erreur instantanée, c'est-à-dire la rapidité du système.
- Gain intégral K_I : plus ce gain est élevé, plus vite le contrôleur va accumuler l'erreur. Ce gain augmente l'ordre du système en introduisant un pôle. Il réduit la stabilité du

système, réduit l'erreur, mais ralentit la vitesse de réponse du système. Il a surtout de l'influence en cas de perturbations appliquées au système.

- Gain anticipatif $K_{v_{ff}}$: ce gain définit le gain global de la vitesse. Plus il sera élevé, plus le système répondra avec force. Il permet de compenser le retard. Le manuel d'utilisation de la carte conseille de le mettre égal à l'inverse du gain du système.

2.2. Partie électrique

Les besoins en tension des différents composants du robot sont les suivants :

- Les moteurs et le Servo Booster de Ajeco nécessitent 24V.
- Tous les éléments de l'électronique de contrôle (ordinateur embarqué, microdrive, carte de contrôle) ainsi que les liens RF, nécessitent 5V.

L'alimentation du robot est donc assurée par deux batteries de 12V chacune, et une carte PC/104 Power Supply.

Les batteries choisies correspondent au modèle LC-R127R2P de chez Panasonic. Leur masse est de 2,47 kg chacune. Elles constituent les éléments les plus pesants du robot. Ces batteries sont mises en série afin de fournir 24V nécessaire à l'alimentation des moteurs ainsi que du Servo Booster de Ajeco. Elles ont été choisies tant pour leur puissance que pour l'autonomie qu'elles offrent, soit environ 3 heures.

La carte PC/104 Power Supply transforme la tension d'alimentation du robot en ± 5V pour assurer l'alimentation de l'électronique de contrôle et des liens RF.

L'alimentation du robot est assurée par l'intermédiaire d'un interrupteur à trois positions: il permet de choisir de fonctionner soit sur les batteries, lors des essais au sol, soit sur une source à courant continu externe, pour des essais roues levées. Cette source permet aussi de recharger les batteries. Un bouton d'arrêt d'urgence est aussi prévu afin de prévenir toute catastrophe.

3. PARTIE INFORMATIQUE

3.1. Architecture

Grâce à l'ordinateur embarqué, la programmation des tâches a pu se faire dans un langage de haut niveau. L'implantation d'un système d'exploitation sur l'ordinateur embarqué évite de programmer en assembleur ou en un autre langage de bas niveau.

Le système d'exploitation choisi sur le robot Spinos, le modèle du prototype utilisé pour le présent travail, est Linux, combiné à RT-Linux. Il est en effet de mise d'utiliser un système d'exploitation offrant des possibilités temps réels. Sont donc exclus une grande partie des produits de la gamme Microsoft. RT-Linux a été choisi par les membres de SAE Robotique car ils étaient tous plus ou moins familiers avec ce système d'exploitation. De plus, une bonne partie du code de bas niveau était développée pour cette plateforme.

Néanmoins, ils pensaient depuis peu transférer tous les modules de programmation sur QNXRTP, un système d'exploitation temps réel de la compagnie QNX. L'occasion du prototype s'y prêtant, Francis Mailhot, un étudiant de génie informatique, a été chargé de ce travail dans le cadre de son projet de fin d'études. Il s'est donc occupé dans un premier temps de créer les pilotes permettant d'utiliser, sous QNXRTP, les cartes de contrôle achetées.

3.2. Structure du logiciel

La structure du logiciel comprend deux parties: un module temps réel et le programme de contrôle.

Le module temps réel est pris en charge par la carte de contrôle. C'est elle qui effectue l'acquisition des encodeurs des moteurs, et d'autres capteurs présents sur le robot s'il y a lieu. Nous avons vu qu'elle peut contrôler jusqu'à quatre axes. De plus, les commandes du programme de contrôle lui sont transmises en temps réel. La communication entre le module temps réel et le programme de contrôle se fait grâce à des FIFOs (first in first out).

Le programme de contrôle, quant à lui, est responsable de planifier les tâches attribuées au robot. L'étudiant dont nous venons de parler travaillait en parallèle afin de construire une architecture informatique C++ et orientée objet, comprenant les fonctions de base utiles au robot. Nous n'avons donc pas pu en bénéficier. Néanmoins, n'ayant pas d'appel à des fonctions, la rapidité du programme est probablement augmentée.

4. RÉCAPITULATION: DIAGRAMME BLOC DE L'ENSEMBLE DU SYSTÈME

Le schéma ci-dessous résume la structure complète du système .

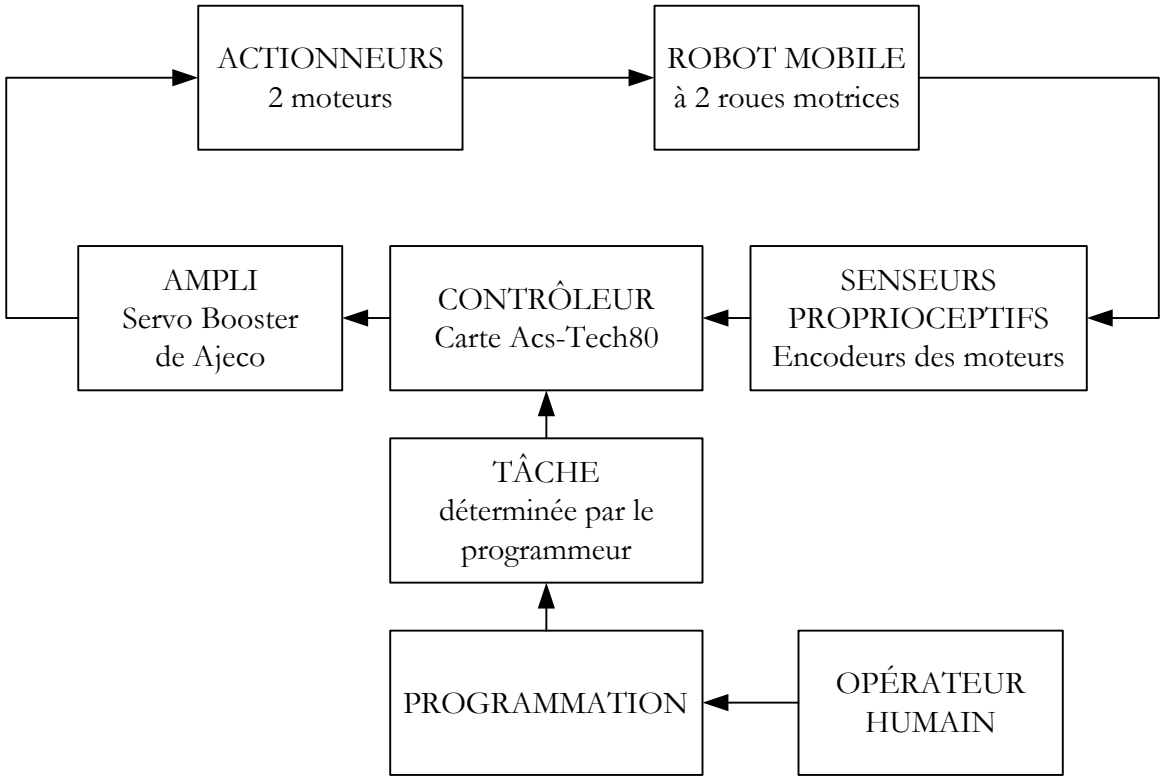


Figure 1.12 *Structure complète du système*

Ceci résume le fonctionnement global du système: l'opérateur humain, par le biais de la programmation, impose une certaine tâche au robot. Cette tâche est envoyée à la carte de contrôle, laquelle génère une commande sous forme d'un signal PWM. Ce signal est amplifié afin d'être traité par les actionneurs du système, c'est-à-dire les moteurs. Cela confère le mouvement désiré au système, le robot mobile. Des senseurs effectuent l'acquisition des encodeurs des moteurs, et la carte de contrôle referme la boucle.

La prise en main de tout ce matériel a été une partie lourde lors de ce travail. Les validations des composantes et le branchement de la carte de contrôle nous ont aidé à mieux comprendre l'architecture du système. Néanmoins, la prise en main véritable s'est effectuée tout au long des essais, par l'utilisation même du robot, grâce ou à cause des bugs informatiques, problèmes techniques et autres.

CHAPITRE II : MODÉLISATION DU SYSTÈME

Après cette validation des composantes du robot, tant nécessaire à la validation en elle-même qu'à la familiarisation avec le robot, une modélisation du système a été réalisée. Elle a été faite selon deux approches : théorique et pratique. Les deux paragraphes de ce chapitre présentent ces approches. Les résultats obtenus par les deux méthodes sont comparés, puis ils sont utilisés pour régler le contrôleur.

1. MODÈLE D'ÉTAT THÉORIQUE DU ROBOT

La procédure de modélisation dynamique du robot est reprise de la référence [2]. Elle est appliquée à un robot mobile à deux roues motrices indépendantes, dont l'axe de rotation passe par le centre de masse du robot.

Le schéma du robot reprenant les différentes notations est présenté ci-dessous:

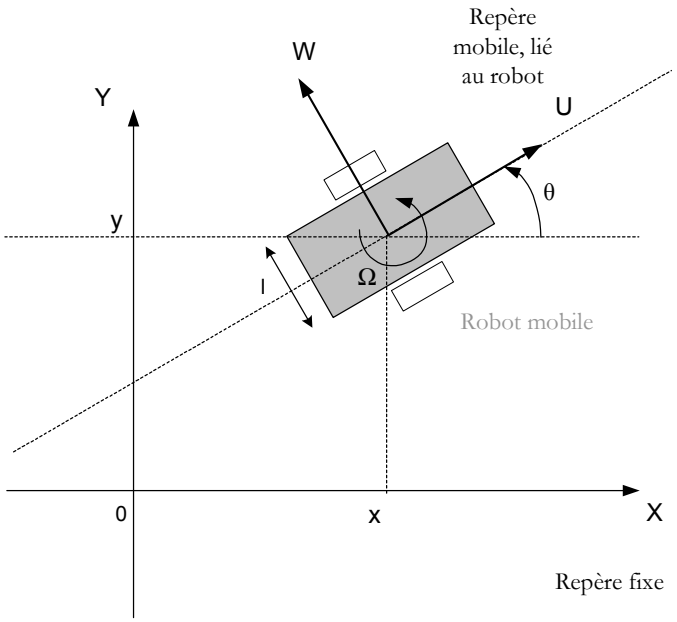


Figure 2.1 *Géométrie du robot mobile*

Un repère avec les axes U et W est lié au robot. La position et l'orientation de ce repère peuvent être représentées par le vecteur :

$$\begin{pmatrix} c \\ \xi \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

Le robot mobile n'est assujetti à aucune contrainte de position/orientation, appelée contrainte holonome. Nous pouvons donc choisir comme vecteur de coordonnées généralisées le vecteur suivant:

$$q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

La relation entre la vitesse du mobile dans le repère fixe, $\begin{pmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{pmatrix}$, et celle dans le repère lié au mobile, $\begin{pmatrix} V_u & V_w & \Omega \end{pmatrix}$, est donnée par :

$$\begin{pmatrix} V_u \\ V_w \\ \Omega \end{pmatrix} = \underbrace{\begin{pmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{J_b} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}$$

Si nous faisons comme hypothèse qu'il n'y a pas de glissement latéral des roues motrices, c'est-à-dire que la vitesse transversale du robot, V_w, est nulle, la vitesse généralisée peut être représentée par le vecteur α :

$$\alpha = \begin{pmatrix} V_u \\ \Omega \end{pmatrix} = \underbrace{\begin{pmatrix} cos\theta & sin\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{J_{nh}} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}$$

Le Jacobien total vaut :

$$J = J_b \, J_{nh}$$

$$\Rightarrow \quad J = \begin{pmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{pmatrix}$$

$$\Rightarrow \quad J = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

La vitesse du centre de masse s'exprime par:

$$\begin{pmatrix} V_{u_{CDM}} \\ V_{w_{CDM}} \\ \Omega_{CDM} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} V_u \\ \Omega \end{pmatrix}$$

Il faut maintenant évaluer les matrices de masse et de vitesse angulaire :

$$M = \begin{pmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & J_0 \end{pmatrix}$$

où m est la masse du robot mobile et J_0 le moment d'inertie du mobile par rapport à l'axe vertical passant par son centre de masse. Si on considère que les batteries reprennent l'ensemble de la masse du robot, ce moment peut être calculé en considérant deux masses ponctuelles situées de part et d'autre du centre de masse, à une distance de 10cm de celui-ci. En se plaçant dans le repère lié au robot, le moment d'inertie recherché s'écrit :

$$\Rightarrow \quad J_0 = m_1 d_{batterie1}^2 + m_2 d_{batterie2}^2$$
$$\Rightarrow \quad J_0 = m\, d_{batterie}^2$$

En effet, $m_1 = m_2 = \frac{m}{2}$ et $d_{batterie1} = d_{batterie2} = d_{batterie}$

La matrice de masse s'écrit alors :

$$M = m \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d_{batterie}^2 \end{pmatrix}$$

La matrice de vitesse angulaire s'exprime simplement par :

$$W = \begin{pmatrix} 0 & -\Omega & 0 \\ \Omega & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Le vecteur des forces actives vaut :

$$w_a = \begin{pmatrix} F_{u_1} + F_{u2} \\ 0 \\ \frac{l}{2}\left(F_{u_1} - F_{u2}\right) \end{pmatrix}$$

où F_{u_1} et F_{u2} sont les forces de propulsion appliquées par les moteurs aux roues motrices.

Tout ceci permet alors de calculer les éléments qui forment le modèle d'état :

- La matrice d'inertie :

$$D = J'MJ$$
$$\Rightarrow \quad D = m\begin{pmatrix} 1 & 0 \\ 0 & d_{batterie}^2 \end{pmatrix}$$

- La matrice de Coriolis :

$$C = J'\left(M\dot{J} + WMJ\right)$$
$$\Rightarrow \quad C = 0$$

- Le vecteur des forces généralisées:

$$\tau = J'w_a$$
$$\Rightarrow \quad \tau = \begin{pmatrix} F_{u_1} + F_{u2} \\ \frac{l}{2}\left(F_{u_1} - F_{u2}\right) \end{pmatrix}$$

Le modèle d'état du mobile est donné par :

$$\begin{cases} \dot{a} = g_0 + g_w\tau \\ \dot{q} = J_{nb}\,q \end{cases}$$

où $g_0 = -D^{-1}C\alpha$ et $g_w = D^{-1}$

C'est-à-dire, en remplaçant chaque inconnue par son expression :

$$\begin{cases} \dot{V}_u = \dfrac{F_{u_1} + F_{u_2}}{m} \\ \dot{\Omega} = \dfrac{\left(F_{u_1} - F_{u2}\right)l}{2J_0} \\ \dot{x} = cos\theta V_u \\ \dot{y} = sin\theta V_u \\ \dot{\theta} = \Omega \end{cases} \qquad [2.1]$$

Il faut maintenant exprimer les forces F_{u_1} et F_{u2} en fonction des tensions de contrôle V_1 et V_2 appliquées à chaque moteur.

Les relations suivantes sont nécessaires :

- Loi d'ohm appliquée au circuit induit de chaque moteur :

$$V_i = R_T I_j + K_E \dot{\theta}_i \qquad [2.2]$$

- Relation entre couple moteur et courant d'induit :

$$\Gamma_i = K_T I_i \qquad [2.3]$$

- Relation entre couple moteur et force motrice :

$$F_{u_i} = \frac{\Gamma_i}{R} \qquad [2.4]$$

Où $\dot{\theta}_i$ est la vitesse angulaire des roues motrices et Γ_i le couple fourni par chaque moteur.

Il suit, en combinant les relations [2.3] et [2.4] :

$$F_{u_i} = \frac{K_T I_i}{R}$$

Nous obtenons alors, en remplaçant I_i par sa valeur extraite de [2.2] :

$$F_{u_i} = \frac{K_T}{R_T R}\left(V_i - K_E \dot{\theta}_i\right)$$

D'où :

$$F_{u_1} + F_{u_2} = \frac{K_r}{R_T R}\left(V_1 + V_2\right) - \frac{K_T K_E}{R_T R}\left(\dot{\theta}_1 + \dot{\theta}_2\right) \qquad [2.5]$$

$$F_{u_1} - F_{u_2} = \frac{K_r}{R_T R}\left(V_1 - V_2\right) - \frac{K_T K_E}{R_T R}\left(\dot{\theta}_1 - \dot{\theta}_2\right) \qquad [2.6]$$

Les vitesses angulaires des roues peuvent s'exprimer en fonction des vitesses linéaire et angulaire du robot. En effet :

$$\begin{cases} V_u = \dfrac{(\omega_1 + \omega_2)}{2} R \\ \Omega = \dfrac{(\omega_1 - \omega_2)}{l} R \end{cases} \Rightarrow \begin{cases} \dot{\theta}_1 + \dot{\theta}_2 = \dfrac{2V_u}{R} \\ \dot{\theta}_1 - \dot{\theta}_2 = \dfrac{l\,\Omega}{R} \end{cases}$$

En utilisant ces expressions dans les relations [2.5] et [2.6] obtenues plus haut, on arrive à :

$$F_{u_1} + F_{u_2} = \frac{K_T}{R_T R}(V_1 + V_2) - \frac{2K_T K_E V_u}{R_T R^2}$$
$$F_{u_1} - F_{u_2} = \frac{K_T}{R_T R}(V_1 - V_2) - \frac{lK_T K_E \Omega}{R_T R^2}$$

En introduisant ces deux dernières équations dans le modèle [2.1], il suit :

$$\dot{V}_u = \frac{K_T}{mR_T R}(V_1 + V_2) - \frac{2K_T K_E}{mR_T R^2} V_u$$
$$\dot{\Omega} = \frac{K_T l}{2J_0 R_T R}(V_1 - V_2) - \frac{l^2 K_T K_E}{2J_0 R_T R^2}\Omega$$

En posant :

$$\begin{cases} K_v = \dfrac{R}{2K_e} \\ \tau_v = \dfrac{mR_T R^2}{2K_T K_E} \end{cases}$$

pour la première relation

et

$$\begin{cases} K_o = \dfrac{R}{K_e l} \\ \tau_o = \dfrac{J_0 R_T R^2}{l^2 K_T K_E} \end{cases}$$

pour la seconde relation

Le modèle [2.1] se réécrit sous la forme :

$$\begin{cases} \dot{V}_u = -\dfrac{1}{\tau_v} V_u + \dfrac{K_v}{\tau_v}(V_1 + V_2) \\ \dot{\Omega} = -\dfrac{1}{\tau_o}\Omega + \dfrac{K_o}{\tau_o}(V_1 - V_2) \\ \dot{x} = cos\theta\, V_u \\ \dot{y} = sin\theta\, V_u \\ \dot{\theta} = \Omega \end{cases}$$

En utilisant la transformée de Laplace, nous obtenons le modèle suivant, formé de deux composantes :

- Une composante dynamique :

$$\begin{cases} V_u(s) = \dfrac{K_v}{1 + s\tau_v}\left[V_1(s) + V_2(s)\right] \\ \Omega(s) = \dfrac{K_o}{1 + s\tau_o}\left[V_1(s) - V_2(s)\right] \end{cases}$$

- Une composante cinétique :

$$\begin{cases} \dot{x} = cos\theta\, V_u \\ \dot{y} = sin\theta\, V_u \\ \dot{\theta} = \Omega \end{cases}$$

Les constantes de ce modèle sont calculées à partir des données des moteurs fournies dans le tableau [1.1] :

$$K_v = 0.57\ m.rad.s.V^{-1}$$
$$\tau_v = 24.32\ s$$
$$K_o = 2.85\ m.rad.s.V^{-1}$$
$$\tau_o = 3.04\ s$$

Ces résultats seront comparés aux valeurs obtenues par voie expérimentale.

2. MODÈLE DYNAMIQUE DU ROBOT

2.1. Evaluation des paramètres du système

Nous venons de voir que le système est représenté par un premier ordre. Sa fonction de transfert est donnée par:

$$\frac{K_m}{\tau_m s + 1}$$

Deux sortes de tests en boucle ouverte ont été réalisés afin de déterminer le gain K_m et la constante de temps τ_m du système : un test en translation et un autre en rotation.

Pour faire ces tests, il était nécessaire d'implanter un petit programme qui envoie aux deux moteurs une commande sous forme d'un pourcentage de PWM, puis qui recueille la valeur des encodeurs. Une commande permet avant tout de configurer les moteurs en mode « boucle ouverte ». Ces données recueillies, il est alors possible de tracer les courbes de vitesse linéaire (test en translation) et angulaire (test en rotation) correspondantes sur Matlab.

Les tests, en translation et en rotation, ont été effectués à différentes vitesses. Néanmoins, nous n'avons pu aller à plus de 50% du PWM, ce qui correspond à 2 m/s. Le courant apporté par les batteries limite cette vitesse.

Les tests effectués à 12.5% se sont révélés inutilisables. Cette vitesse est trop proche de la friction de Coulomb.

Une autre remarque s'impose : lors des essais en translation, le robot ne roulait pas droit. Nous n'avons donc pas envoyé exactement le même pourcentage de PWM aux deux roues afin de compenser cette erreur. En boucle ouverte, il n'existe pas de fonction qui met à jour la vitesse envoyée aux moteurs à chaque boucle. La commande de vitesse n'est donc pas envoyée simultanément aux deux roues. Une roue démarre avant l'autre. Le décalage n'est pas grand, mais il suffit à faire dévier le robot quelque peu.

Les courbes de vitesse des deux tests, en translation et en rotation, sont présentées ci-après.

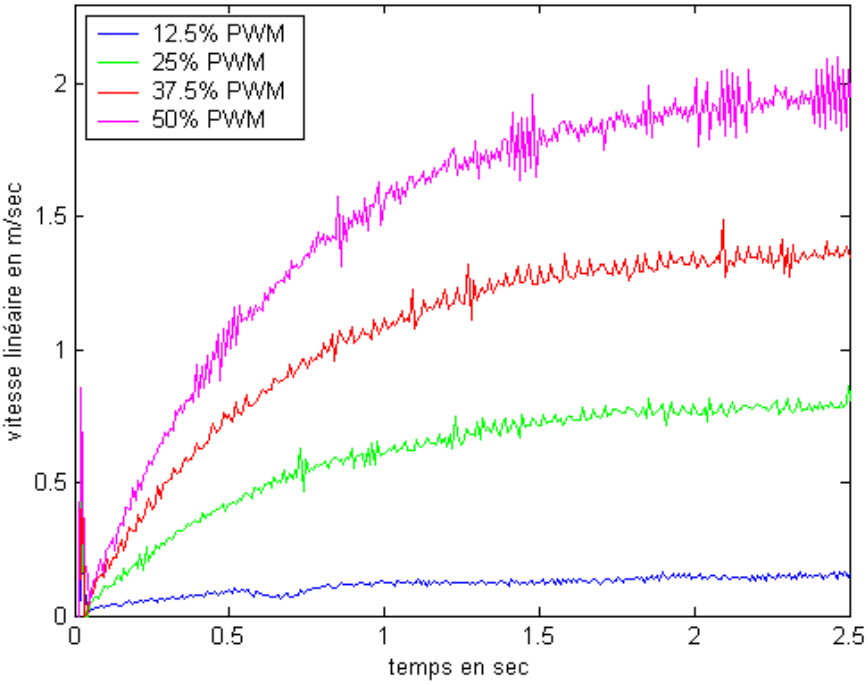


Figure 2.2 *Test en translation en boucle ouverte*

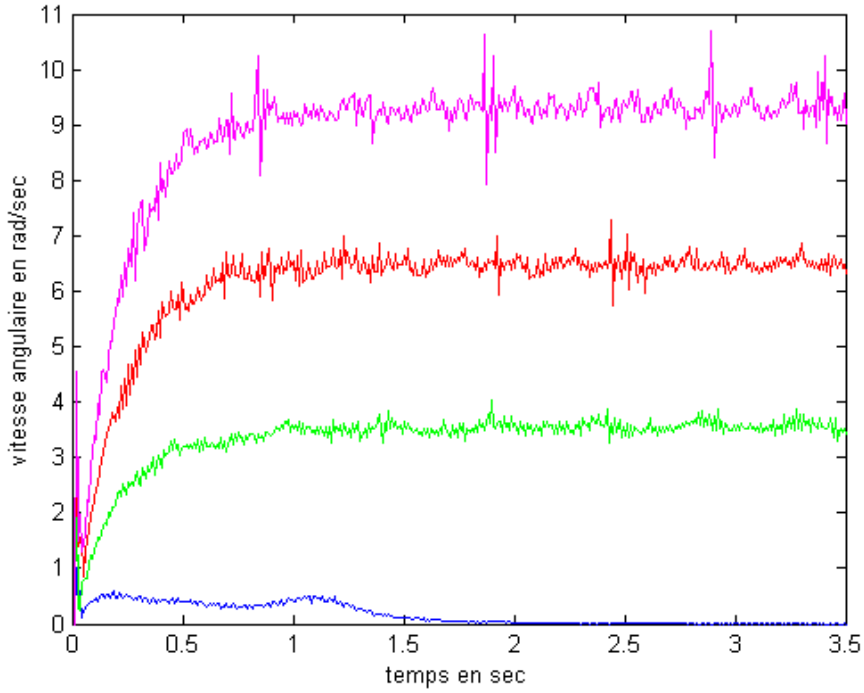


Figure 2.3 *Test en rotation en boucle ouverte*

L'évaluation des paramètres est réalisée selon une méthode graphique:

- K_m correspond à la valeur de l'asymptote à la courbe pour $t \to \infty$.
- τ_m a été évalué de deux manières différentes, afin d'augmenter la précision de la mesure. τ_m a pour valeur l'abscisse de l'intersection de la pente de la courbe avec l'asymptote à celle-ci pour $t \to \infty$. Cette constante de temps correspond aussi à l'abscisse de l'intersection de la courbe avec une droite ayant pour ordonnée une valeur égale à 63% celle du gain.

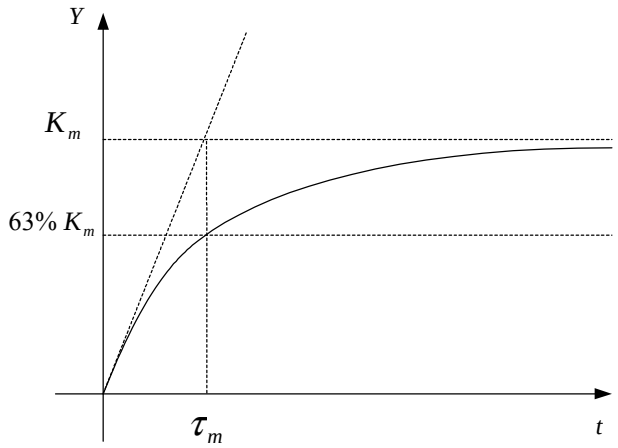


Figure 2.4 *Evaluation des paramètres pour une réponse à l'échelon d'un second ordre*

Les valeurs mesurées sont présentées dans le tableau ci-dessous:

	Commande	**K_m**	**τ_m**
Test en translation	50% PWM	5,06	0,63
	37,5% PWM	5,20	0,63
	25% PWM	5,60	0,63
Moyenne		**5,29**	**0,63**
Test en rotation	50% PWM	4,78	0,06
	37,5% PWM	4,92	0,10
	25% PWM	4,66	0,14
Moyenne		**4,79**	**0,10**

Tableau 2.1 *Résultats des paramètres K_m et τ_m*

Ces résultats tiennent compte de la force de frottement statique (friction de Coulomb). Cette force étant non négligeable, elle a une certaine influence sur les paramètres mesurés. La procédure utilisée pour mesurer la force de frottement statique est assez délicate, mais le résultat

obtenu est considéré comme fiable. Cette force correspond à la plus petite vitesse nécessaire pour faire bouger le robot. Nous avons donc augmenté petit à petit le pourcentage de PWM envoyé, jusqu'à atteindre une valeur qui mette en mouvement le robot. Elle vaut 0xEFF, soit 11.7% de PWM.

Les résultats obtenus pour le comportement en translation et en rotation diffèrent. Le gain et la constante de temps sont plus élevés dans le cas des tests en translation. Le robot répond plus vite à une rotation pure. Mais ce dernier cas ne sera normalement jamais appliqué ou très rarement (pour des demi-tours, par exemple). En effet, le robot sera plus utilisé en translation ou translation/rotation qu'en rotation pure. Les valeurs des paramètres obtenus lors des essais en translation seront donc utilisés dans la suite.

Comparons ces résultats à ceux obtenus lors de la modélisation théorique. Le tableau ci-dessous reprend les valeurs trouvées:

	Méthode	$\mathbf{K_m}$	$\boldsymbol{\tau_m}$
Test en translation	Théorie	0,57	24,32
	Expérimentation	5,29	0,63
Test en rotation	Théorie	2,85	3,04
	Expérimentation	4,78	0,10

Tableau 2.2 *Comparaison des valeurs théoriques et expérimentales obtenues pour les paramètres* K_m *et* τ_m

Les valeurs expérimentales diffèrent énormément des valeurs théoriques obtenues au paragraphe précédent.

Des erreurs apparaissent de chaque côté. D'une part, dans le modèle théorique, la friction n'est pas prise en compte. Le moment d'inertie calculé n'est pas exact, et la masse du robot peut s'écarter quelque peu de la réalité. De plus, les paramètres des moteurs utilisés dans les calculs sont définis pour les moteurs sans charge. Des éléments comme la résistance interne des moteurs ou les couples induits ne sont donc pas corrects. Enfin, aux gains trouvés de manière théorique, il faut multiplier le gain de l'amplificateur qui est inconnu. D'autre part, les mesures effectuées pour la recherche expérimentale des paramètres ainsi que la mesure de la force de friction ne sont pas très précises. La mécanique, comme nous le verrons plus loin, introduit un certain bruit non négligeable dans le tracé des courbes.

Plusieurs tests ont été réalisés afin d'observer la répétitivité du système, et ceci en respectant les mêmes conditions. Les résultats obtenus sont sensiblement identiques.

Les courbes ci-dessous présentent trois essais réalisés à différents moments :

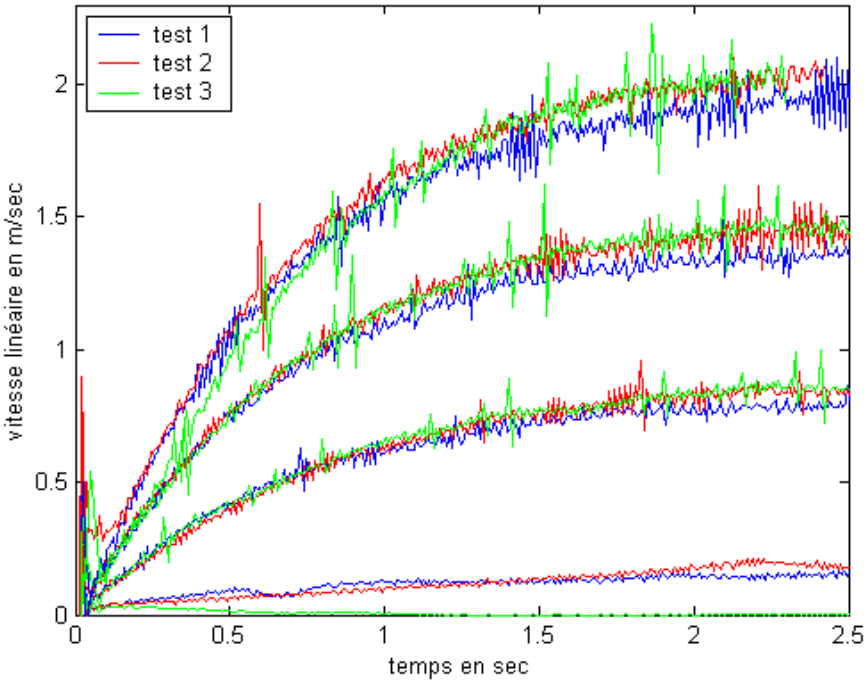


Figure 2.5 *Tests de répétitivité*

Les courbes rouge (test 2) et verte (test 3) se superposent dans la majorité du tracé. Pour le test réalisé à la vitesse la plus grande, les deux courbes ne sont pas confondues au départ, alors qu'elles le sont ensuite. La courbe bleue (test 1) coïncide tout à fait aux deux autres courbes, en ce qui concerne la pente de départ. Néanmoins, en régime, elle s'écarte des deux autres courbes. L'état de charge des batteries doit en être la cause. Il influence considérablement le gain, tandis qu'il n'agit pas sur la constante de temps. Il est donc de mise de veiller à conserver un niveau de charge élevé pour les batteries, ou tout au moins constant.

2.2. Réglage du contrôleur

2.2.1. Description du contrôleur

Le contrôleur de la carte est décrit au paragraphe 2.1.5.2. Il correspond à un PID avec gain anticipatif. En simulation, nous avons utilisé le contrôleur suivant:

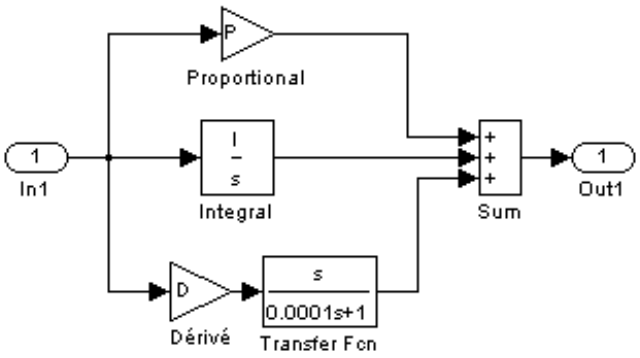


Figure 2.6 *Schéma du contrôleur simulé*

Le gain anticipatif n'est pas représenté car il ne sera pas utilisé dans la procédure de réglage du contrôleur. On y retrouve bien les gains proportionnel, intégral et dérivé.

Nous avons été confrontés à un problème important lors de la simulation: le contrôleur n'agissait pas comme prévu. L'erreur a été localisée dans la dérivation effectuée dans le cas du gain dérivé. Nous avons donc remplacé cette dérivation par une fonction de transfert dont le rôle est identique à celui de la dérivée numérique, mais qui supprime tout problème de discontinuité.

2.2.2. Choix des gains pour une réponse à l'échelon

Le réglage du PID a été réalisé par placement de pôles. Pour cela, nous avons réalisé une simulation de réponse à l'échelon de notre système en boucle fermée, à partir de la connaissance des paramètres du système et de la forme du contrôleur. Nous avons ensuite déterminé la fonction de transfert de ce système, et l'avons traitée de manière à appliquer la méthode de placement des pôles.

2.2.2.1. Modèle de simulation

Voici le schéma Simulink réalisé pour la simulation :

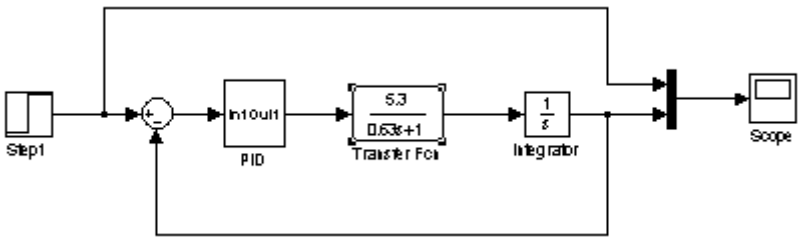


Figure 2.7 *Simulation de réglage du contrôleur*

La procédure correspond à une réponse à l'échelon du système considéré.

Cette simulation est très simple: une commande sous forme d'un échelon est envoyée au système. Elle passe à travers le contrôleur, puis dans le système. Une rétroaction s'opère enfin.

2.2.2.2. Fonction de transfert

La fonction de transfert du système en boucle fermée est calculée à partir des figures [2.6] et [2.7] :

$$\frac{X}{X_{ref}} = \frac{\left(K_P + K_D\, s + \frac{K_I}{s}\right)\frac{K_m}{s(s\tau_m + 1)}}{1 + \left(K_P + K_D\, s + \frac{K_I}{s}\right)\frac{K_m}{s(s\tau_m + 1)}}$$

$$= \frac{K_m K_P + K_m K_D\, s + \frac{K_m K_I}{s}}{\tau_m s^2 + (1 + K_m K_D)s + \frac{K_m K_I}{s}} \qquad [2.7]$$

Nous avons dans un premier temps réglé un contrôleur PD.

La fonction de transfert s'en trouve simplifiée, puisque $K_I = 0$. Elle s'écrit :

$$\frac{X}{X_{ref}} = \frac{K_m K_P + K_m K_D\, s}{\tau_m s^2 + (1 + K_m K_D)s}$$

$$= \frac{\frac{K_m K_D}{\tau_m}\left(s + \frac{K_P}{K_D}\right)}{s^2 + \left(\frac{1 + K_m K_D}{\tau_m}\right)s + \frac{K_m K_P}{\tau_m}} \qquad [2.8]$$

Les simplifications effectuées ont permis d'obtenir une fonction de transfert du second ordre. Néanmoins, cette fonction de transfert contient un zéro.

La méthode de placement de pôles ne prend pas en compte un éventuel zéro. Le choix des pôles à partir des spécifications ne permet pas de tenir compte de la présence d'un zéro.

Avant de présenter la méthode, l'influence du zéro dans une telle fonction de transfert va être étudiée afin de s'assurer que l'application de cette méthode n'introduira pas trop d'erreur.

2.2.2.3. Influence du zéro

L'analyse qui suit est reprise de la référence [4].

La fonction de transfert [2.8] peut être vue comme la somme d'une fonction de transfert du deuxième ordre dite « pure » et d'une fonction de transfert du deuxième ordre avec un terme en « s » au numérateur :

$$\frac{X}{X_{ref}} = \frac{\frac{K_m K_P}{\tau_m}}{s^2 + \left(\frac{1 + K_m K_D}{\tau_m}\right)s + \frac{K_m K_P}{\tau_m}} + \frac{\frac{K_m K_D}{\tau_m}s}{s^2 + \left(\frac{1 + K_m K_D}{\tau_m}\right)s + \frac{K_m K_P}{\tau_m}} \quad [2.9]$$

Si X_{ref} est un échelon unitaire, la réponse vaut alors :

$$X = \frac{\frac{K_m K_P}{\tau_m}}{s\left[s^2 + \left(\frac{1 + K_m K_D}{\tau_m}\right)s + \frac{K_m K_P}{\tau_m}\right]} + \frac{\frac{K_m K_D}{\tau_m}s}{s\left[s^2 + \left(\frac{1 + K_m K_D}{\tau_m}\right)s + \frac{K_m K_P}{\tau_m}\right]}$$

Si le premier terme du second membre est noté $X_a(s)$, l'équation s'écrit :

$$X(s) = X_a(s) + \left(\frac{1}{a}\right)s\,X_a(s)$$

où $a = \frac{K_P}{K_D}$

Dans cette expression, $X_a(s)$ est la transformée d'une réponse à l'échelon d'un second ordre pur, et $\left(\frac{1}{a}\right)s\,X_a(s)$ correspond à $\left(\frac{1}{a}\right)$ fois la transformée de la dérivée d'une réponse à l'échelon d'un second ordre pur.

Donc, en transformant dans le temps, la réponse totale s'écrit :

$$x(t) = x_a(t) + \frac{1}{a}\left(\frac{dx_a}{dt}\right)$$

La réponse totale peut être tracée comme étant la somme de deux réponses :

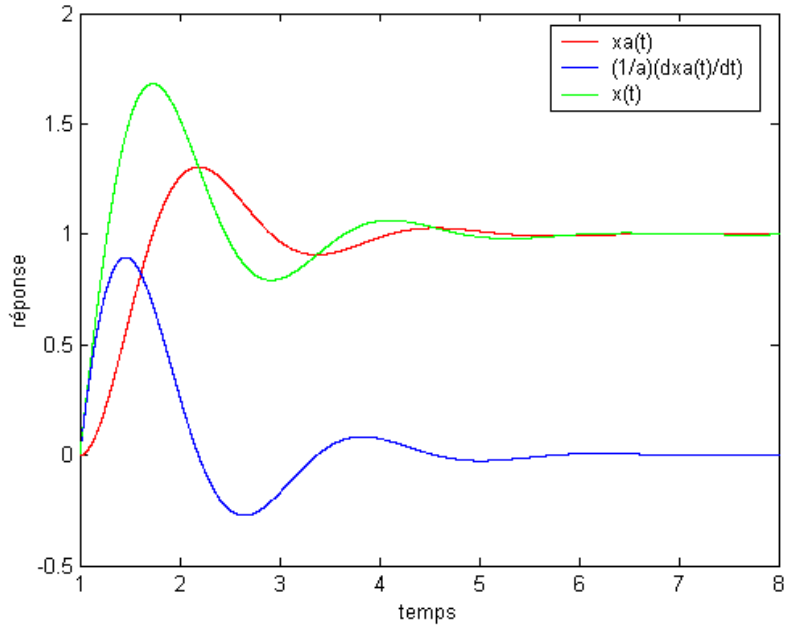


Figure 2.8 *Influence d'un zéro dans une fonction de transfert du second ordre*

L'analyse des courbes montre que le dépassement qu'a la réponse totale $x(t)$ est supérieur à celui qu'a la courbe $x_a(t)$. En outre, les oscillations de $x(t)$ sont plus prononcées que celles de $x_a(t)$.

Plus a sera élevé, c'est-à-dire, plus le zéro sera éloigné du centre du repère, moins le terme dérivé aura d'influence, et donc plus la réponse totale s'approchera de la réponse du second ordre pur. C'est dans ce cas que la méthode, appliquée à un second ordre possédant un zéro sera la plus fiable.

Il faut donc, dans la recherche des gains du PD, essayer de situer le zéro le plus loin dans la partie réelle négative, afin d'utiliser la méthode de placement de pôles pour une fonction de transfert du second ordre. Ce zéro a pour valeur $-\frac{K_P}{K_D}$. Le gain proportionnel doit être beaucoup plus grand que le gain dérivé.

2.2.2.4. Détermination des gains proportionnel et dérivé

Reprenons l'équation [2.8], et considérons uniquement la partie de l'équation qui ne contient pas de zéro. Cette fonction de transfert possède deux pôles, soit p_1 et p_2. Le dénominateur est donc de la forme $(s-p_1)(s-p_2)$, qui vaut, en développant l'équation : $s^2-(p_1+p_2)s+p_1p_2$.

En identifiant cette forme au dénominateur de la fonction de transfert considérée, nous pouvons exprimer les gains K_P et K_D en fonction des pôles p_1 et p_2 :

$$\begin{cases} K_P = \dfrac{\tau_m}{K_m} p_1 p_2 \\ K_D = -\dfrac{1}{\tau_m}\left[\tau_m (p_1 + p_2) + 1\right] \end{cases} \qquad [2.10]$$

D'autre part, une fonction de transfert du deuxième ordre « pure », peut s'écrire sous la forme :

$$\frac{X}{X_{ref}} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

où ω_n et ζ sont la pulsation propre et le facteur d'amortissement du système.

Le pôle double s'écrit :

$$p_{1,2} = -\zeta\omega_n \pm i\omega_n\sqrt{1-\zeta^2} \qquad \text{si} \quad \zeta < 1 \qquad [2.11]$$

La figure suivante illustre ces variables.

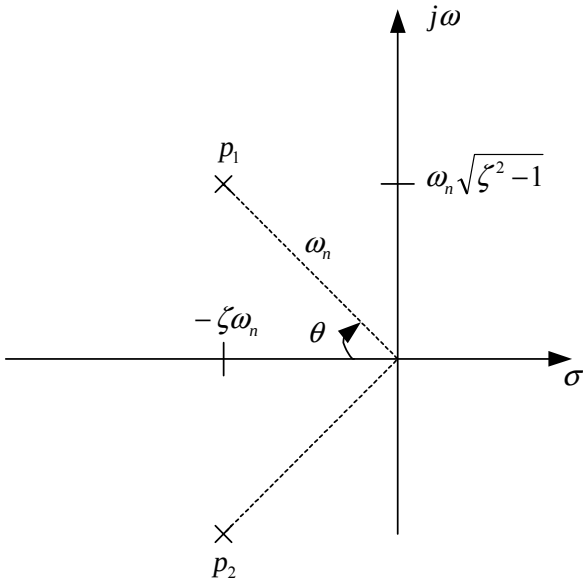


Figure 2.9 *Configuration des pôles doubles d'un second ordre*

En introduisant la valeur des pôles de [2.11] dans l'expression des gains [2.10], une nouvelle expression des gains est obtenue :

$$\begin{cases} K_P = \dfrac{\tau_m}{K_m}\left[(\zeta\omega_n)^2 + \omega_n^2\left(1-\zeta^2\right)\right] \\ K_D = -\dfrac{1}{\tau_m}\left[\tau_m\left(-2\zeta\omega_n\right)+1\right] \end{cases}$$

Afin de vérifier nos résultats, nous avons exprimé le dépassement d et le temps de dépassement T_P, en fonction du facteur d'amortissement ζ et de la pulsation propre ω_n :

$$\begin{cases} \zeta = \sqrt{\dfrac{\left(ln\dfrac{d}{100}\right)^2}{\left(ln\dfrac{d}{100}\right)^2 + \pi^2}} \\ \omega_n = \dfrac{\pi}{T_P\sqrt{1-\zeta^2}} \end{cases} \Rightarrow \begin{cases} d = 100\, e^{\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right)} \\ T_P = \dfrac{\pi}{\omega_n\sqrt{1-\zeta^2}} \end{cases}$$

A partir du choix du facteur d'amortissement et de la pulsation propre désirés, il est possible de déterminer les pôles de la fonction de transfert étudiée, puis de vérifier le dépassement et le temps de dépassement de la réponse du système.

Le tableau suivant reprend les résultats obtenus pour différentes valeurs des paramètres ζ et ω_n :

Paramètres du système	K_m	5,3			
	τ_m	0,63			
Paramètres choisis	ζ	0,707	0,707	0,5	0,5
	ω_n	8	15	8	15
Résultat des gains proportionnel et dérivé	K_p	7,61	26,75	7,61	26,75
	K_d	1,16	19,62	6,41	13,41
Dépassement et temps de dépassement calculés	d	4,33	4,33	16,30	16,30
	T_p	0,56	0,30	0,45	0,24

Tableau 2.3 *Résultats des gains en fonction des paramètres ζ et ω_n*

Pour effectuer le choix des gains, il faut prendre en compte l'influence du zéro. Comme vu ci-dessus, son effet peut être négligé en prenant un gain K_P bien supérieur au gain K_D. Parmi les différentes solutions, celle qui permet de négliger l'influence du zéro sera retenue afin que la simulation s'approche le plus de la réalité.

Les deux dernières configurations donnent un dépassement trop important. Nous avons choisi de présenter ici les courbes obtenues pour la première configuration, qui est la plus courante.

Les gains retenus sont $K_P = 7.6$ et $K_D = 1.16$.

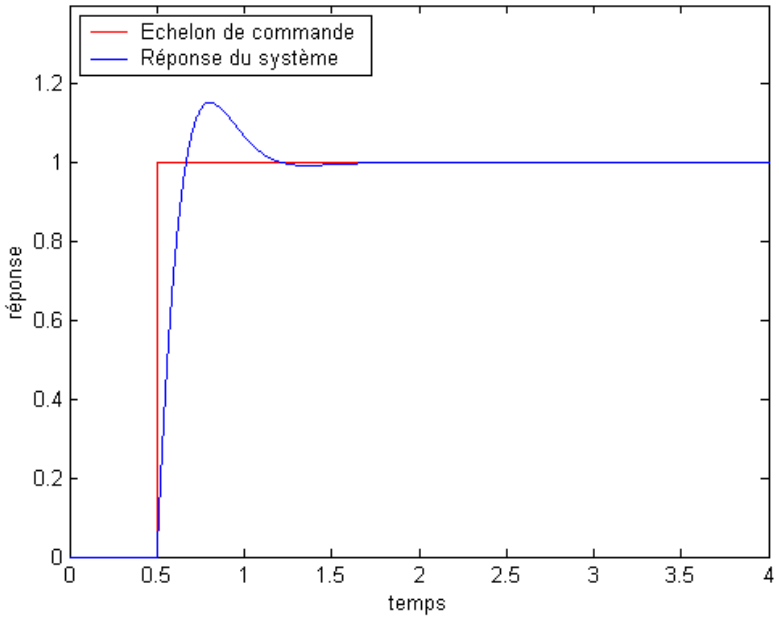


Figure 2.10 *Réponse à l'échelon obtenue pour les gains théoriques*

Cette figure montre bien que le dépassement est supérieur au dépassement calculé dans le tableau [2.3]. Il faut se rappeler que les gains ont été déterminés en considérant un second ordre « pur ». L'influence du zéro, telle que décrite plus haut, est confirmée.

Nous avons essayé d'ajuster ces gains. La méthode utilisée a été d'augmenter, l'un après l'autre, les gains proportionnel et dérivé. Les gains choisis sont $K_P = 35$ et $K_D = 15$, pour lesquels la réponse était satisfaisante, et la commande répondait aux spécifications. La courbe de vitesse linéaire obtenue pour ces gains est présentée ci-contre.

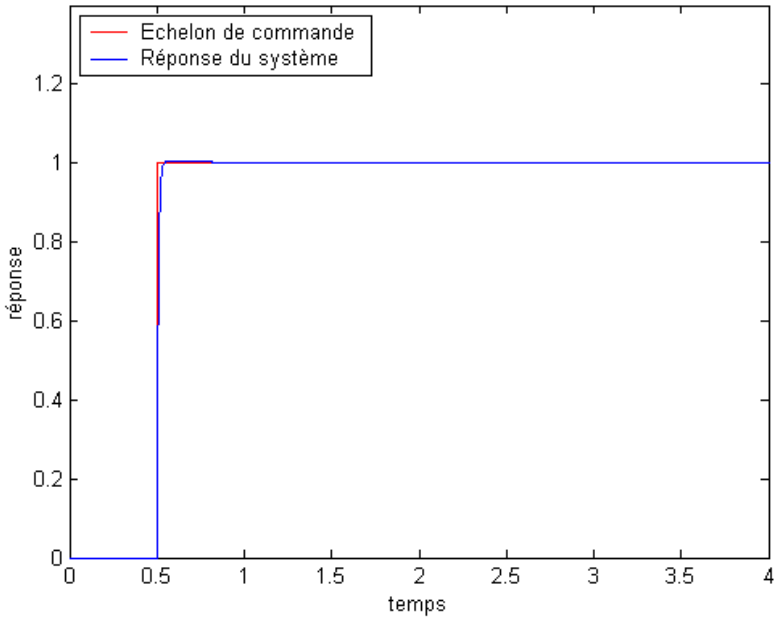


Figure 2.11 *Réponse à l'échelon obtenue pour les gains ajustés*

2.2.3. Validité des gains obtenus

Le chapitre suivant présente des tests en boucle fermée et compare, pour une commande trapézoïdale, la simulation et l'expérimentation. De plus, l'effet des gains intégral et anticipatif, volontairement omis lors du réglage du contrôleur, y est analysé.

CHAPITRE III : TESTS EN BOUCLE FERMÉE

Les tests en boucle fermée qui sont présentés dans ce troisième chapitre ont permis de valider le choix des gains, en simulation et en expérimentation.

La simulation réalisée afin de comparer les tests réels à la théorie est tout d'abord présentée. Dans le deuxième paragraphe, les résultats des différents essais sont analysés, et l'influence des gains anticipatif et intégral, négligés lors du réglage du contrôleur, sont discutés. Enfin, nous présentons les résultats obtenus lors de l'ajustement des gains.

1. SIMULATION DU CONTRÔLEUR EN BOUCLE FERMÉE

1.1. Programme Simulink

La simulation reproduit le mieux possible le système réel.
Voici le programme réalisé sur Simulink :

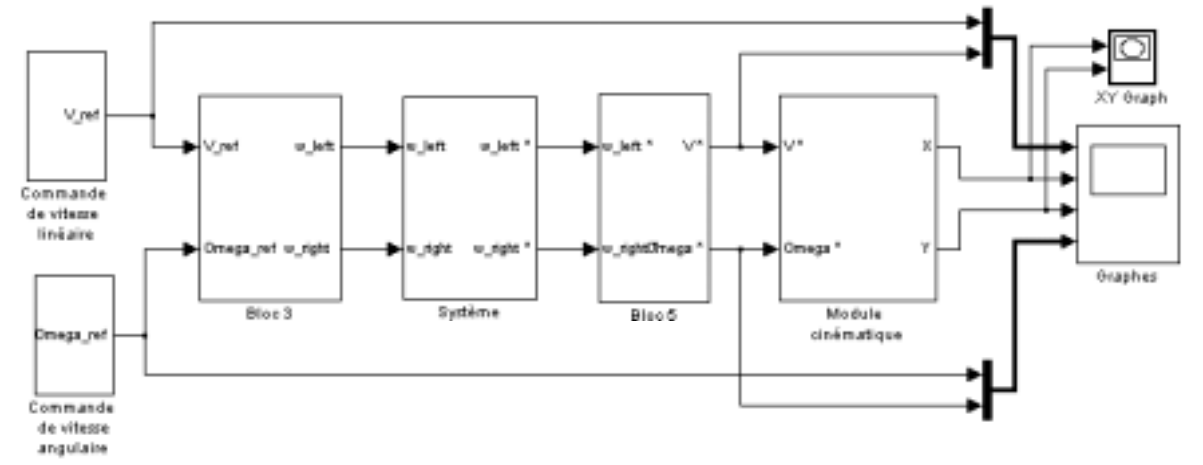


Figure 3.1 *Schéma Simulink du contrôle en boucle fermée*

Détaillons chaque bloc (l'intérieur de chaque bloc est présenté en annexe) :

- Les deux premiers blocs, intitulés « commande de vitesse linéaire » et « commande de vitesse angulaire », génèrent le profil de vitesse souhaité. La carte de contrôle des moteurs offre plusieurs profils, comme nous l'avons vu au chapitre I. Le profil trapézoïdal a été choisi. Ces deux blocs proposent donc d'utiliser soit un profil trapézoïdal, soit une vitesse constante. Ce choix permet de réaliser des tests en translation pure et en translation/rotation.
- Les roues étant indépendantes, et couplées chacune à un moteur, les vitesses linéaire v et angulaire Ω désirées pour le robot doivent être transformées en une vitesse applicable à chaque roue. C'est l'objet du troisième bloc.

Les vitesses sont transformées selon les formules suivantes :

Vitesse de la roue gauche : $$\omega_{left} = \frac{v - \frac{\Omega l}{2}}{R} \quad \text{en rad.s}^{-1}$$

Vitesse de la roue droite : $$\omega_{right} = \frac{v + \frac{\Omega l}{2}}{R} \quad \text{en rad.s}^{-1}$$

Où l est la distance entre les roues motrices, et R le rayon des roues.

- Le quatrième bloc reproduit la dynamique du système. Il comporte, pour chaque roue, donc pour chaque moteur, le contrôleur attaché à chaque moteur et la fonction de transfert correspondant au système. Le contrôleur est celui présenté à la figure [2.6].
- Le cinquième bloc effectue l'opération inverse du troisième bloc : il transforme les vitesses appliquées aux roues en deux vitesses : une vitesse linéaire et une vitesse angulaire. Ces dernières vont nous permettre d'obtenir la position du robot. Les formules utilisées sont les suivantes :

Vitesse linéaire : $$v = \frac{\omega_{right} + \omega_{left}}{2} \quad \text{en m}$$

Vitesse angulaire : $$\Omega = \frac{\omega_{right} - \omega_{left}}{l} \quad \text{en rad}$$

- Le dernier bloc permet de calculer la position du robot à partir des vitesses linéaire et angulaire. Les équations utilisées sont celles de cinématique :

$$\theta = \int \Omega \, dt$$
$$x = \int v \cos\theta \, dt$$
$$y = \int v \sin\theta \, dt$$

1.2. Résultats de la simulation

Nous avons simulé la réponse du système pour les gains obtenus dans le chapitre précédent, soit $K_P = 35$ et $K_D = 15$. Le but étant de vérifier, dans un premier temps, si les gains étaient bien ajustés pour la simulation, cette simulation a été testée pour un profil trapézoïdal de vitesse linéaire et une vitesse angulaire nulle. Les courbes de vitesse et de position obtenues lors de cette simulation sont présentées ci-après.

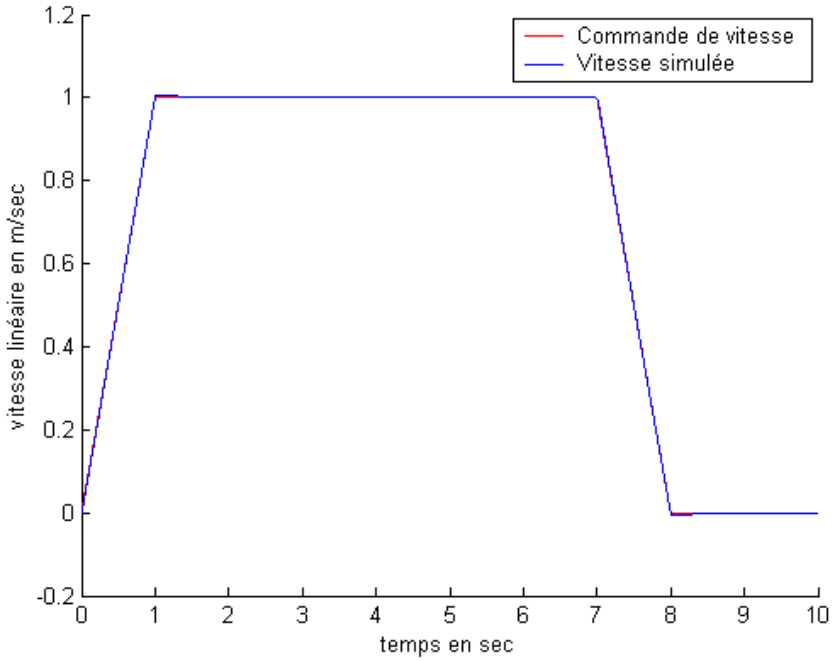


Figure 3.2 *Réponse à une commande de vitesse linéaire trapézoïdale - Tracé de la vitesse linéaire - $K_P = 35$ et $K_D = 15$ (en simulation)*

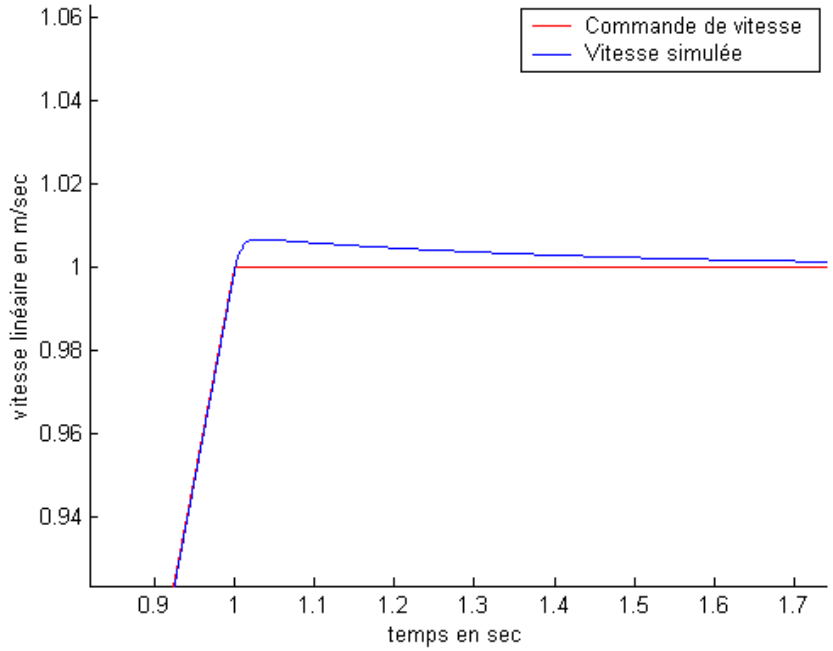


Figure 3.3 *Réponse à une commande de vitesse linéaire trapézoïdale - Zoom du tracé de la vitesse linéaire - $K_P = 35$ et $K_D = 15$ (en simulation)*

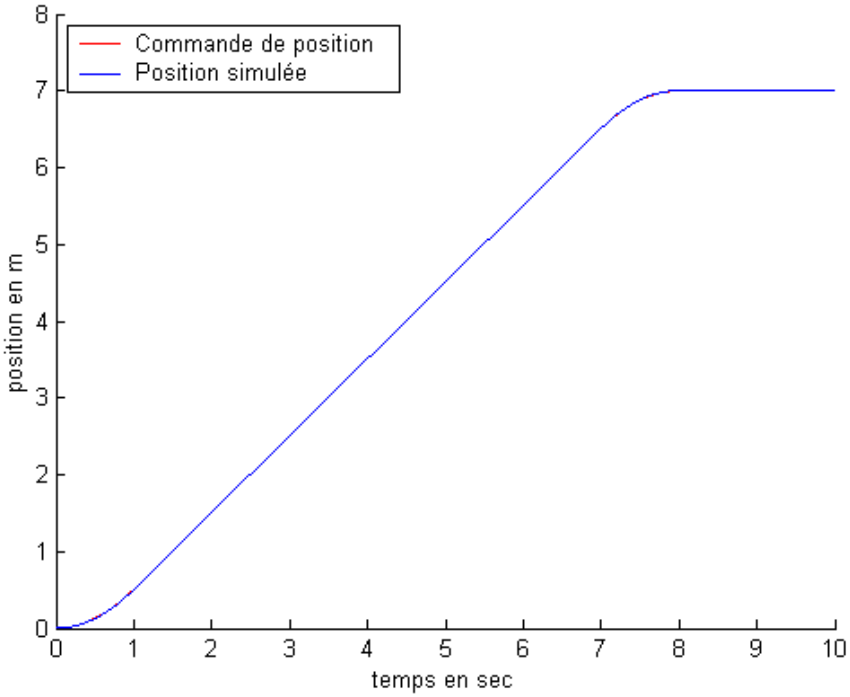


Figure 3.4 *Réponse à une commande de vitesse linéaire trapézoïdale - Tracé de la position -* $K_P = 35$ *et* $K_D = 15$ *(en simulation)*

Sur ce dernier tracé, les courbes de commande de position et de position simulée, se superposent parfaitement. C'est pourquoi seule une des deux courbes se voit.

Ces résultats sont satisfaisants. La vitesse envoyée au robot suit bien la commande que nous avions fixée. Nous pouvons, sans devoir ajuster ces gains, passer à l'expérimentation.

2. EXPÉRIMENTATION DU CONTRÔLEUR EN BOUCLE FERMÉE

2.1. Procédure des essais

2.1.1. Problèmes rencontrés

2.1.1.1. Mesure du temps

La mesure du temps est une chose assez délicate dans des essais de ce genre. L'envoi des commandes, et la lecture des encodeurs se fait en temps réel, donc il faut des mesures précises du temps, car le calcul de la vitesse du robot en dépend. Il faut donc un moyen fiable de le mesurer.

Nous avons tout d'abord voulu utiliser le temps que nous fournit la carte de contrôle, et ce via la commande « *get_time* ». Le manuel de référence de la carte de contrôle nous renseigne sur cette fonction : « *get_time* » retourne la valeur actuelle du temps du système. Il est précisé que cette valeur est exprimée en nombre de boucles effectuées depuis la mise en marche de la carte, et est

envoyée sous la forme d'un nombre de 32 bits. Ayant eu beaucoup problèmes de conversion, nous avons préféré éviter ces soucis de conversion.

Le temps donné par QNX a été préféré. Cette mesure est assez précise, puisque il est possible de l'obtenir en nanosecondes. Nous avons validé les mesures du temps par quelques tests pour vérifier que le temps mesuré était conforme au temps réel.

2.1.1.2. Temps de boucle

Le programme tel qu'implanté sur le robot et détaillé au paragraphe 2.1.2, comprend deux parties : une partie séquentielle, concernant l'initialisation des données et la configuration de la carte, et une partie temps réel. Cette dernière est construite sous forme d'une boucle. Seules sont incluses dans cette boucle les commandes propres au fonctionnement du système en temps réel (mesure du temps et des encodeurs), afin d'effectuer un relevé de mesures très fréquent. Malgré les tentatives de diminution du temps de boucle, ce dernier est resté fort élevé.

Nous avons alors effectué des tests simples pour mesurer le temps pris par des commandes de lecture et/ou d'écriture. Ces tests ont été réalisés pour différentes périodes d'échantillonnage de la carte. D'une part, cette variation de la période n'a pas influencé les résultats, et de l'autre, les temps d'instructions, de l'ordre de la milliseconde, sont apparus fort élevés. La structure informatique a tout d'abord été mise en cause. Nous avons supposé qu'elle créait des ralentissements dans le traitement des données. Cependant, des tests effectués avec une autre carte de contrôle, la carte Ajeco, et sur le même système, ont donné de biens meilleurs temps d'instruction.

Nous en sommes venus à la conclusion suivante : la carte ne permet pas des temps plus rapides. La carte Ajeco n'offrant pas de fonctionnement en boucle ouverte, et étant destinée à gérer uniquement deux axes, nous avons conservé la carte Acs-Tech80.

Il faut donc garder à l'esprit ce problème, qui va beaucoup perturber les tests par la suite. En effet, la longueur de ces temps d'instructions implique que la lecture des encodeurs n'est pas simultanée, et les mesures recueillies sont donc erronées.

2.1.1.3. Interruption du CPU

Un autre problème a été constaté en analysant le temps de boucle. Périodiquement, ce temps est plus élevé de quelques millisecondes. Les temps de boucle ne sont pas constants, et il est nécessaire, pour tracer les courbes de vitesse, d'avoir des périodes d'échantillonnage constantes. Cette inconstante est due au fait que la tâche principale est partiellement interrompue par le système d'exploitation. Ne pouvant éviter ces désagréments, nous avons « forcé » la boucle à s'exécuter en un temps imposé. Le temps choisi est supérieur au temps moyen afin de ne pas être dérangé par les interruptions du CPU. Nous avons donc placé, à la fin de la boucle, une boucle d'attente permettant d'effectuer la boucle temps réel en un temps constant et exactement connu. Cela a permis d'obtenir des courbes moins bruitées, donc plus faciles à interpréter.

2.1.1.4. Conversion des paramètres

La conversion des unités réelles en unités de la carte s'est avérée assez complexe et a généré beaucoup de problèmes.

Pour effectuer ces conversions, il faut tenir compte :

- De la résolution des encodeurs

- Du format exigé par la carte. Ce dernier est tantôt sur 16 bits, tantôt sur 32, tantôt en signé, tantôt en non signé.

Les données à convertir sont celles utilisées par la carte : les longueurs, les vitesses, les accélérations et les gains.

- La conversion est immédiate pour les longueurs : la résolution des encodeurs étant de 11800 tics/tour, et le rayon des roues de 0.0525 m, on calcule que le nombre de tics/m vaut $K_L = \frac{11800}{2\pi\ 0.025}$, soit 35772.

- La vitesse, quant à elle, est exprimée en 32 bits non signés en counts/sample, avec un facteur d'échelle de 2^{-16}.

Il faut tout d'abord convertir les rad.sec^{-1} en counts/sample. Le facteur de conversion est le suivant : $\frac{11800}{2\pi}\left(\frac{tics}{rad}\right).T_{éch}\left(\frac{sec}{échantillon}\right)$. Pour une période d'échantillonnage $T_{éch}$ fixée à 400.10-6 sec, le facteur est égal à $\frac{2.36}{\pi}\ \ \frac{tics.sec}{rad.éch}$.

Pour obtenir le facteur de conversion final en unité de la carte, il faut multiplier par le facteur d'échelle. Cela donne finallement : $K_V = \frac{2.36*2^{16}}{\pi}\ \ \frac{sec}{rad}*unités\ de\ la\ carte$.

- La conversion des accélérations s'effectue selon le même schéma.

En considérant l'accélération en m.sec^{-2}, le facteur de conversion de l'accélération en counts/sample2 vaut : $\frac{11800}{2\pi\pi}\left(\frac{tics}{m}\right).T_{éch}^2\left\{\frac{sec^2}{échantillon^2}\right\}$, ce qui donne, en prenant une période d'échantillonnage $T_{éch}$ de 400.10-6 sec : $\frac{9.44e-4}{2\pi\pi*0.02}\ \ \frac{tics.sec^2}{m.éch^2}$.

Le facteur d'échelle étant identique pour l'accélération, le facteur de conversion final vaut : $K_A = \frac{9.44e-4*2^{16}}{2\pi\pi*0.02}\ \ \frac{sec^2}{m}*unités\ de\ la\ carte$.

- Pour les gains, il faut se ramener au contrôleur pour en connaître les unités précises. En considérant uniquement un gain proportionnel, nous avons : $u = Ke$, où u est la commande (en unité de vitesse), K le gain et e l'erreur (en unité de longueur). Le gain K est donc égal au quotient de la commande par l'erreur.

Les facteurs de conversion de l'erreur et de la commande ont été calculés ci-dessus. La commande variant entre -215+1 et 215-1, nous pouvons calculer le facteur de conversion à appliquer aux gains. Il vaut : $K_G = \frac{2^{15}-1}{35772}$, soit $\frac{32767}{35772}$.

2.1.1.5. Problèmes techniques

Du à un besoin urgent pour une compétition de robotique, le prototype a été construit assez rapidement et sa structure comporte quelques problèmes mécaniques.

D'une part, le matériel ne permet pas la meilleur précision : les roues sont fort larges ce qui implique que le point de contact avec le sol est assez imprécis. Elles n'adhèrent pas beaucoup au sol, les glissements sont de ce fait nombreux. De plus, des cliquetis provenant des moteurs nous incitent à penser que les engrenages sont quelque peu usés.

D'autre part, les fixations roues/axe/moteur ne sont pas très solides. Il existe un jeu important que des resserrages fréquents ne parviennent pas à éliminer. La roue droite a subi un fort dommage, mais la réparation est un peu « artisanale », et la roue subit des secousses en tournant autour de son axe. Nous avons tracé les courbes de vitesse de chaque roue, et nous avons en effet constaté que la roue droite introduisait davantage de bruit que la gauche.

Le comportement du robot s'en est vu fort troublé, et les courbes tracées sont entachées de bruit relié à ces problèmes.

Comme nous n'avions pas de matériel de rechange, nous avons tout de même continué les essais, en étant conscients que les perturbations ne seraient pas présentes sur le prochain robot.

2.1.1.6. Problèmes divers

D'autres problèmes divers ont été rencontrés lors des tests :

- La prise en main du matériel, tant au niveau du robot lui-même qu'à celui des liens RF, a été parsemée d'embûches. Une certaine adaptation a été nécessaire, car de nouveaux problèmes s'ajoutaient à chaque fois.
- La prise en main du système d'exploitation QNX, inconnu de l'utilisateur et très peu documenté, a été fastidieuse.
- Des bugs informatiques, inévitables lors de la programmation, ont du être résolus avec méthode et patience.

2.1.2. Programmation

Les tests réels en boucle fermée ont été programmés en C++, conformément à la structure existante.

Le schéma bloc du contrôleur en boucle fermée indique la procédure suivie pour la programmation. Le bloc « robot » est réalisé dans une boucle temps réel. Le bloc « cinématique directe » est programmé en Matlab :

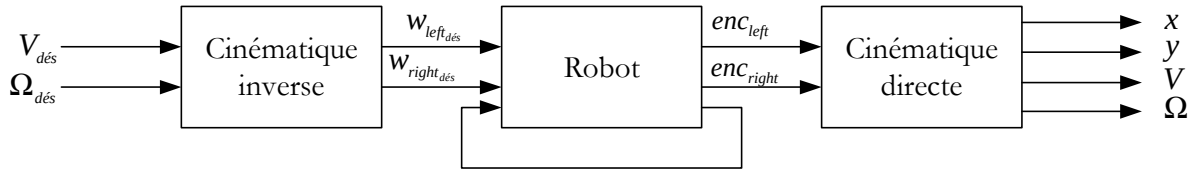


Figure 3.5 *Schéma bloc du contrôleur réel en boucle fermée*

Le code écrit pour ces tests se trouve en annexe, le pseudo code de ce programme est présenté ci-dessous.

```
Déclaration des données
Initialisation des données
Conversion des données
Initialisation de la carte en boucle fermée
Envoi des commandes au registre
Application simultanée des commandes
Boucle
     Lecture du temps
     Lecture des encodeurs
     Boucle d'attente
     Incrémentation des indices
Tant que la distance désirée n'est pas atteinte
Remise à zéro des vitesses
Stockage des résultats dans un fichier
```

Remarque : la vitesse envoyée au moteur gauche est l'opposée de la vitesse calculée car les deux moteurs sont montés en face à face, et il faut donc prendre l'opposé de cette vitesse pour que les deux roues du robot tournent dans le même sens dans le cas d'une translation pure.

2.2. Résultats

2.2.1. Recueil et tracé des données

Afin d'analyser les résultats et de les comparer à la simulation, nous devons tracer les courbes de vitesse et de position à partir des données stockées. La matrice des données, stockée sur le micro drive du robot, est formée des *n* mesures du temps et de chaque encodeur (*n* étant le nombre de boucles effectuées). Elle est transférée grâce aux liens RF, vers un PC de bureau disposant de Matlab.

Les mesures des encodeurs, données en tics, sont converties avant d'être utilisées pour tracer les courbes de vitesse et de position :

$$\underbrace{encodeur[i]}_{en\ mètres} = \underbrace{encodeur[i]}_{en\ tics} * \frac{2\pi\pi}{\underbrace{11800}_{en\ m/tics}}$$

où *i* est l'indice d'itération.

La vitesse de chaque roue est donnée par :

$$v_{left}[i] = \frac{enc\_left[i] - enc\_left[i-1]}{\Delta T[i]}$$

$$v_{right}[i] = \frac{enc\_right[i] - enc\_right[i-1]}{\Delta T[i]}$$

Les données que nous désirons tracer sont calculées comme suit:

- La position : $position[i] = \frac{enc\_left[i] + enc\_right[i]}{2}$
- La vitesse linéaire : $v[i] = \frac{v_{left}[i] + v_{right}[i]}{2}$
- La vitesse angulaire : $\Omega[i] = \frac{v_{right}[i] - v_{left}[i]}{l}$

Chaque temps de boucle est sommé pour obtenir le temps total en fonction duquel on tracera les courbes : $T[i] = T[i-1] + \Delta T[i]$.

Les courbes sont superposées aux courbes extraites de la simulation.

2.2.2. Courbes

2.2.2.1. Tests en translation pure

Nous avons, dans un premier temps, effectué des tests pour les mêmes gains que ceux utilisés lors de la simulation. Pour ces deux premiers essais, les paramètres suivants ont été définis :

- $K_P = 35$
- $K_D = 15$
- $K_I = K_{v_{ff}} = 0$
- vitesse linéaire : $v = 0.5\ m.s^{-1}$
- vitesse angulaire : $\Omega = 0\ rad.s^{-1}$
- accélération : $a = 1\ m.s^{-2}$

La vitesse de chaque roue est calculée à partir des vitesses linéaire et angulaire. Ces données, ainsi que l'accélération, sont utilisées par la carte de contrôle afin de générer un profil de vitesse trapézoïdal.

La mesure de la vitesse est bruitée, mais la réponse du système réel est assez conforme à la simulation. Les résultats se trouvent à la page suivante :

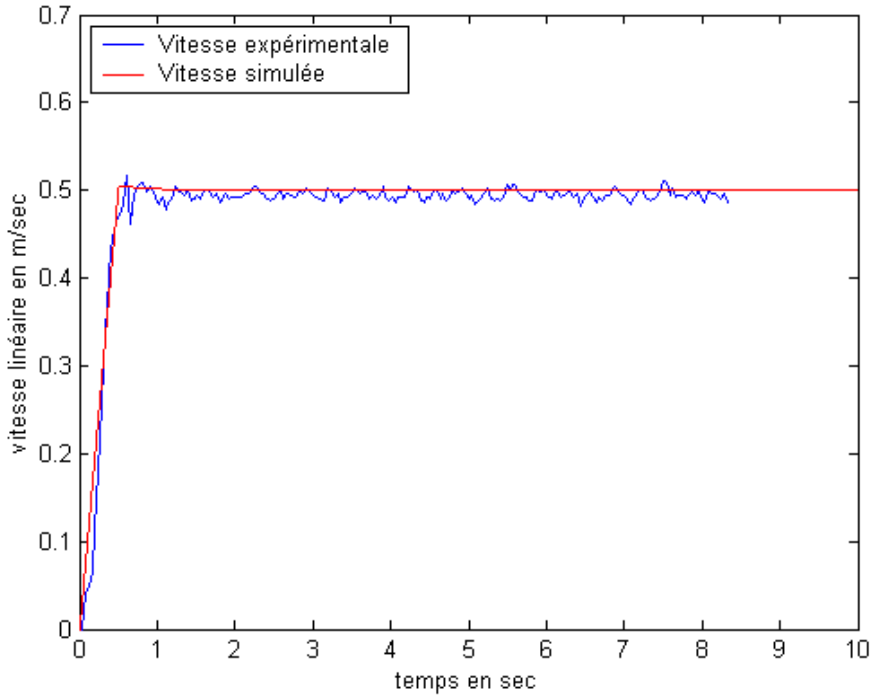


Figure 3.6 *Réponse à une commande de vitesse trapézoïdale - Tracé de la vitesse linéaire - en expérimentation, pour* $K_P = 35$, $K_D = 15$, $K_I = K_{v_{ff}} = 0$, $v = 0.5\ m.s^{-1}$ *et* $a = 1\ m.s^{-2}$

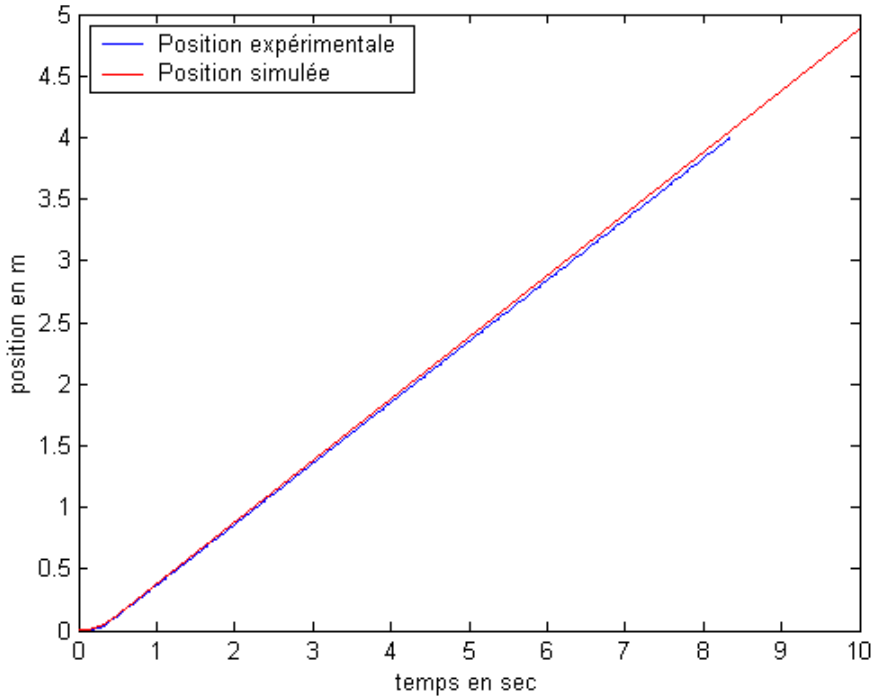


Figure 3.7 *Réponse à une commande de vitesse trapézoïdale - Tracé de la position - en expérimentation, pour* $K_P = 35$, $K_D = 15$, $K_I = K_{v_{ff}} = 0$, $v = 0.5 m.s^{-1}$ *et* $a = 1\ m.s^{-2}$

Afin de valider davantage le comportement du système, nous avons choisi volontairement des gains qui créent un dépassement, et nous avons analysé la réponse du système. Le robot doit dépasser sa vitesse de régime pour y retourner.

Voici les résultats obtenus pour des gains proportionnel $K_P = 15$ et dérivé $K_D = 5$ (les autres paramètres sont conformes à l'essai précédent) :

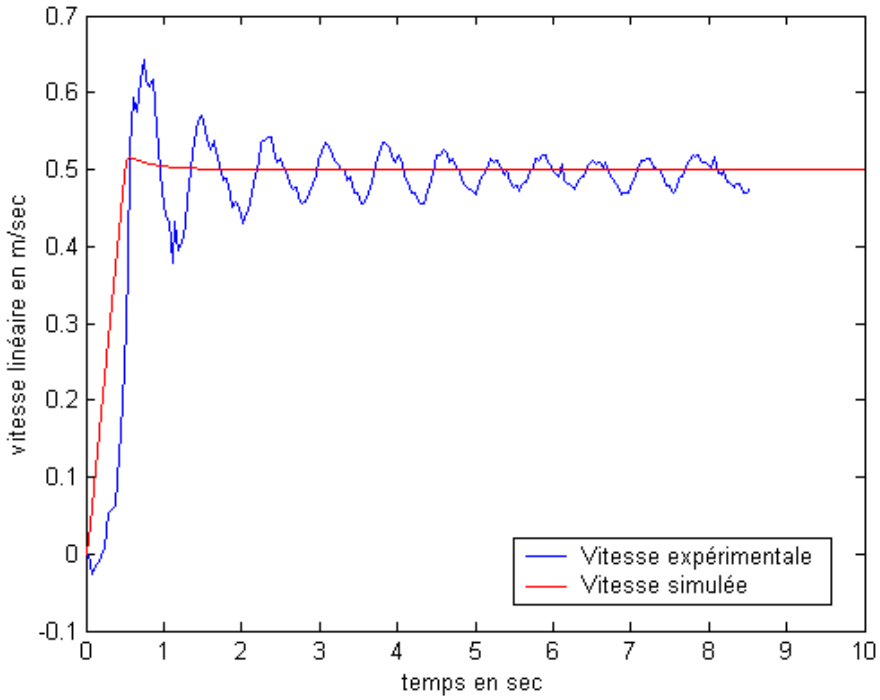


Figure 3.8 *Réponse à une commande de vitesse trapézoïdale, avec dépassement - Tracé de la vitesse linéaire - en expérimentation, pour* $K_P = 15$ *et* $K_D = 5$

Le robot effectue bien le dépassement demandé, mais il oscille ensuite autour de sa vitesse de régime. Le bruit est probablement dû à ces résultats. Mais si on prend une valeur moyenne, on voit que la vitesse tend vers la vitesse de régime.

2.2.2.2. Tests en translation/rotation mixte

Le robot étant amené, dans ses différentes tâches, à effectuer des déplacements courbes, nous avons effectué des tests pour une vitesse angulaire non nulle.

Les gains choisis pour le contrôleur sont les suivants :

- $K_P = 35$
- $K_D = 15$
- $K_I = 0$
- $K_{v_{ff}} = 0$

Les vitesses, quant à elles, sont fixées à :

- $v = 0.5\ m.s^{-1}$

- $\Omega = 0.5\ rad.s^{-1}$
- $a = 1\ m.s^{-2}$

La vitesse de chaque roue est calculée à partir des deux premières vitesses. Ces données, ainsi que l'accélération, sont utilisées par la carte de contrôle afin de générer un profil de vitesse trapézoïdal.

Les courbes de vitesse linéaire, angulaire et de position, obtenues en expérimentation sont présentées ci-dessous. Elles sont superposées aux courbes obtenues en simulation.

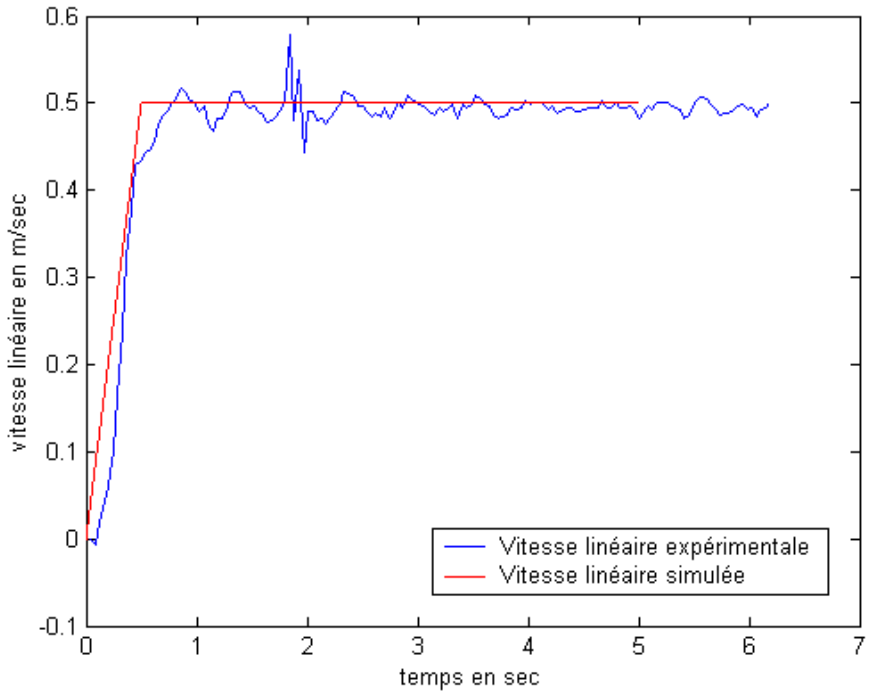


Figure 3.9 *Test mixte (translation/rotation) en boucle fermée, en réponse à un profil de vitesse trapézoïdal - Tracé de la vitesse linéaire*

Cette courbe présente un petit pic en t=2 sec. Ce genre d'incident est inévitable lors des expérimentations : n'importe quelle irrégularité sur le sol perturbe immédiatement le robot.

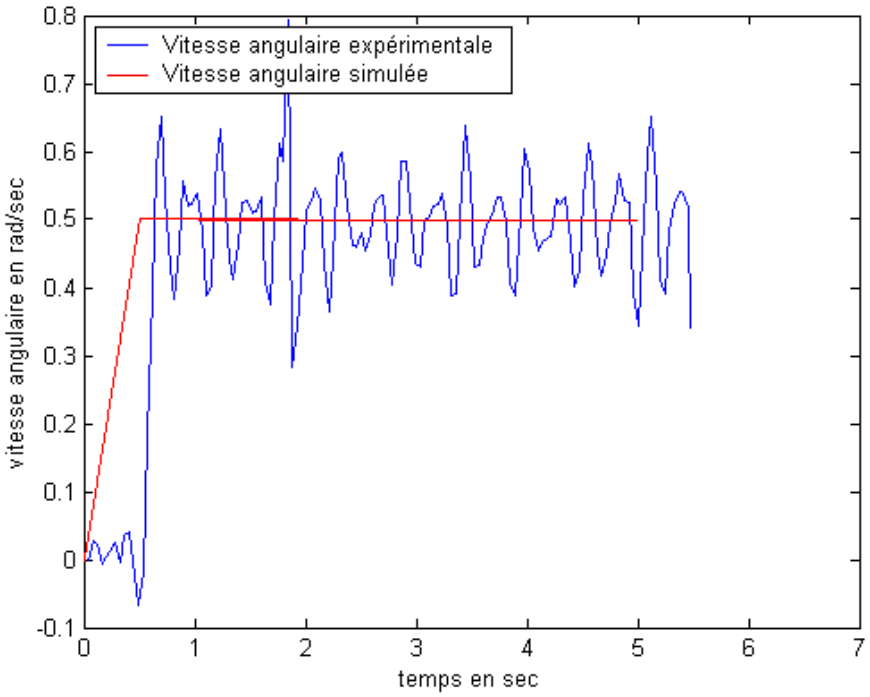


Figure 3.10 *Test mixte (translation/rotation) en boucle fermée, en réponse à un profil de vitesse trapézoïdal – Tracé de la vitesse angulaire*

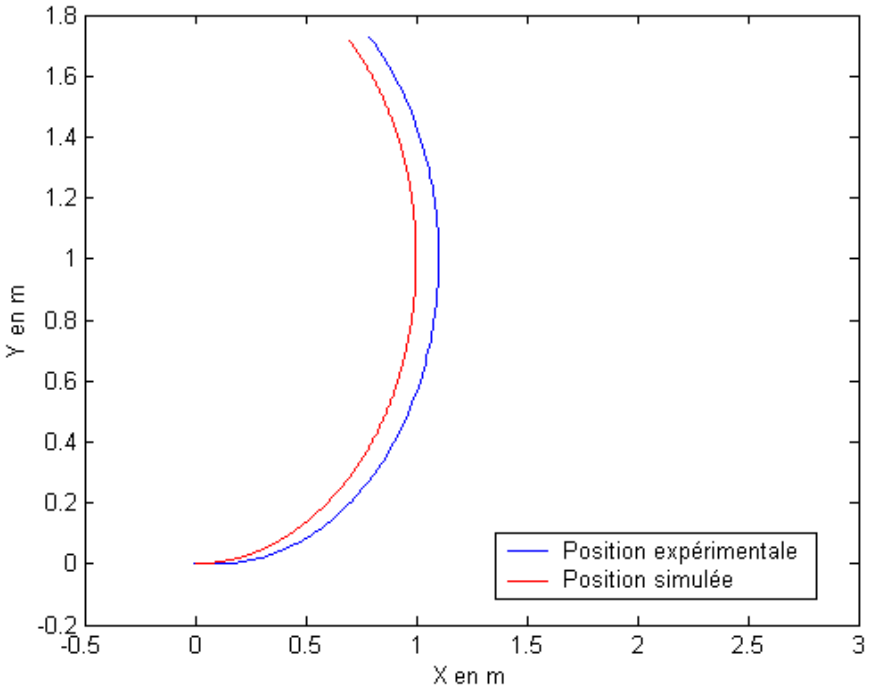


Figure 3.11 *Test mixte (translation/rotation) en boucle fermée, en réponse à un profil de vitesse trapézoïdal – Tracé de la position*

La courbe de la figure [3.9] montre un léger retard de la vitesse. Les oscillations de celle-ci sont plus fortes qu'en translation pure. La vitesse angulaire, quant à elle, est très bruitée. Elle oscille de près de 0.1 $rad.sec^{-1}$ autour de sa valeur de régime. En voyant les courbes de vitesse linéaire, et, surtout, de vitesse angulaire, nous ne sommes pas surpris du tracé de la position. La courbe de position réelle s'écarte de la simulation de plus de 10 cm. Cette erreur est normale puisque seule la vitesse est contrôlée en boucle ouverte. On remarque une erreur en régime permanent, mais les transitoires observés sont mauvais.

Le décalage entre les deux courbes provient du glissement des roues sur le sol. D'une part, les roues sont assez lisses, et de l'autre, le sol est lui-même glissant.

Le comportement du robot est jugé assez mauvais en translation/rotation. Mais ici encore, la mécanique est mise en cause.

2.3. Influence des gains anticipatif et intégral

Les conclusions suite aux essais effectués ci-dessus sont assez mitigées. Les résultats sont très bons en translation, mais nettement moins en translation/rotation. Dans ce paragraphe, nous analysons le comportement du robot en lui appliquant les gains anticipatif et intégral. Ces tests sont effectués pour les essais en translation pure, les autres étant considérés comme trop mauvais.

2.3.1. Influence du gain anticipatif

Le manuel d'utilisation de la carte indique que le gain anticipatif doit être mis à une valeur égale à $1/K_m$. Cette valeur permet d'obtenir la vitesse désirée en régime permanent

Les courbes suivantes montrent les résultats obtenus en effectuant des tests en translation pure. Un test effectué sans gain anticipatif y est superposé. Les courbes montrent que ce gain n'a pas un grand effet sur le comportement du robot.

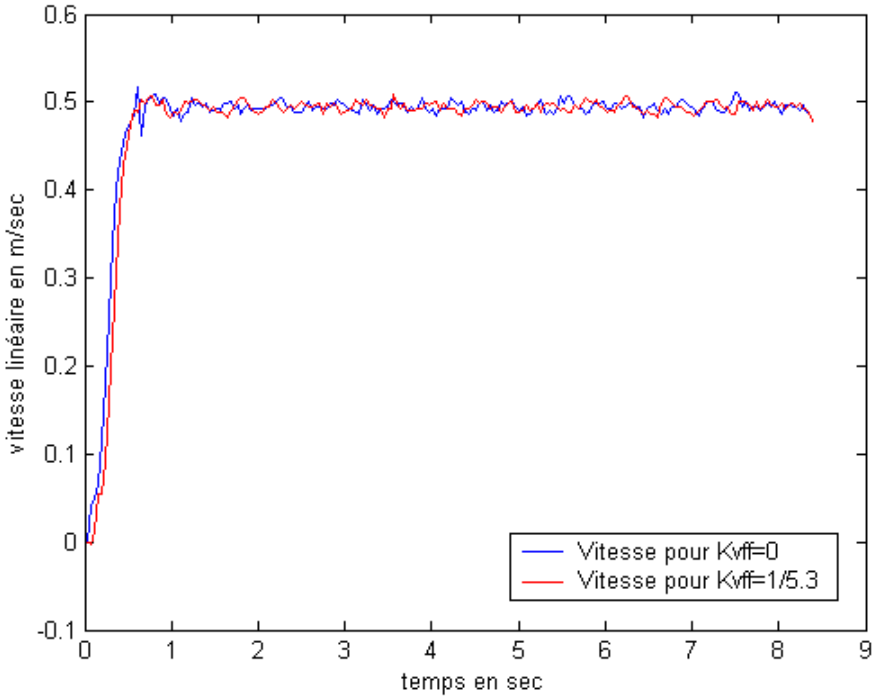


Figure 3.12 *Comparaison du comportement du robot avec et sans gain anticipatif - Tracé de la vitesse*

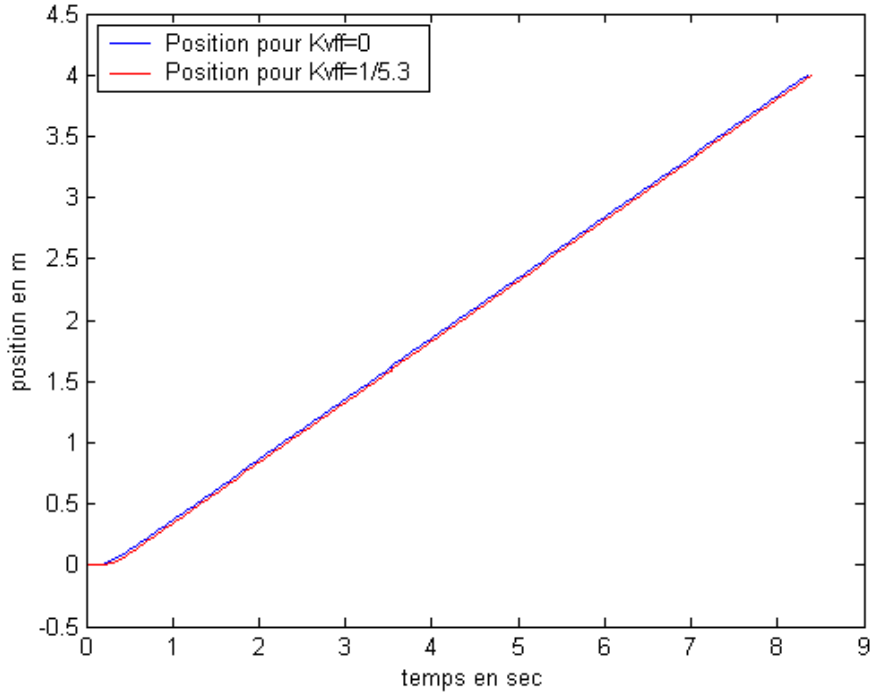


Figure 3.13 *Comparaison du comportement du robot avec et sans gain anticipatif - Tracé de la position*

2.3.2. Influence du gain intégral

Le gain intégral n'aura d'influence qu'en cas de perturbation. Les courbes suivantes montrent le comportement du robot lors d'une perturbation, avec et sans gain intégral. Nous avons essayé de reproduire au mieux la perturbation dans les deux cas.

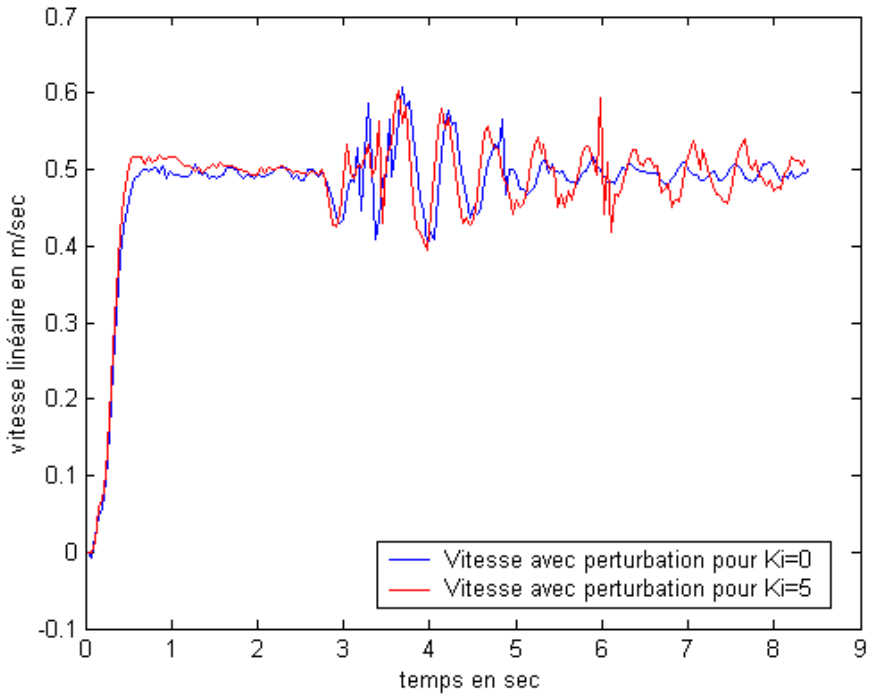


Figure 3.14 *Comparaison du comportement du robot avec et sans gain intégral, lors d'une perturbation - Tracé de la vitesse*

Ici encore, nous voyons que le gain intégral n'influence aucunement le comportement du robot. Néanmoins, la perturbation s'effectuant de manière manuelle (frein constant), ces tests ne sont pas très significatifs.

2.4. Ajustement des gains

Nous avons ensuite réalisé quelques essais afin de voir si de meilleurs résultats pouvaient être obtenus en changeant les gains. Toutefois, afin de respecter les limites de commande, nous ne pouvions les augmenter trop. Les courbes ci-dessous montrent les résultats obtenus pour $K_P = 45$ et $K_D = 30$. Ces courbes sont comparées à celles correspondant aux gains obtenus lors de la détermination théorique.

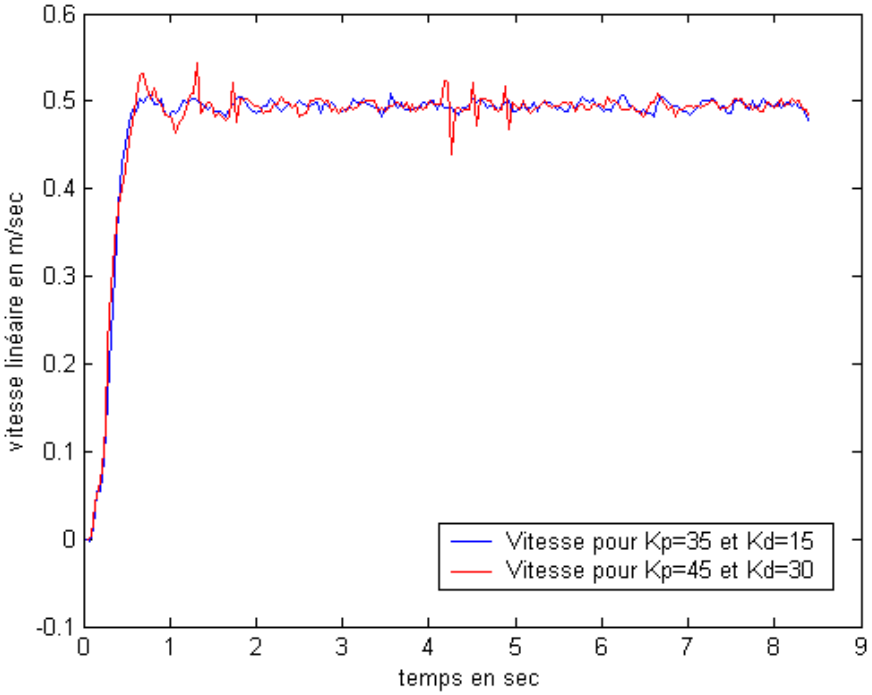


Figure 3.15 *Comparaison du comportement du robot pour des gains proportionnel et dérivé différents - Tracé de la vitesse linéaire linéaire*

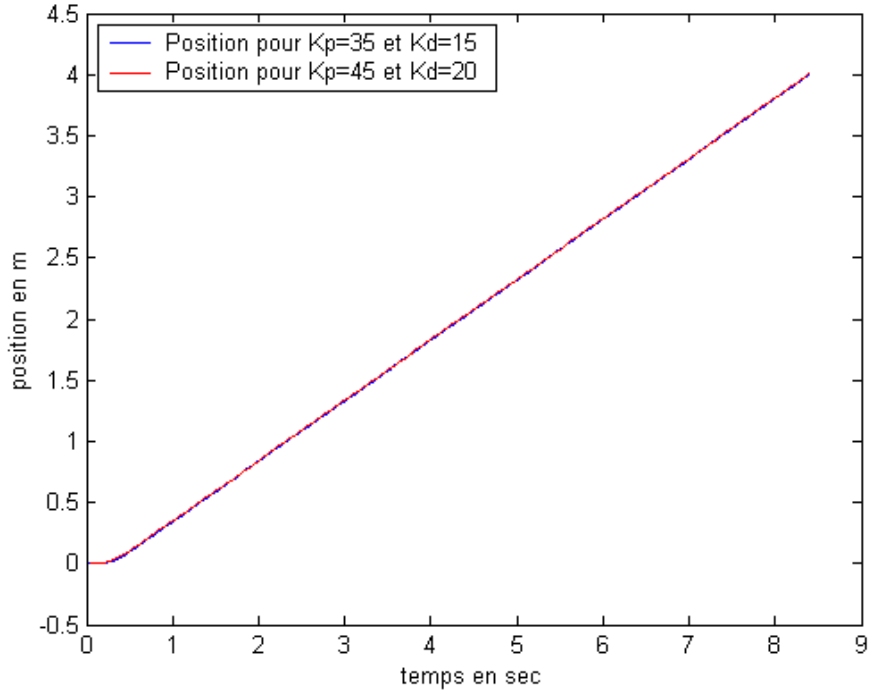


Figure 3.16 *Comparaison du comportement du robot pour des gains proportionnel et dérivé différents - Tracé de la position*

Les courbes présentées sont assez semblables pour les deux choix de paire de gains.

Les gains n'ont pas été davantage augmentés car cela se ferait au dépend de l'amplitude de la commande. Une commande pas trop élevée est maintenue, et les changements de gains sont très peu visibles sur le tracé des courbes. Le bruit est un élément perturbateur qui ne permet pas de juger sur la validité des gains.

Par la suite, les premiers gains obtenus lors de l'ajustement du réglage du contrôleur ont été conservés.

CHAPITRE IV : SUIVI DE CHEMIN

Les tests en boucle fermée ont confirmé le choix des gains. Afin d'approfondir l'étude du comportement du robot mobile, nous avons effectué des tests de suivi de chemin.

Dans ce chapitre, la stratégie utilisée pour assurer que le robot suive un certain chemin est présentée. Cette stratégie a été simulée et implantée sur le robot, et ce pour deux types de chemins : une ligne droite et un cercle. Les résultats expérimentaux sont présentés et analysés. Ils ont été comparés à la simulation.

1. STRATÉGIE DE COMMANDE DU ROBOT

1.1. Objectifs

L'objectif du contrôleur qu'il est ici question de réaliser est d'assurer que le robot mobile suive un certain chemin. Dans un premier temps, nous avons défini ce chemin comme une ligne droite, puis comme en cercle.

Ce contrôleur est basé sur la notion d'erreur de suivi. Cette erreur est constituée de trois composantes :

- L'erreur latérale l_{os} : c'est la distance entre le point P, centre de masse du robot, et le point C, qui correspond au point le plus proche de P sur le chemin à suivre.
- L'erreur d'orientation ε_θ : c'est l'angle entre l'orientation du robot, et la tangente au point C.
- L'erreur longitudinale λ_{os} : c'est la distance à parcourir sur le chemin, entre la projection du point P sur celui-ci et le point d'arrivée. Elle n'est pas considérée dans cette étude. Le robot est arrêté dès qu'il a atteint une certaine valeur.

La figure suivante illustre ces erreurs.

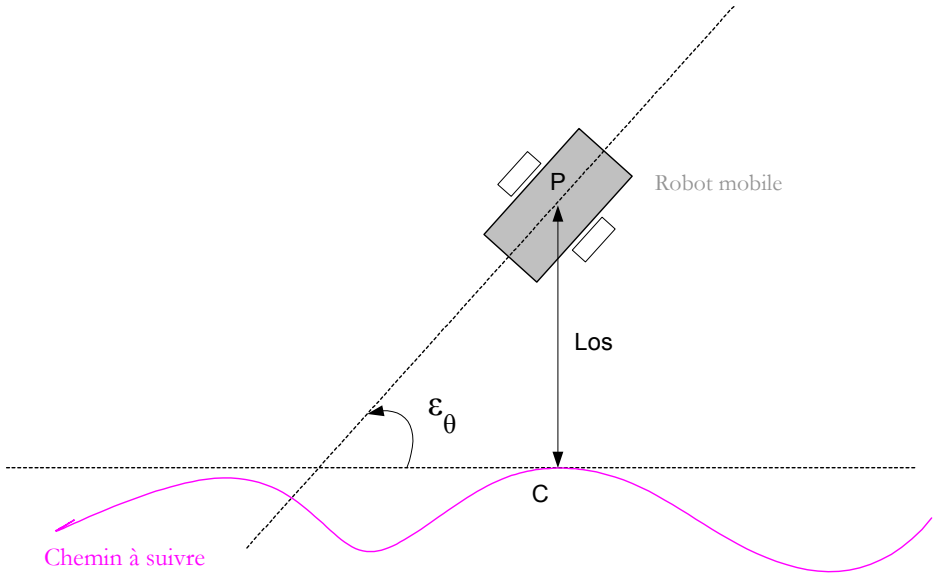


Figure 4.1 *Représentation des erreurs dans le suivi de chemin*

La stratégie de commande du véhicule est donc basée sur l'adaptation des vitesses afin de corriger l'erreur de suivi.

1.2. Calcul des vitesses pour corriger l'erreur de suivi

1.2.1. Calcul des vitesses

Les vitesses linéaire et angulaire servent à contrôler respectivement les erreurs de suivi longitudinal et latéral. Comme nous n'imposons pas de suivi longitudinal, la vitesse linéaire ne nécessite pas de contrôle.

Par contre, il faut contrôler la vitesse angulaire afin de corriger l'erreur latérale. La loi de commande est la suivante :

$$\Omega^* = \Omega_{ref} - \frac{k_1}{V_{ref}} l_{os} - \frac{k_2}{V_{ref}} \dot{l}_{os} \qquad [4.1]$$

Pour démontrer cette loi de commande, on part de l'expression de la dérivée de l'erreur latérale l_{os} :

$$\dot{l}_{os} = V_{ref} \sin \varepsilon_\theta$$

En effectuant l'approximation des petits angles, cette équation peut se réécrire :

$$\dot{l}_{os} = V_{ref} \, \varepsilon_\theta$$

La dérivée de cette expression s'écrit :

$$\ddot{l}_{os} = V_{ref}\left(\Omega - \Omega_{ref}\right)$$

On introduit un contrôle auxiliaire u^* égal à $\ddot{l}_{os}$.

Il suffit alors de poser $u^* = -k_1 l_{os} - k_2 \dot{l}_{os}$, pour envoyer l'erreur à zéro à l'aide d'un PID.

En identifiant les deux expressions obtenues pour le contrôle auxiliaire u^*, nous pouvons arriver à la loi de commande exposée plus haut :

$$V_{ref}\left(\Omega^* - \Omega_{ref}\right) = -k_1 l_{os} - k_2 \dot{l}_{os}$$

$$\Rightarrow \quad \Omega^* = \Omega_{ref} - \frac{k_1}{V_{ref}} l_{os} - \frac{k_2}{V_{ref}} \dot{l}_{os}$$

Le choix des gains est présenté au paragraphe suivant.

1.2.2. Choix des gains de la dynamique de l'erreur latérale

Les gains k_1 et k_2 doivent être choisis de manière à ajuster la dynamique de l'erreur latérale décrite par l'équation suivante :

$$\ddot{l}_{os} + k_2 \dot{l}_{os} + k_1 l_{os} = 0$$

Ce choix est fait par la méthode de placement des pôles. En transformant l'équation précédente par Laplace, et en considérant l'erreur initiale $l_{os}(0)$, l'erreur latérale, en variable de Laplace, s'écrit :

$$L_{os} = \frac{l_{os}(0)}{s^2 + k_2 s + k_1}$$

Cette fonction de transfert est un second ordre. Elle a donc deux pôles. Son dénominateur s'écrit : $(s - p_1)(s - p_2)$ où p_1 et p_2 sont les pôles du système correspondants à la fonction de transfert. En développant cette expression, nous obtenons : $s^2 - (p_1 + p_2)s + p_1 p_2$.

En identifiant les deux expressions du dénominateur, k_1 et k_2 peuvent être exprimés en fonction de p_1 et p_2 :

$$\begin{cases} k_1 = p_1 p_2 \\ k_2 = -(p_1 + p_2) \end{cases}$$

Si des pôles complexes conjugués de la forme $p_{1,2} = x \pm i\, y$ sont choisis, les gains peuvent s'exprimer directement en fonction des coordonnées x et y des pôles :

$$\begin{cases} k_1 = x^2 + y^2 \\ k_2 = -2x \end{cases}$$

Les pôles choisis pour satisfaire la dynamique de l'erreur latérale ne peuvent pas être plus rapides que ceux du système. De ce fait, nous avons choisi des pôles conjugués égaux à $p_{1,2} = -1 \pm i$. Ces pôles donnent les gains suivants :

$$\begin{cases} k_1 = 1 \\ k_2 = 2 \end{cases}$$

Ce sont ces gains qui vont être utilisés pour la simulation comme pour l'expérimentation. Ils seront ajustés au besoin.

1.3. Calcul de l'erreur latérale

1.3.1. Cas du suivi de ligne droite

Nous avons considéré, dans ce premier cas, que le chemin à suivre est une ligne droite horizontale, dont l'ordonnée est quelconque.

Dans ce cas, le calcul de l'erreur latérale est immédiat : l_{os} correspond à la différence entre la position selon Y du robot et l'ordonnée de la ligne à suivre.

Le schéma ci-dessous illustre cette erreur latérale :

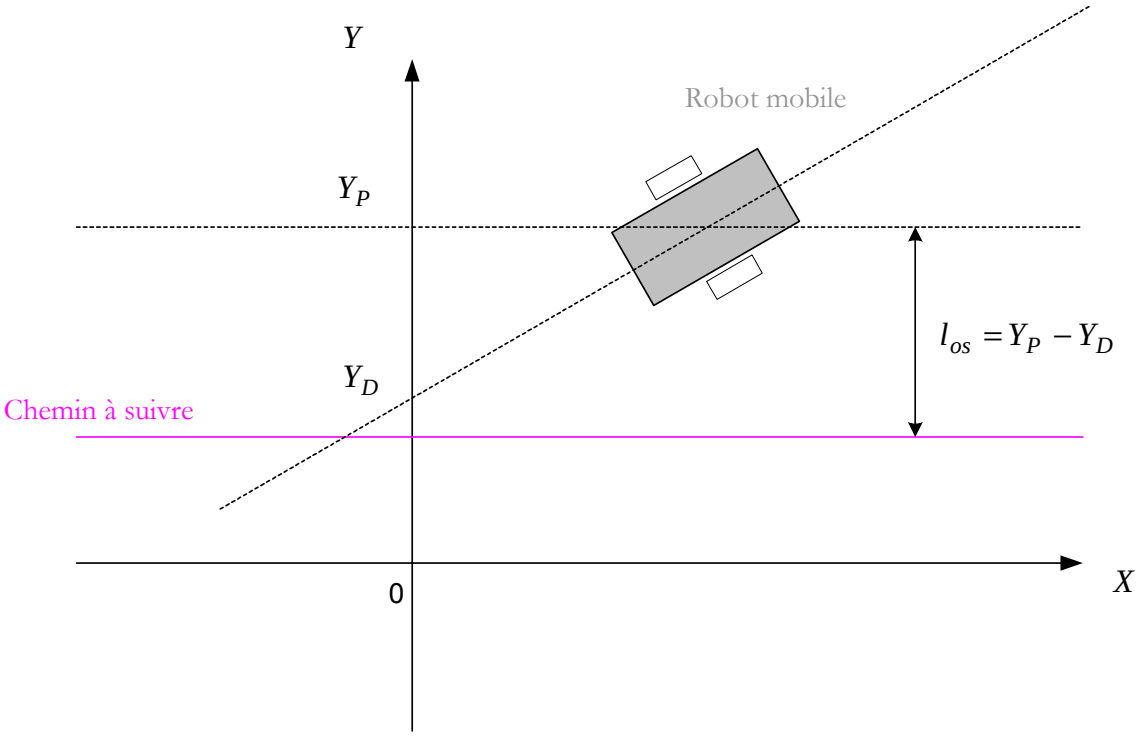


Figure 4.2 *Suivi de ligne droite*

1.3.2. Cas du suivi de cercle

Dans ce second cas, le calcul de l'erreur latérale n'est pas immédiat.
Le schéma ci-dessous présente la configuration du suivi de cercle ainsi que les variables qui vont être utilisées pour calculer l_{os}.

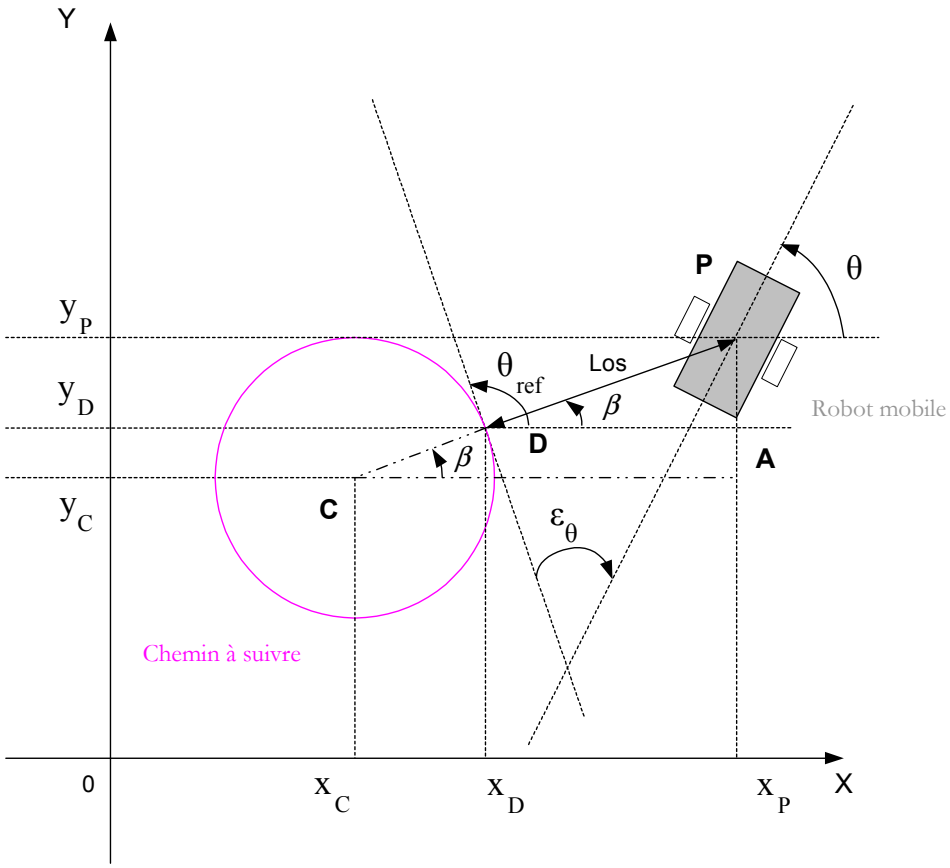


Figure 4.3 *Suivi de cercle*

L'erreur de position latérale va être calculée à partir de schéma.

Considérons le triangle ADP, rectangle en A.
On a :

$$\begin{cases} \cos\beta = \dfrac{x_P - x_D}{l_{os}} \\ \sin\beta = \dfrac{y_P - y_D}{l_{os}} \end{cases} \qquad [4.2]$$

où (x_P, y_P) sont les coordonnées du robot, et (x_D, y_D) celles du point de la courbe où devrait se trouver le robot.

Les coordonnées du point D peuvent s'exprimer par:

$$\begin{cases} x_D = x_C + R_C \cos(\beta) \\ y_D = y_C + R_C \sin(\beta) \end{cases} \quad [4.3]$$

où (x_C, y_C) sont les coordonnées du centre du robot et R_C son centre.

L'angle β vaut, quant à lui:

$$\beta = atan\left(\frac{y_P - y_C}{x_P - x_C}\right) \quad [4.4]$$

Dans les tests, tant en simulation qu'en expérimentation, la fonction $atan(\ )$ a été remplacée par $atan2(\ )$. Celle-ci correspond à l'arc tangente exprimée dans les 4 cadrans.

En multipliant la première équation de [4.2] par $\cos\beta$ et la seconde par $\sin\beta$, et en sommant les deux équations ainsi construites, on obtient :

$$\cos^2\beta + \sin^2\beta = \frac{x_P - x_D}{l_{os}}\cos\beta + \frac{y_P - y_D}{l_{os}}\sin\beta = 1$$

$$\Rightarrow \quad l_{os} = (x_P - x_D)\cos\beta + (y_P - y_D)\sin\beta$$

Ce qui fournit une expression de l'erreur latérale l_{os} en fonction d'un angle connu.

Enfin, en remarquant que $\beta = \theta_{ref} - \frac{\pi}{2}$, on peut réécrire cette expression en fonction de l'angle θ_{ref} :

$$l_{os} = (x_P - x_D)\cos\left(\theta_{ref} - \frac{\pi}{2}\right) + (y_P - y_D)\sin\left(\theta_{ref} - \frac{\pi}{2}\right)$$

$$\Rightarrow \quad l_{os} = -(x_P - x_D)\sin\theta_{ref} + (y_P - y_D)\cos\theta_{ref} \quad [4.5]$$

L'angle θ_{ref} dépend lui-même de l'erreur d'orientation ε_θ et de l'angle d'orientation du robot θ :

$$\theta_{ref} = \theta + \varepsilon_\theta$$

2. SIMULATION DU SUIVI DE CHEMIN

2.1. Cas du suivi de ligne

2.1.1. Schéma bloc de la simulation

Comme nous l'avons montré plus avant, l'erreur latérale est simple à calculer. Rappelons qu'elle correspond à la différence d'ordonnée (si la ligne est horizontale dans le plan XY) entre la position du robot et la ligne à suivre.

A la simulation en boucle fermée, seuls les calculs de l'erreur latérale l_{os} et de la commande de vitesse angulaire Ω^* ont été ajoutés.

Voici le programme Simulink réalisé :

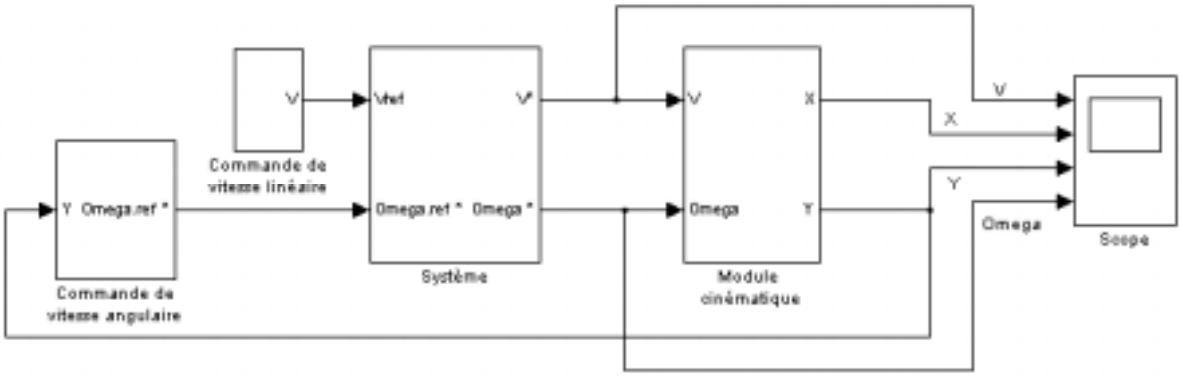


Figure 4.4 *Programme de simulation du suivi de ligne*

Le bloc intitulé « système » comprend les blocs suivant : un bloc de dynamique du système (système et contrôleur) et deux blocs qui transforment les vitesses linéaires et angulaires en vitesse appliquée à chaque roue, et réciproquement.

Le module cinématique permet d'obtenir la position du système à partir des vitesses linéaire et angulaire, et de l'angle d'orientation du robot.

Le bloc « commande de vitesse linéaire » génère un profil de vitesse trapézoïdal.

Pour plus de précisions, ces différents blocs sont détaillés en annexe.

Enfin, et il va être décrit plus en détail, le bloc « commande de vitesse angulaire » permet de calculer la loi de commande.

Voici ce qu'il y a dans la première couche intérieure dudit bloc :

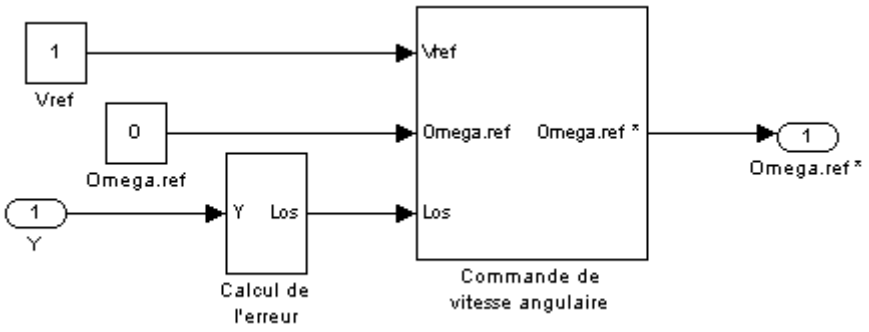


Figure 4.5 *Détails du bloc de commande de vitesse angulaire*

Nous distinguons deux blocs : le bloc de calcul de la commande proprement dite, et celui du calcul de l'erreur. Ce dernier effectue la différence entre l'ordonnée Y du robot, et l'ordonnée Y_C de la ligne à suivre. Le premier est présenté ci-dessous, il est commun au deux sortes de suivi de chemin considéré dans ce travail :

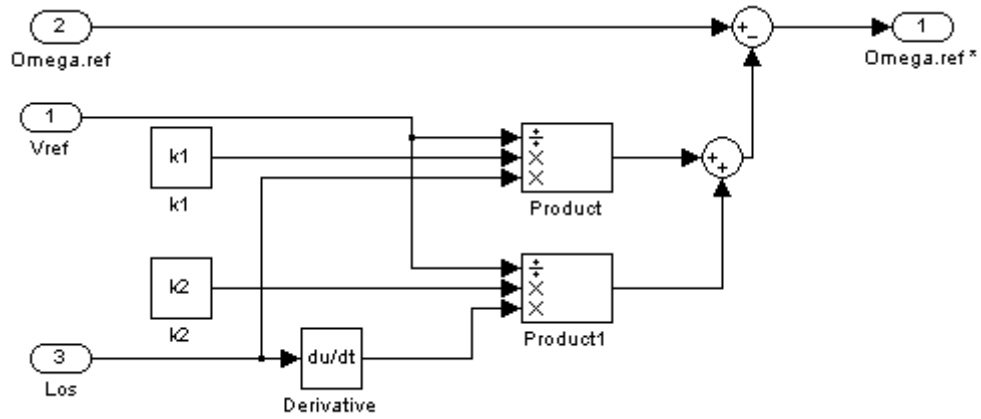


Figure 4.6 *Calcul de la loi de commande sur Simulink*

La simulation s'exécute assez lentement, mais elle donne d'excellents résultats qui seront présentés au paragraphe suivant.

2.1.2. Résultats

La courbe [4.7] présente le trajet effectué par le robot, en simulation, pour rejoindre la ligne droite. Les conditions initiales sont les suivantes:

- $x(0)=0$
- $y(0)=1$
- $\theta(0)=0$

Ces valeurs sont implantées comme conditions initiales des intégrateurs correspondants du modèle de simulation.

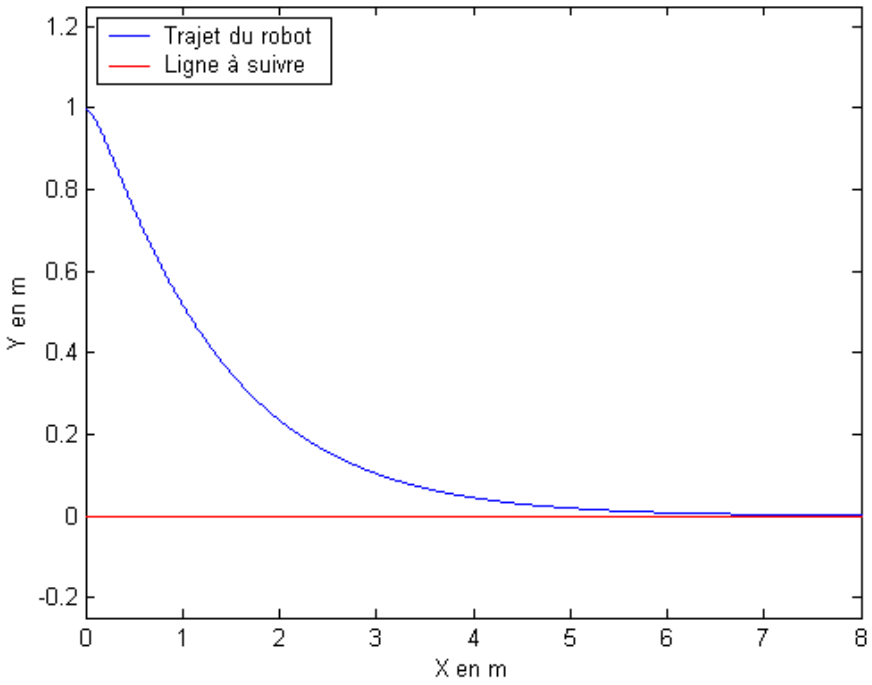


Figure 4.7 *Simulation du suivi de ligne droite – Graphe de y=f(x)*

Le robot rejoint le chemin à suivre selon une dynamique très satisfaisante.
Il est intéressant de tracer la position du robot selon Y en fonction du temps:

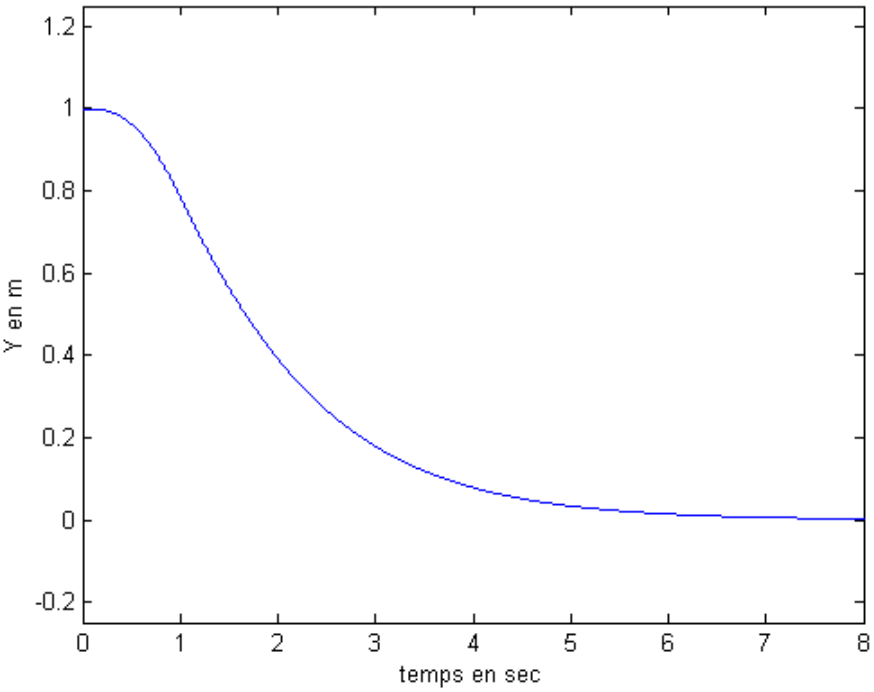


Figure 4.8 *Simulation du suivi de ligne droite – Graphe de y=f(t)*

Cette courbe permet de voir la rapidité avec laquelle le robot parvient à annuler son erreur de suivi latérale.

Dans ce test, la ligne est placée en zéro, mais il est clair qu'elle peut-être placée en une ordonnée quelconque. Cependant, dans le calcul de la loi de commande, l'approximation des petits angles est considérée. Le modèle ne sera donc pas fonctionnel si l'erreur latérale est trop importante: le robot, en voulant rejoindre la ligne, va créer une erreur d'orientation trop importante qui ne permettra pas de calculer la loi de commande nécessaire à annuler l'erreur de suivi latérale.

2.2. Cas du suivi de cercle

2.2.1. Schéma bloc de la simulation

Dans le cas du suivi de cercle, la simulation a nécessité davantage de réflexion. D'une part, l'erreur latérale est plus lourde à calculer, et de l'autre, il faut générer une trajectoire circulaire pour avoir une référence permettant de calculer l'erreur latérale.

Le schéma de la simulation est identique à celui de la figure [4.4], mais dans le bloc « commande de vitesse angulaire », représenté à la figure [4.5], le calcul de l'erreur latérale diffère. Ce bloc est représenté à la figure suivante:

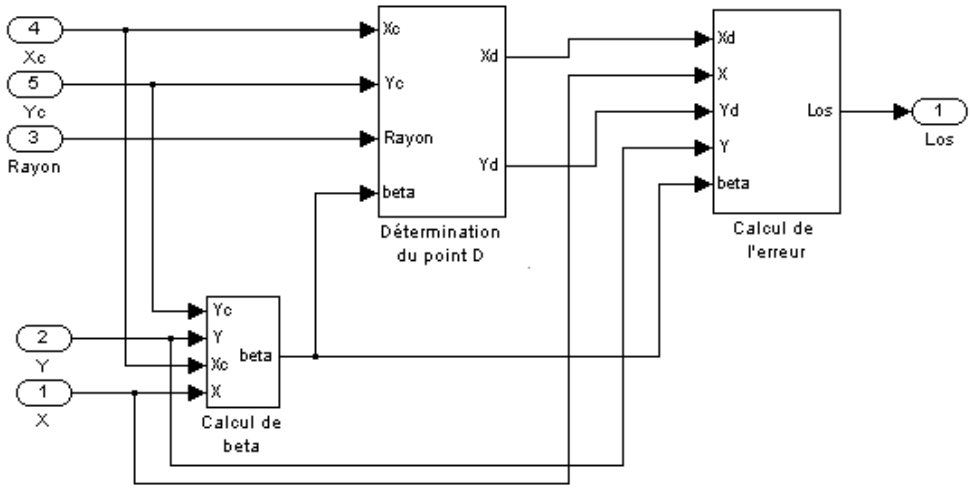


Figure 4.9 *Programme Simulink de calcul de l'erreur latérale dans le cas du suivi de cercle*

Un profil de trajectoire circulaire y est généré à partir de l'angle β et des coordonnées x_c et y_c du centre du cercle, afin de déterminer le point de la courbe où devrait se trouver le robot. Ceci est réalisé dans le bloc « Détermination du point D », et utilise la formule [4.3].

L'angle β est lui-même calculé dans un autre bloc selon la formule [4.4].

L'erreur est calculée selon la formule [4.5], à partir de l'angle θ_{ref} et des coordonnées du robot et du point D.

Tous ces blocs sont présentés en annexe.

2.2.2. Résultats

La courbe suivante présente le trajet effectué par le robot pour rejoindre le cercle. Lors du premier tracé, le chemin à suivre avait été superposé, mais quand l'erreur de suivi s'annule, les deux tracés se confondent et on ne voit alors qu'une seule de deux courbes. Cela prêtait à confusion.

Les conditions initiales de cet essai sont les suivantes:

- $x_0 = 4$
- $y_0 = 4$
- $\theta_0 = \frac{\pi}{2}$

Le cercle à suivre est centré en $(0,0)$ et a un rayon égal à 4 m.

Le tracé de la position du robot est présenté ci-dessous :

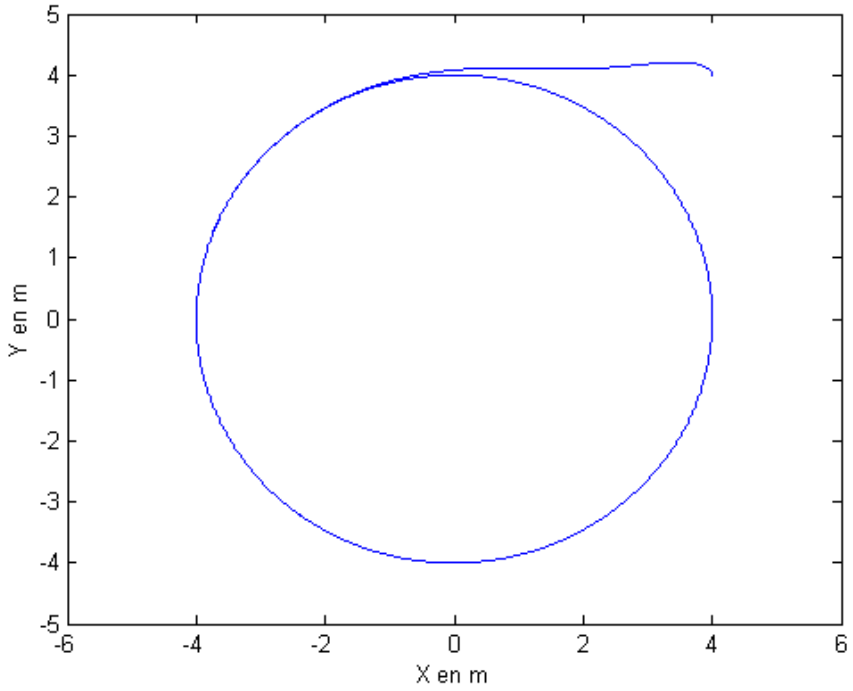


Figure 4.10 *Simulation du suivi de cercle pour une position initiale du robot en* $x_0 = 4$, $y_0 = 4$ *- Graphe de y=f(x)*

Ici encore, la simulation montre que le robot rejoint rapidement le cercle en annulant son erreur de suivi latérale.

Ces résultats satisfaisants nous permettent de passer à l'implantation du suivi de chemin sur le robot.

3. EXPÉRIMENTATION DU SUIVI DE CHEMIN

3.1. Protocole des tests

Le suivi de chemin a été implanté sur le robot, en C++, conformément à la structure. L'erreur latérale ainsi que la commande de vitesse angulaire sont calculées dans la boucle. Les calculs de position du robot utilisent les équations de cinématique, où les intégrations sont effectuées de manière numérique:

$$\begin{cases} \theta(i) = \theta(i-1) + \Omega(i)\Delta T(i) \\ x(i) = x(i-1) + v(i)\cos[\theta(i)]\Delta T(i) \\ y(i) = y(i-1) + v(i)\sin[\theta(i)]\Delta T(i) \end{cases}$$

Nous pouvons alors déterminer l'erreur de suivi latéral, selon les formules définies précédemment. Celles-ci permettent de calculer la commande de vitesse angulaire :

$$\Omega^*(i) = \Omega(0) - \frac{k_1}{v(0)} l_{os} - \frac{k_2}{v(0)} \frac{y(i) - y(i-1)}{\Delta T(i)}$$

Avec cette nouvelle vitesse, nous calculons la vitesse à envoyer à chaque roue, et la lui transmettons.

Le pseudo code des programmes de suivi de chemin général (suivi de ligne droite ou de cercle) est présenté ci-dessous. Le code complet est disponible en annexe.

```
Déclaration des données
Initialisation des données
Conversion des données
Initialisation de la carte en boucle fermée
Envoi des commandes au registre
Application simultanée des commandes
Boucle
     Lecture du temps
     Lecture des encodeurs
     Calcul de la vitesse linéaire
     Calcul de la vitesse angulaire
     Calcul de l'angle thêta
     Calcul de la position du robot, selon X et Y
     Calcul de l'erreur latérale
     Calcul de la nouvelle vitesse angulaire Ω*
     Envoi des commandes
Multi update pour envoyer les commandes simultanément
     Boucle d'attente
     Incrémentation des indices
Tant que la distance désirée n'est pas atteinte
Remise à zéro des vitesses
Stockage des résultats dans un fichier
```

3.2. Résultats

3.2.1. Cas du suivi de ligne

Plusieurs configurations, tant au niveau de la position initiale que des vitesses linaire et accélération, ont été testées. Les résultats obtenus pour quelques configurations sont présentés. Pour tous ces tests, l'angle d'orientation initiale est nul.

Première configuration :

- $x_0 = 0\,m$
- $y_0 = 0.5\,m$
- $V = 0.5\,m.s^{-1}$
- $\Omega = 0\,rad.s^{-1}$
- $a = 1\,m.s^{-2}$

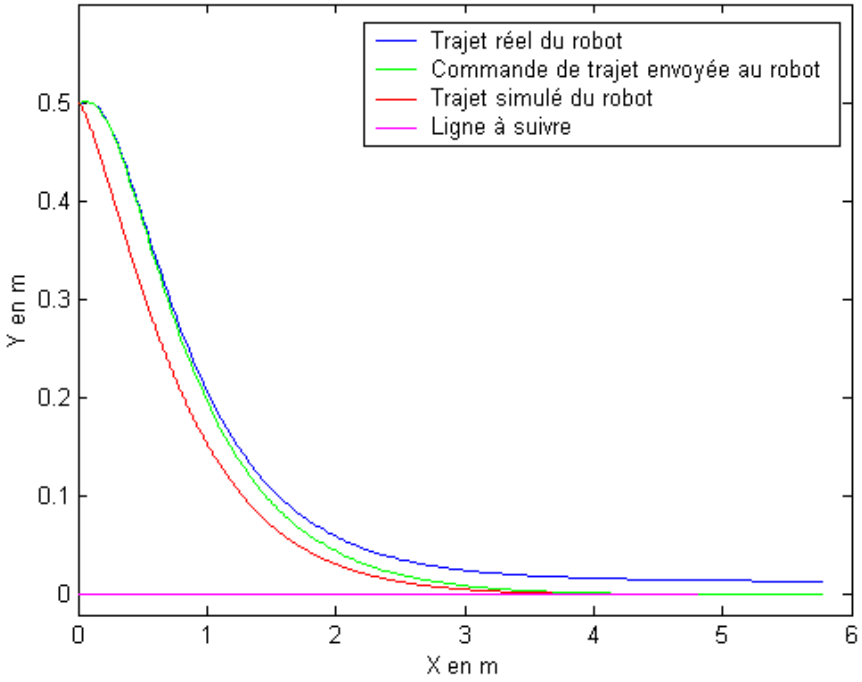


Figure 4.11 *Suivi de ligne pour la configuration* $y_0 = 0.5\,m$, $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de la position*

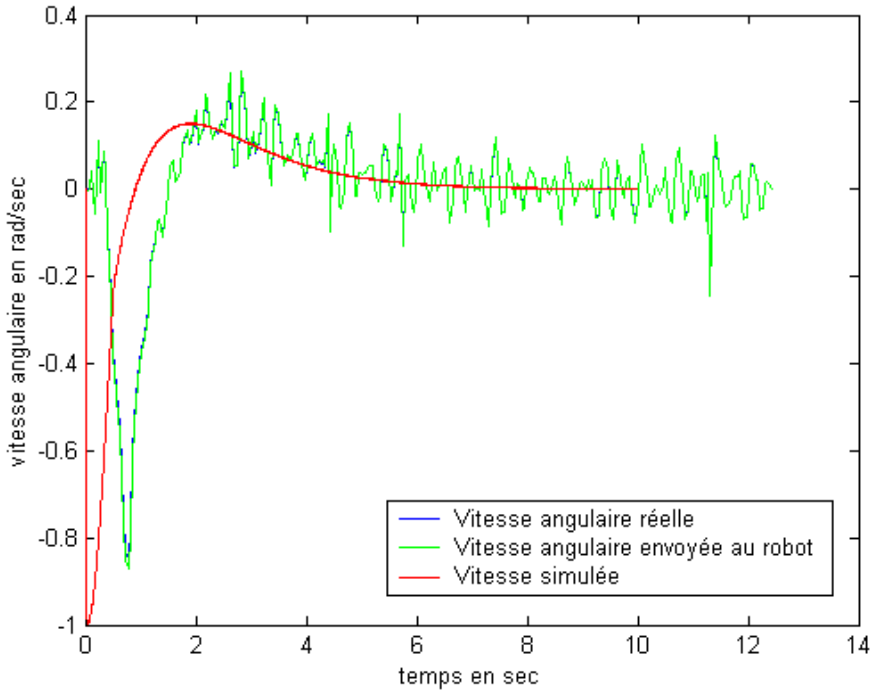


Figure 4.12 *Suivi de ligne pour la configuration* $y_0 = 0.5\,m$, $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de la vitesse angulaire*

Le décalage observé entre les courbes de position simulée et expérimentale est confirmé par le tracé de la vitesse angulaire. En simulation, une vitesse angulaire non nulle est envoyée dès le début. Par contre, la vitesse angulaire expérimentale est « retardée » dans le cas de l'expérimentation. Deux raisons peuvent être à la base de cette erreur : soit il subsiste une erreur dans le programme de suivi de ligne, soit, et ce sera vérifié avec le prochain robot, les erreurs d'odométrie sont en cause. Les position et orientation du robot qui sont renvoyées à la carte de contrôle sont erronées. En effet, la lecture des encodeurs n'est pas simultanée. De plus, les tests de translation/rotation en boucle fermée ont montré les mêmes erreurs. Le glissement des roues sur le sol est une des causes. Le robot rejoint le chemin qu'il doit suivre, mais plus lentement que lors de la simulation. A la fin, des erreurs en régime permanent apparaissent.

Voici les conditions initiales de la seconde configuration :

- $x_0 = 0\,m$
- $y_0 = 1\,m$
- $V = 0.5\,m.s^{-1}$
- $\Omega = 0\,rad.s^{-1}$
- $a = 1\,m.s^{-2}$

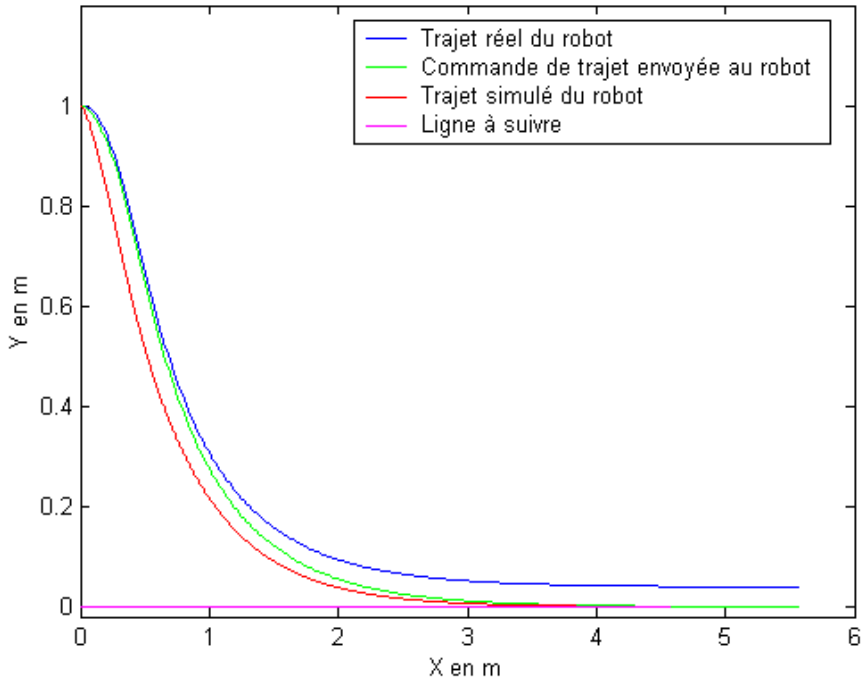


Figure 4.13 *Suivi de ligne pour la configuration* $y_0 = 1\,m$, $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de la position*

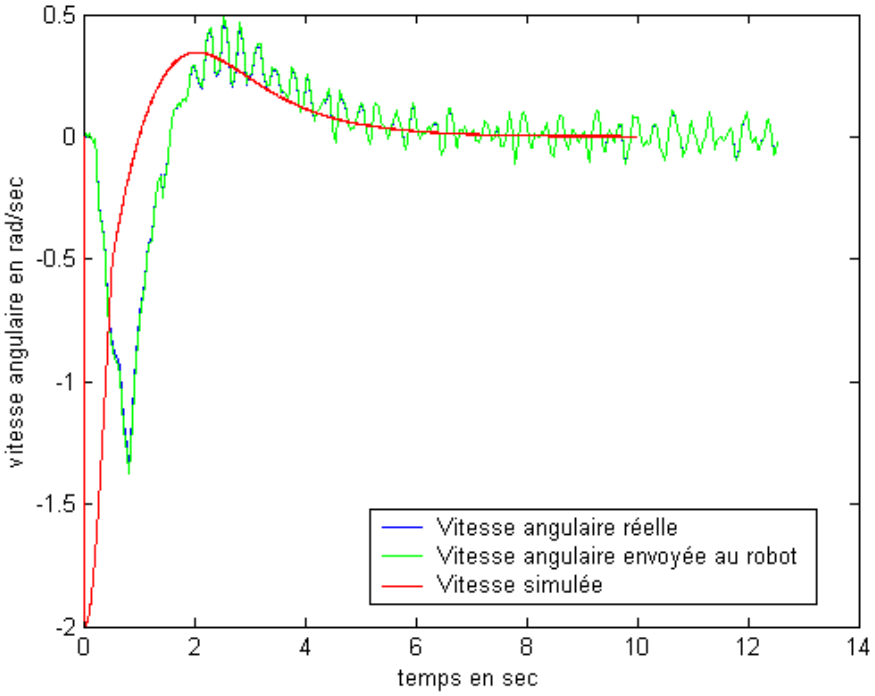


Figure 4.14 *Suivi de ligne pour la configuration* $y_0 = 1\,m$, $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de la vitesse angulaire*

Les mêmes conclusions sont applicables à ce deuxième essai. La vitesse angulaire générée est plus grande en simulation qu'en expérimentation. De ce fait, le robot converge plus vite vers le chemin à suivre.

Quelques tests supplémentaires à vitesse linéaire et/ou accélération plus élevées ont été effectués. Les courbes d'un des tests sont présentées ci-contre.

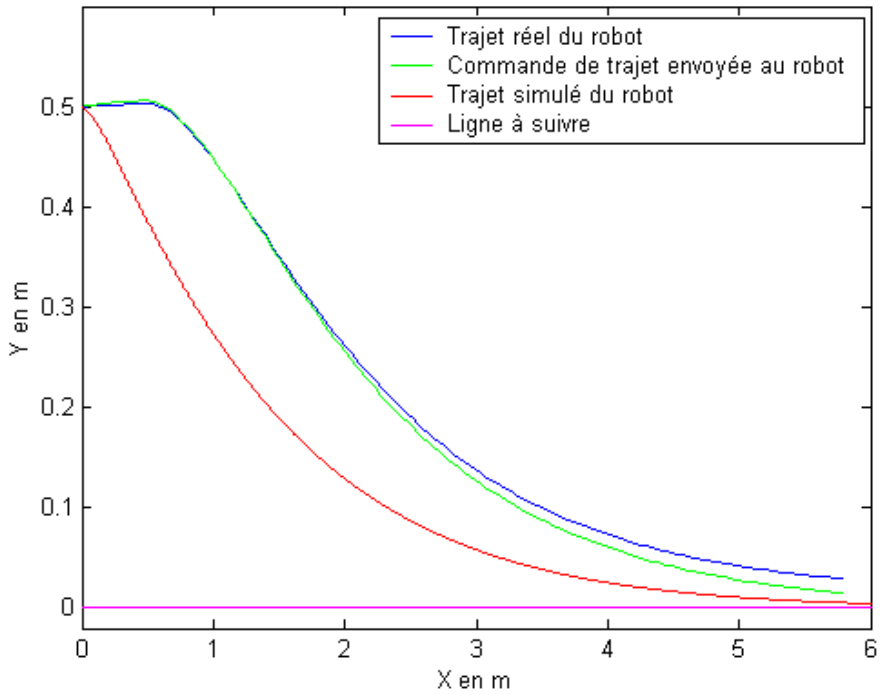


Figure 4.15 *Suivi de ligne pour la configuration* $y_0 = 0.5\,m$, $V = 1\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de la position*

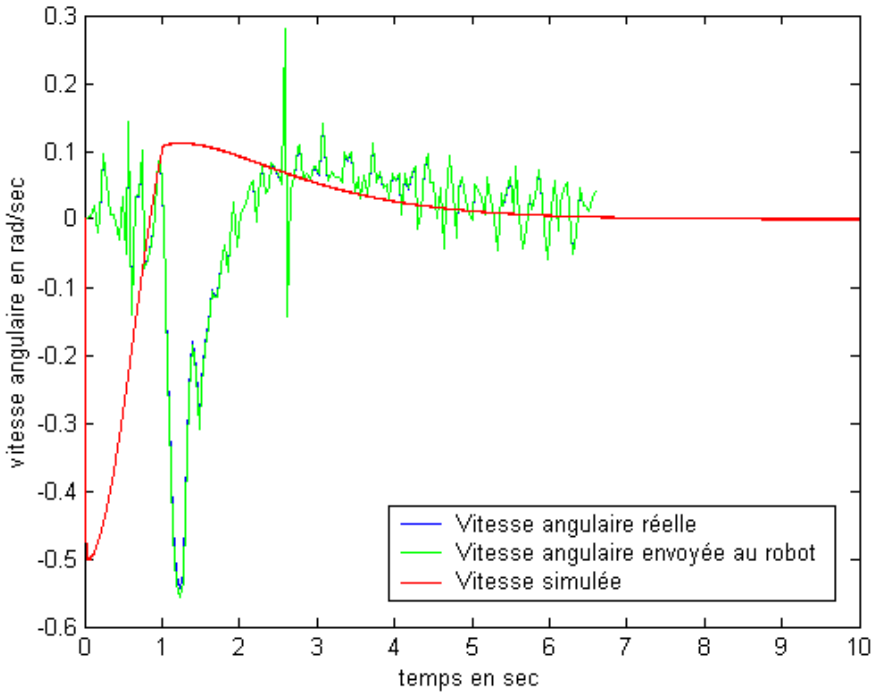


Figure 4.16 *Suivi de ligne pour la configuration* $y_0 = 0.5\,m$, $V = 1\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de la vitesse angulaire*

Il est également intéressant de comparer les courbes de position du robot selon Y, en fonction du temps, et ce pour des tests à vitesse différente.

La position initiale du robot es identique dans les deux cas, soit :

- $x_0 = 0\,m$
- $y_0 = 0.5\,m$

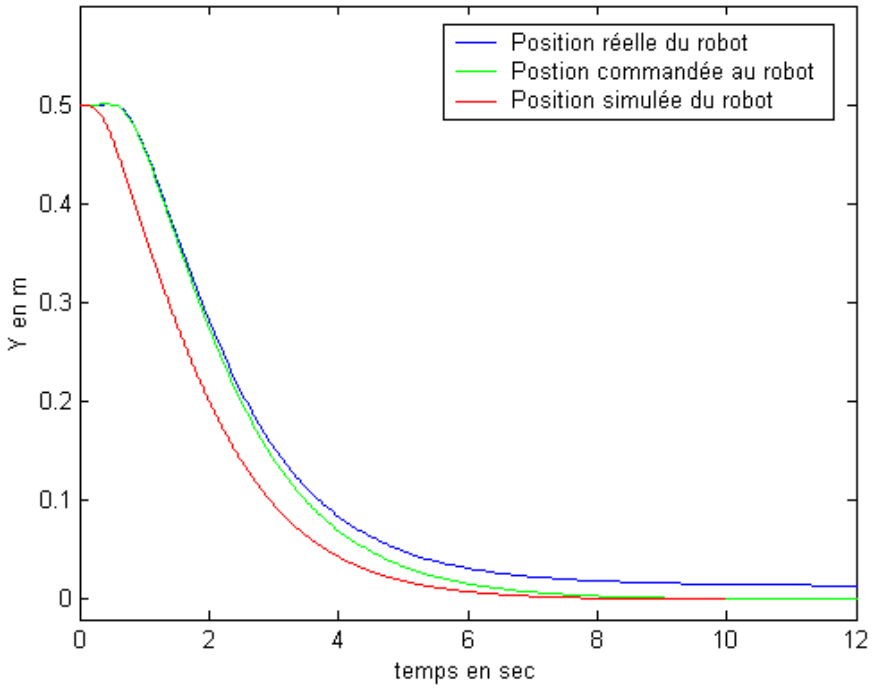


Figure 4.17 *Suivi de ligne pour la configuration* $y_0 = 0.5\,m$, $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de y=f(t)*

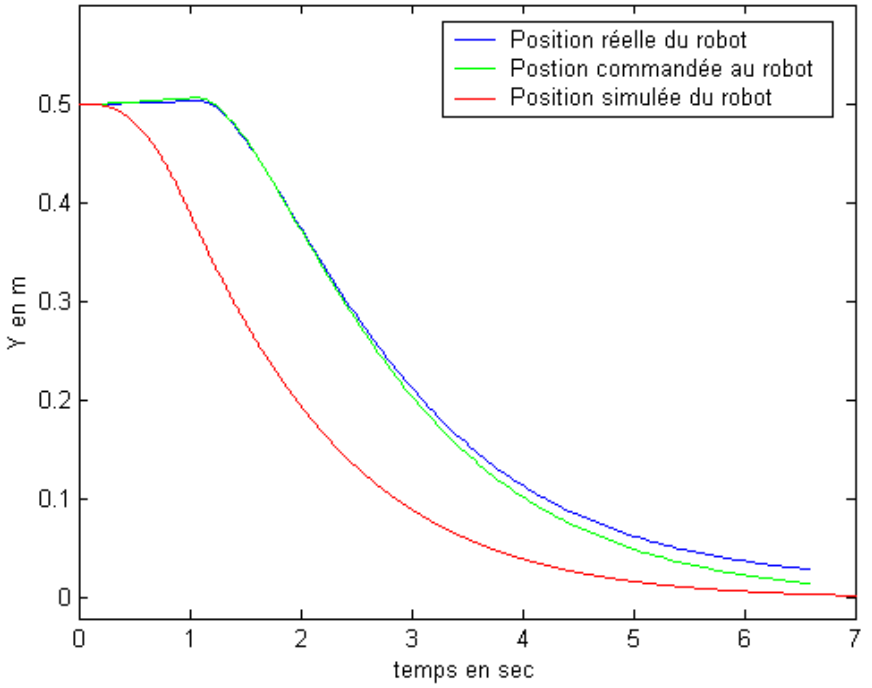


Figure 4.18 *Suivi de ligne pour la configuration* $y_0 = 0.5\,m$, $V = 1\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *– Tracé de y=f(t)*

Malgré le changement d'échelle, les courbes sont sensiblement pareilles. Cela confirme que la dynamique de l'erreur ne dépend pas de la vitesse.

3.2.2. Cas du suivi de cercle

Pour ce second suivi de chemin, un test sans erreur initiale a d'abord été effectué. Le robot devrait suivre parfaitement le cercle. Les courbes de position et de vitesse angulaire de cet essai sont présentées ci-dessous. Sauf mention contraire, les tests sont effectués pour une orientation initiale égale à $\frac{\pi}{2}$.

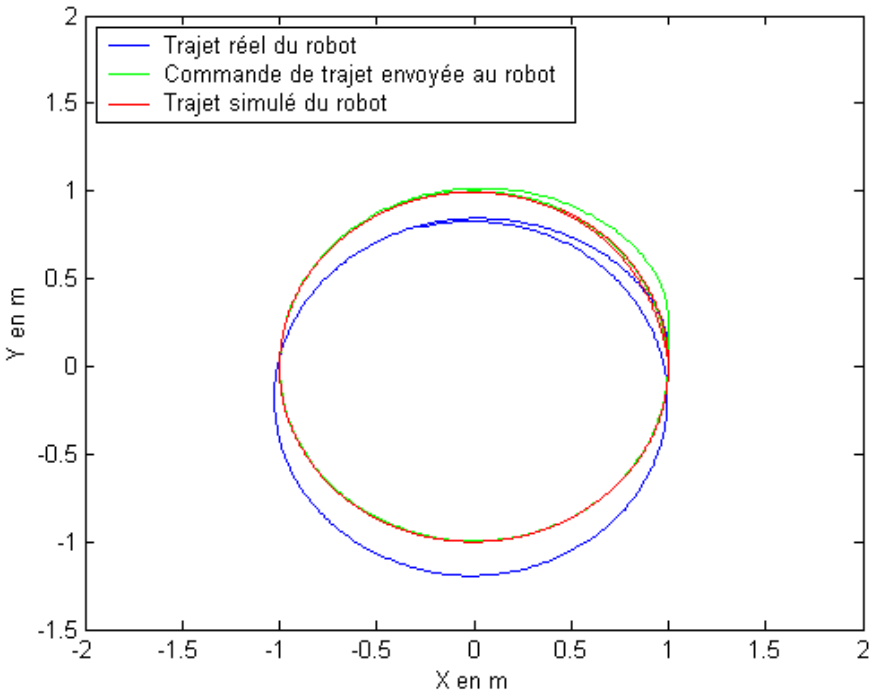


Figure 4.19 *Suivi de cercle pour la configuration* $x_0 = 1\,m$, $y_0 = 0\,m$ $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *- Tracé de la position*

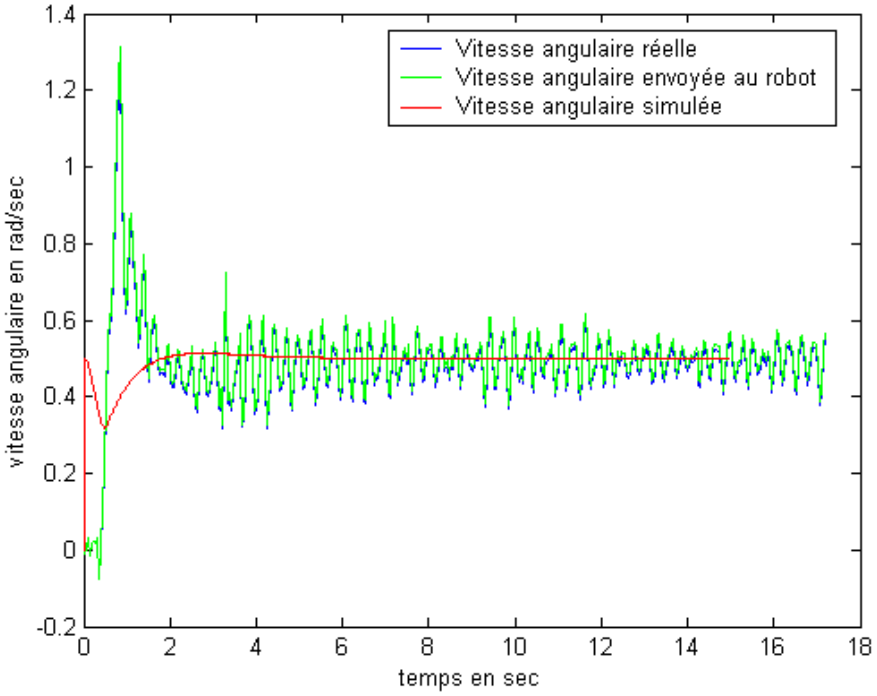


Figure 4.20 *Suivi de cercle pour la configuration* $x_0 = 1\,m$, $y_0 = 0\,m$ $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *- Tracé de la vitesse angulaire*

En simulation, le robot se comporte parfaitement bien. Par contre, du point de vue expérimental, les résultats sont mitigés : le tracé de la position réelle du robot montre que celui-ci accumule une erreur au départ, mais il ne la corrige pas par la suite. Le robot effectue un cercle, mais qui est décalé du « vrai » cercle. Pourtant, la commande qui lui est envoyée est conforme, ou presque, à la simulation. Ici encore, soit une erreur dans le programme, soit les glissements du robot sur le sol sont en cause. Néanmoins, cette seconde hypothèse est fort probable : lors de l'accélération initiale, il est possible que le robot dérape. La position relevée par les encodeurs est alors acceptable du point de vue de l'erreur. L'absence d'erreur fait que la commande de vitesse n'est pas modifiée, et le robot se décale du chemin théorique. Il corrige alors l'erreur par rapport à un chemin où il croit être, mais pas par rapport au chemin où il est. Un système extérieur serait nécessaire pour localiser le robot.

Un test avec une légère erreur initiale a été réalisé. Les résultats sont présentés ci-dessous :

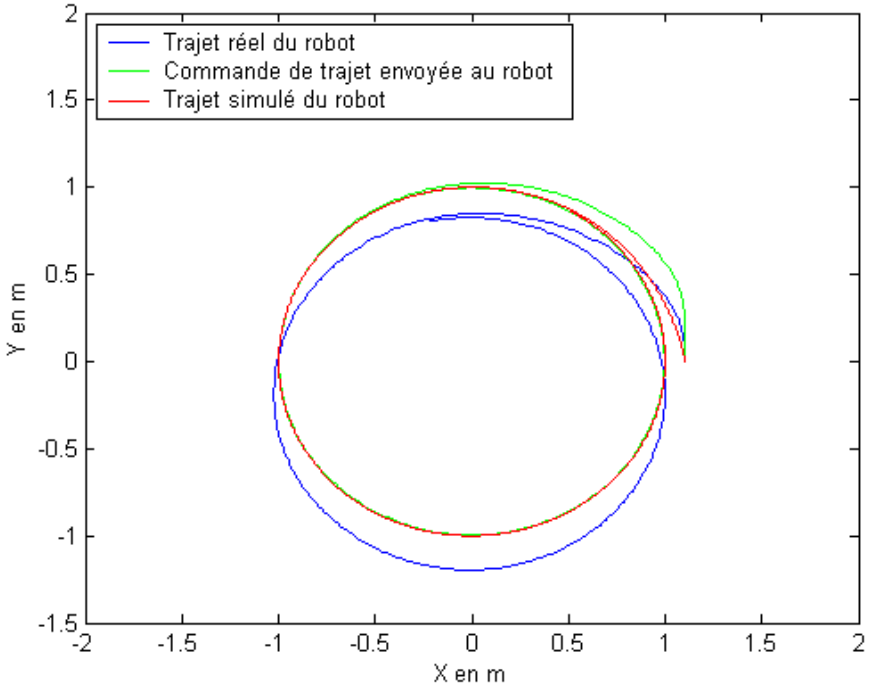


Figure 4.21 *Suivi de cercle pour la configuration* $x_0 = 1.1\,m$, $y_0 = 0\,m$ $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *- Tracé de la position*

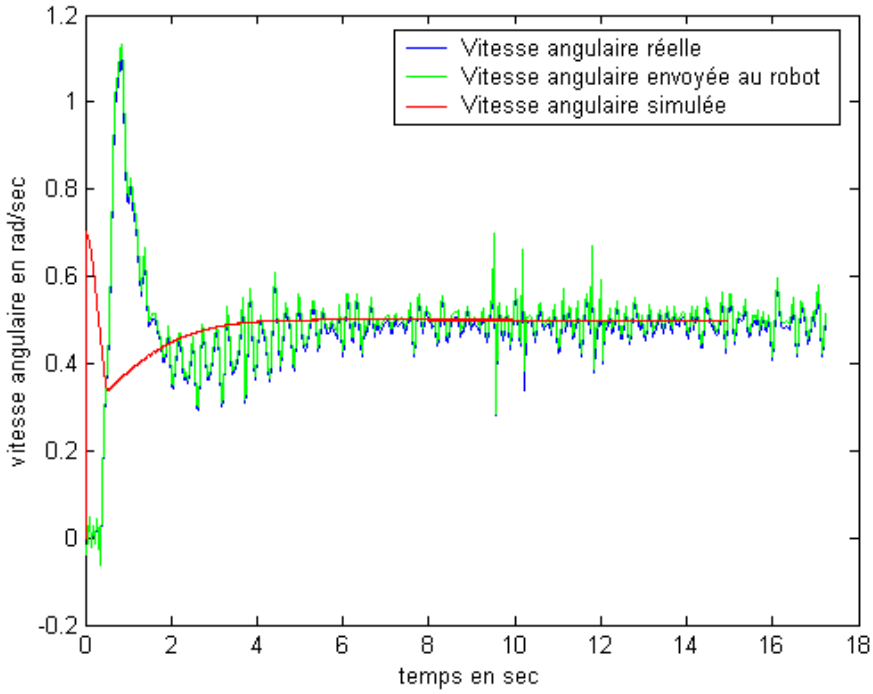


Figure 4.22 *Suivi de cercle pour la configuration* $x_0 = 1.1\,m$, $y_0 = 0\,m$ $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *- Tracé de la vitesse angulaire*

Enfin, un test avec des conditions initiales plus critiques est présenté :

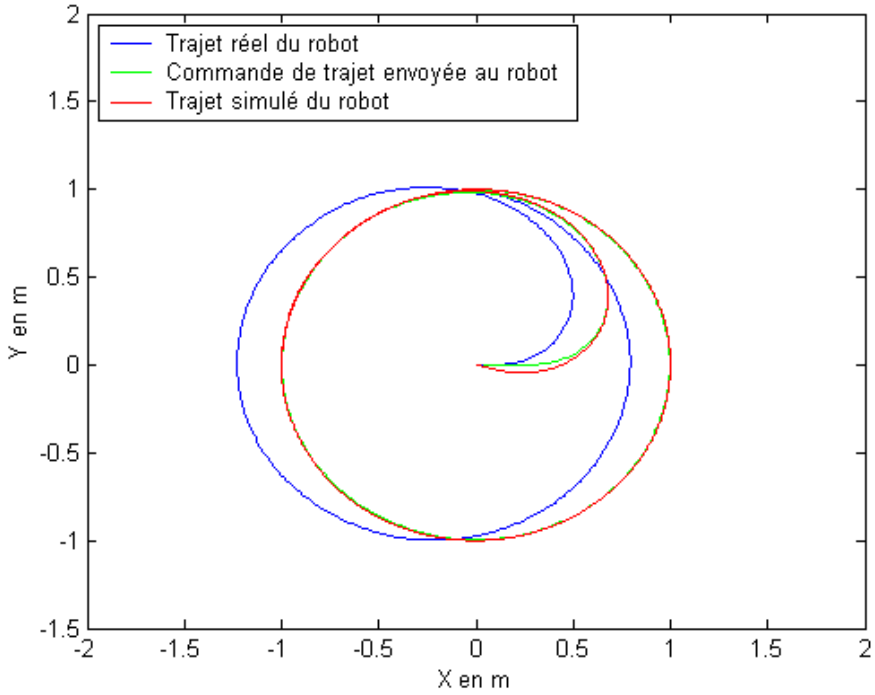


Figure 4.23 *Suivi de cercle pour la configuration* $x_0 = 0\,m$, $y_0 = 0\,m$, $\theta_0 = 0\,rad$ $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *- Tracé de la position*

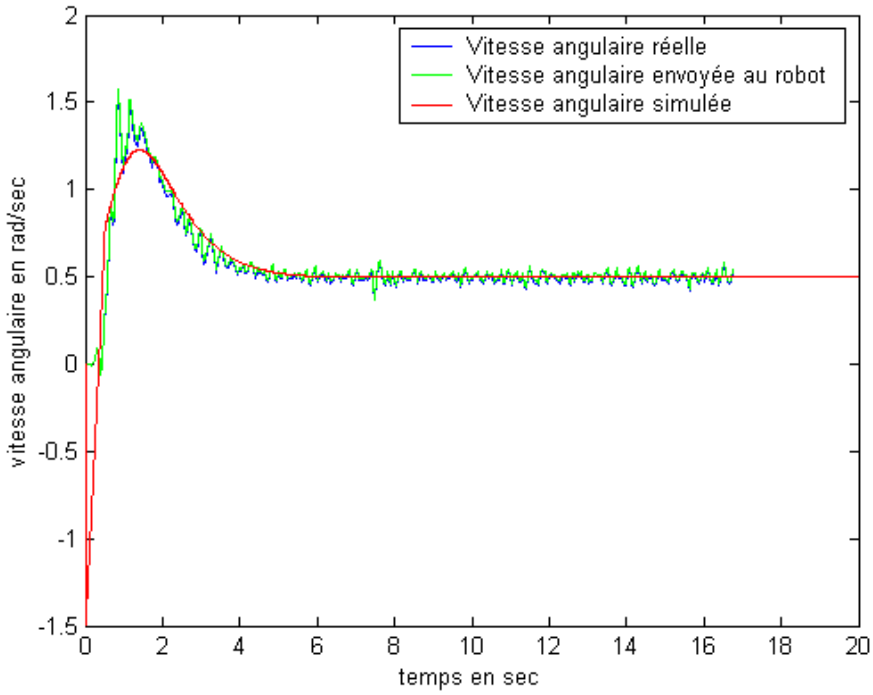


Figure 4.24 *Suivi de cercle pour la configuration* $x_0 = 0\,m$, $y_0 = 0\,m$, $\theta_0 = 0\,rad$ $V = 0.5\,m.s^{-1}$, $a = 1\,m.s^{-2}$ *- Tracé de la vitesse angulaire*

Dans les deux cas, les courbes de position obtenues en expérimentation présentent un léger décalage par rapport à la commande envoyée, et par rapport à la simulation. Les conclusions sont identiques à celles du premier essai : le robot dérape sur le sol au démarrage. Une accumulation d'erreurs d'odométrie créée dès le départ empêche le robot d'éliminer l'erreur par la suite. La mesure des encodeurs n'est pas fiable, car elle ne se fait pas simultanément.

CONCLUSION

Ce travail a débuté avec la prise en main du système robotique. Les composantes du robot ont été analysées afin d'être validées. Du point de vue mécanique, le choix des composantes constitue un travail délicat car, et cela s'est vu tout au long du projet, les performances du robot en dépendent grandement. Ainsi, des roues d'un plus grand diamètre et d'une autre texture ont été choisies. Conjointement à ce choix, de nouveaux moteurs ont été sélectionnés. La forme générale du robot, quant à elle, a été jugée satisfaisante et est donc conservée pour construire les nouveaux robots. En ce qui concerne les parties électronique et électrique, elles ont été analysées pour mieux comprendre le fonctionnement du robot.

Ces composantes validées et mieux connues de l'utilisateur, une modélisation du système a été réalisée. Ce mo0dèle comprend une partie cinématique et une partie dynamique. Les paramètres du modèle dynamique ont été, d'une part, calculés à partir des caractéristiques connues des composantes (données du manufacturier), et d'autre part, évaluées à partir d'essais expérimentaux (réponse à l'échelon). De grandes différences ont été constatées en comparant les deux approches utilisées. Cette différence s'explique par le fait que le modèle théorique n'inclut pas l'effet de la friction. Pour la suite du travail, les paramètres obtenus expérimentalement ont été utilisés.

Pour mettre au point les contrôleurs de vitesse en translation et en rotation, un simulateur a été développé. Dans chacun des cas, un contrôleur PD a été mis au point et testé expérimentalement. La correspondance avec les simulations s'avère très bonne et les performances des contrôleurs de vitesse rencontrent les spécifications. Des tests complémentaires n'ont pas permis de montrer l'influence positive importante des gains anticipatif et intégral sur les performances du système. Les tests de translation pure se sont révélés très bons. Cependant, la vitesse angulaire estimée est très bruitée et ceci limite les performances atteignables.

La même méthode a été utilisée pour le suivi de chemin. Le contrôleur de mouvement a été mis au point en simulation, et par la suite testé expérimentalement. La correspondance entre les résultats de simulation et expérimentaux est satisfaisante. Elle présente cependant quelques différences. Malgré l'envoi d'une commande conforme à la simulation, des erreurs en régime permanent subsistent. Le robot est décalé par rapport au chemin à suivre. Ces décalages s'expliquent en partie par les erreurs d'odométrie. Le robot ne pouvant se fier qu'à la mesure des encodeurs des moteurs, il ne peut suivre les chemins désirés avec précision. Le robot est très sensible aux perturbations provenant du sol notamment (poussières, piste tracée…). Ces questions devront être analysées plus en détail pour améliorer les performances du robot.

Le contrôle des moteurs est effectué par une carte de contrôle spécialement destinée à ce genre d'asservissement. Cette carte contrôle chacun des moteurs de façon indépendante. Elle a été testée tout au long du projet, et ses performances ne sont pas satisfaisantes. Les temps mis

pour une requête sont trop longs, ce qui entraîne une lecture non simultanée des déplacements des roues, ce qui entraîne des erreurs importantes sur le résultat des essais. La deuxième carte (du constructeur Ajeco) devra être testée. Quelques tests sur cette nouvelle carte ont permis d'observer que la structure informatique n'est pas mise en cause : les résultats du temps de requête pour cette carte sont beaucoup plus courts. Elle permettra probablement d'obtenir de meilleurs résultats, les problèmes mécaniques devant être résolus avec le prochain prototype.

BIBLIOGRAPHIE

1. **Richard Hurteau**, *Asservissements,* notes du cours ELE 3201, Ecole Polytechnique de Montréal, janvier 2002

2. **Romano M. De Santis**, *Dynamique des systèmes mécaniques sous contraintes holonomes et non holonomes,* notes du cours ELE 6207, Ecole Polytechnique de Montréal, janvier 2001

3. **Romano M. De Santis**, *On the motion control of autonomous vehicles,* notes du cours ELE 6207, Ecole Polytechnique de Montréal, mars 2002

4. **Robert N.Clark**, *Introduction to automatic control systems*, John Wiley and Sons, Inc, 1962

5. **Julien Beaudry**, *Projet Spinos : conception et contrôle d'un robot mobile à vitesses différentielles*, Ecole Polytechnique de Montréal, décembre 2001

6. **PMD**, *Advanced Multi-Axis Motion Control Chipset*, Pittman, Novembre 1997

7. **Ajeco**, *Servo Booster, Dual PWM Power Amplifier PC/104 Card*, *Technical Reference Manual*, Ajeco, janvier 1999

8. **Pittman**, *Bulletin LCG*, Pittman (http://www.pittmannet.com/122100.html), 2001

9. **IBM**, *IBM family of Microdrives*, IBM (www.ibm.com/storage/microdrive)

10. **Michael Barr**, *Introduction to Pulse Width Modulation*, EmbeddedSystems Programming, vol.14 n°10 Septembre 2001 (http://www.embedded.com/story/OEG20010821S0096)

11. **Francis Mailhot**, *Conception d'une architecture informatique sous QNXRTP pour le contrôle d'un robot rouleur*, Ecole Polytechnique de Montréal, avril 2002

12. **Benjamin C. Kuo**, *Automatic Control Systems*, Prentice Hall, Sixth Edition, 1991

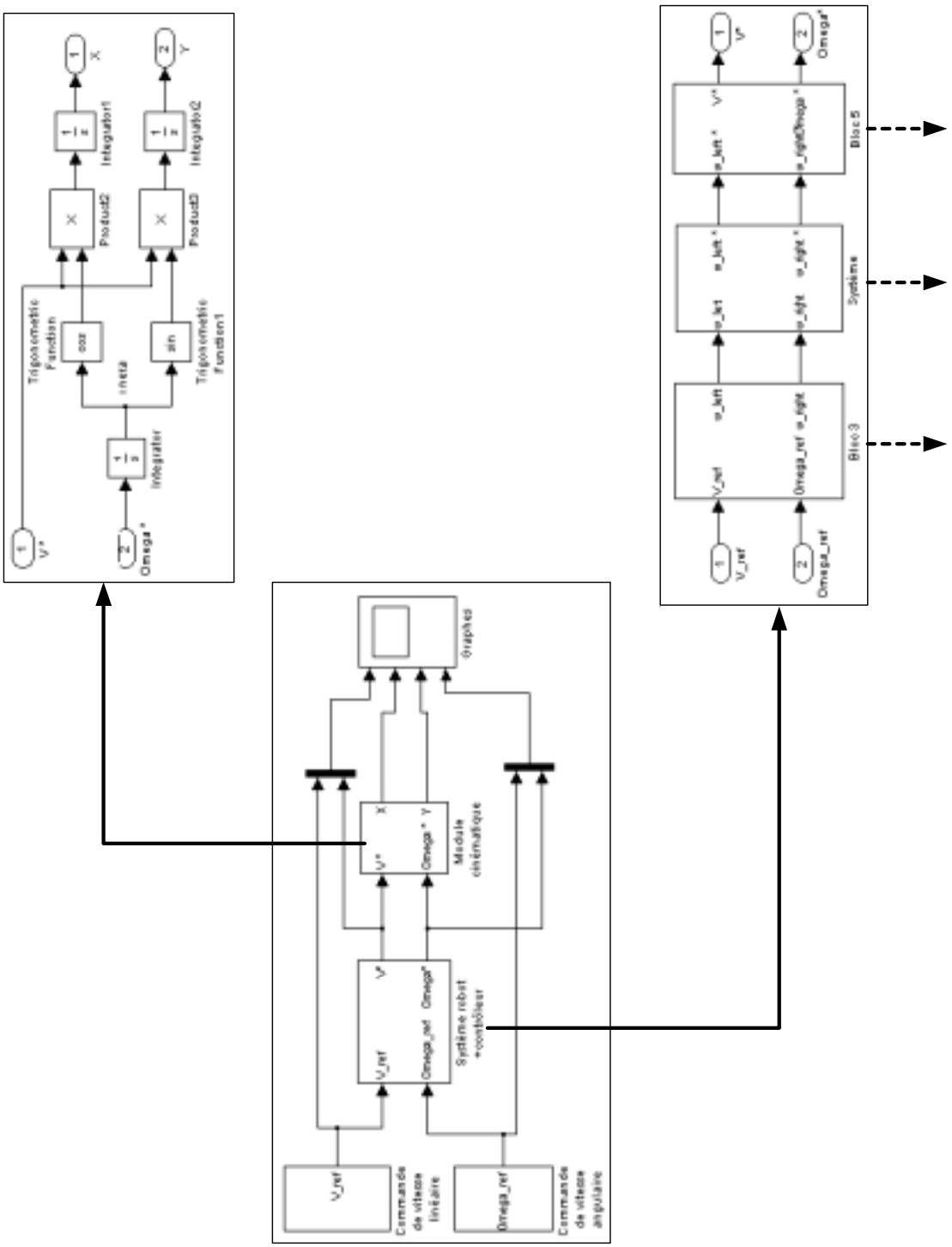
13. **Laurent Benon et Richard Hurteau**, *Suivi de chemin pour un véhicule électrique à deux roues motrices,* laboratoire n°4 du cours ELE 4201, Ecole Polytechnique de Montréal, automne 2001

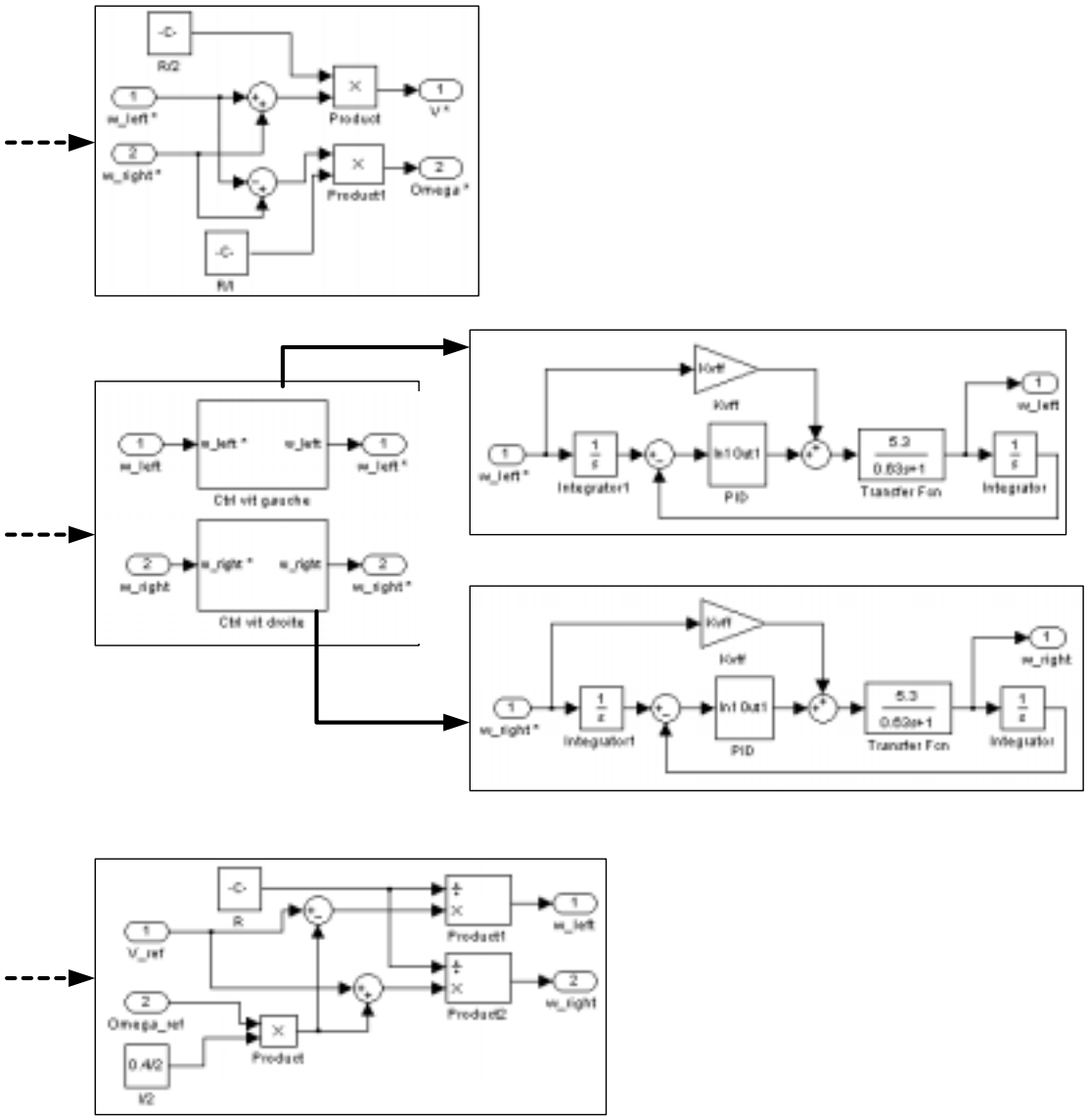
ANNEXES

1. ANNEXE A

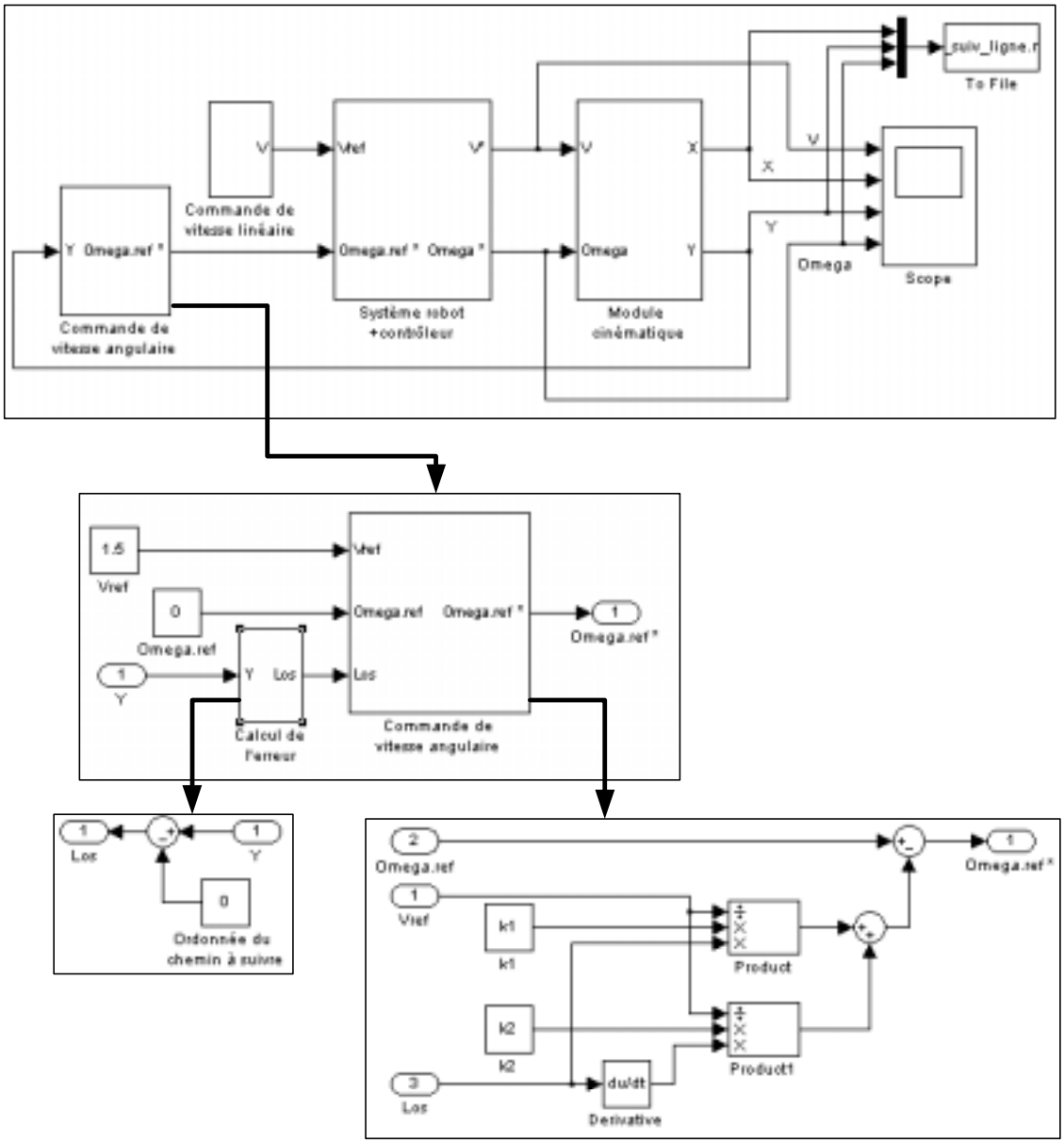
Dans cette première annexe, les simulations réalisées sont présentées en détails. Elles sont disponibles sur le CD d'annexe (cf. Annexe C), et sont programmées sur Matlab 6.1.

1.1. Programme Simulink de la simulation en boucle fermée

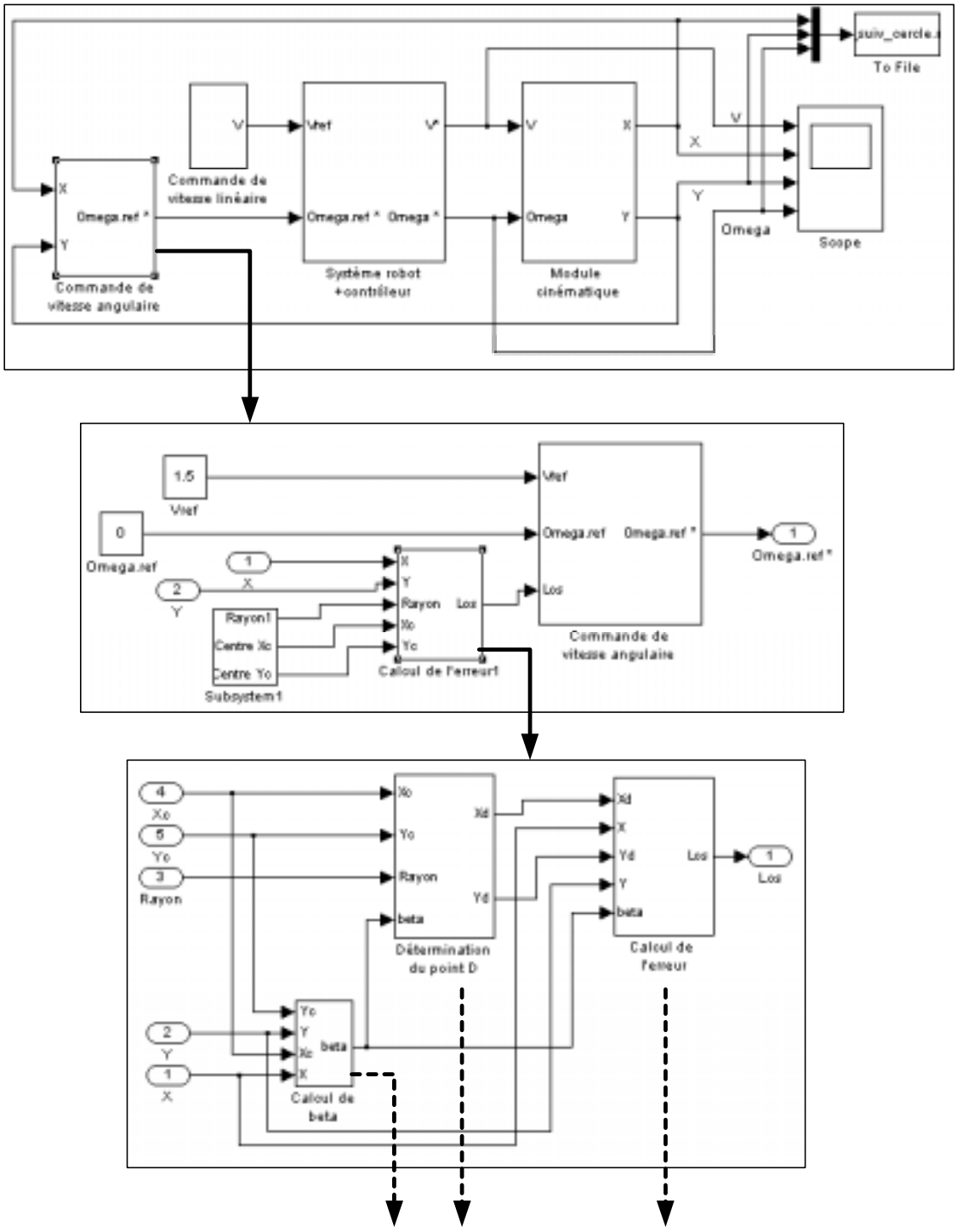


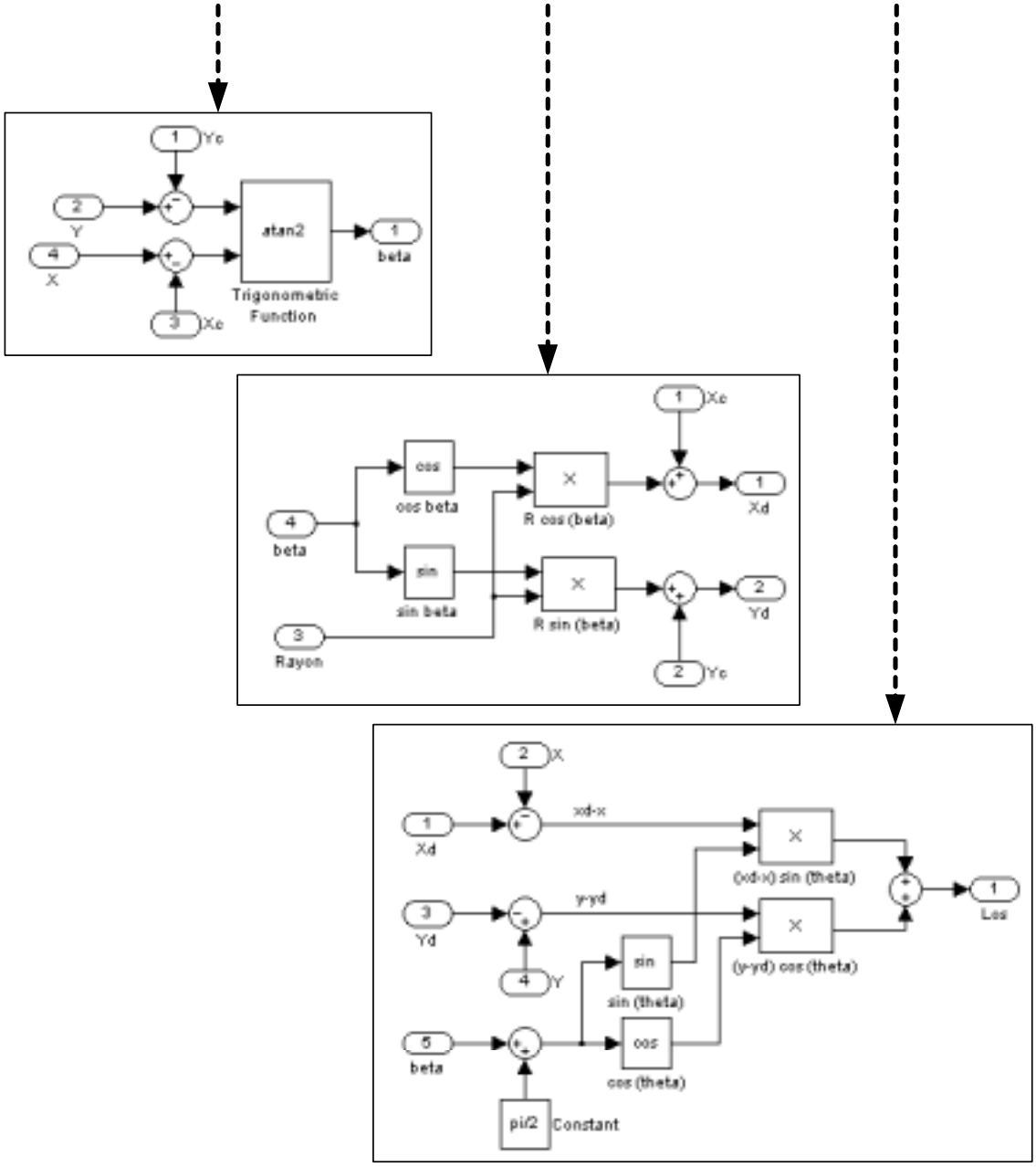

R/2
w_left *
w_right *
Product
Product1
V *
Omega *
R/l
Ctrl vit gauche
Ctrl vit droite
Kvff
Integrator1
PID
Transfer Fcn
Integrator
V_ref
Omega_ref
R
Product2
0.4/2
l/2

1.2. Programme Simulink de suivi de ligne



1.3. Programme Simulink du suivi de cercle





Pour les blocs de la structure de base qui ne sont pas représentés, il faut se référer aux deux paragraphes précédents.

2. ANNEXE B : CODE SOURCE DES PROGRAMMES

Les programmes de suivi de ligne et de suivi de cercle sont donnés ici dans leur entièreté.

2.1. Code source du suivi de ligne

```
#include <devctl.h>
#include "acs_qnx.macro"
#include <iostream>
#include <fstream>
#include <iomanip.h>
#include <fcntl.h>
#include <string>
#include <time.h>

#define BILLION 1000000000

int main ()
{
  int fd;
  long int value;
  long int val;
  long int err = 0;

  int kp  = 35;
  int kd  = 15;
  int ki  = 0;
  double kvff = 1/5.3;
  int k1=1;
  int k2=2;

  int kp_pmd;
  int kd_pmd;
  int ki_pmd;
  double kvff_pmd;
  int k1_pmd;
  int k2_pmd;

  int sample = 400;         //valeur en usec
  double cntsperrad = 11800/(2*3.14159); //nb de tics par radian

  double lg = 0.4;          //distance entre les 2 roues motrices
  double lg_pmd;
  double R = 0.0525;        //rayon des roues motrices
  double R_pmd;

  float speed = 1.5;        //vitesse linéaire en m/s
  float speed_pmd;
  float omega = 0;          //vitesse angulaire en rad/s
  float omega_pmd;
  float omega_etoile;
  float acc = 1.5;          // accélération en m/s^2

  double wleft, wright, aleft, aright;
  double wleft_pmd, wright_pmd, aleft_pmd, aright_pmd;

  int enc_left[2000];
```

```
int enc_right[2000];
float vit_left[2000];
float vit_right[2000];
float vit_lin[2000];
float vit_ang[2000];
float ometoi[2000];

int pos_des = 5;         //valeur en metres
int pos_des_tics;

double temps[2000];
double tempsdeux[2000];
double sec[2000];
double nsec[2000];
double bissec;
double bisnsec;
struct timespec last,first,second;
double avant, apres;

int i=1;
int k=0;
int cpt=0;
double delta = 0;

double dist=0;
long double theta=0;
double delta_theta;
double x=0;
long double y=0.5;
double y_pmd;
double y_diff[2000];
double pos_ang[2000];
double pos_x[2000];
long double pos_y[2000];

//conditions initiales
enc_left[0]=0;
enc_right[0]=0;
vit_lin[0]=speed;
vit_ang[0]=omega;
pos_y[0]=y;
temps[0]=0.04;

//facteurs de conversion
double fgain = 0.916;
double flong=35772;           //nb de tics/m
double fang=cntsperrad*sample*1e-6;   //0.75 nb de tics/sample
double flin=35772*1878*sample*1e-6;   //nb de tics*tics/sample

aleft=acc;
aright=acc;

/*conversion des gains*/
kp_pmd=kp*fgain;
kd_pmd=kd*fgain;
ki_pmd=ki*fgain;
kvff_pmd=kvff*fgain;
k1_pmd=k1*fgain;
```

```
      k2_pmd=k2*fgain;

      /*conversion des longueurs*/
      R_pmd=R * flong;
      lg_pmd=lg * flong;
      y_pmd=y * flong;
      pos_des_tics = pos_des * flong;

      /* conversion de la vitesse du robot en rad/s */
      wleft = (speed-(omega*lg)/2)/R;
      wright = (speed+(omega*lg)/2)/R;

      /* conversion des vitesses en counts/sample */
      wleft_pmd = wleft*fang;
      wright_pmd = wright*fang;

      /* conversion en format de la carte*/
      wleft_pmd = (long)((((long)fabs(wleft_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wleft_pmd) - (long)fabs(wleft_pmd)) *
65536) & 0x0000FFFF));
      wright_pmd= (long)((((long)fabs(wright_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wright_pmd) - (long)fabs(wright_pmd)) *
65536) & 0x0000FFFF));

      /* conversion de l'accélération des roues en counts/sample^2*/
      aleft=(aleft/R)*cntsperrad*sample*sample*1e-12;
      aright=(aright/R)*cntsperrad*sample*sample*1e-12;

      /* conversion en format de la carte de l'accélération */
      aleft_pmd = (long)((((long)fabs(aleft) << 16) & 0xFFFF0000) +
((long)((fabs(aleft) - (long)fabs(aleft)) * 65536) & 0x0000FFFF));
      aright_pmd= (long)((((long)fabs(aright) << 16) & 0xFFFF0000) +
((long)((fabs(aright) - (long)fabs(aright)) * 65536) & 0x0000FFFF));

      ofstream fichier;

      fichier.open ("suivi_ligne.m");

      if ((fd = open("/dev/ctrl", O_RDONLY)) < 0)  // FIXME:
HARDCODE VALUE
          {
          cout<<"Error when trying to open /dev/ctrl in
Controller_acs"<<endl;
          return -1;
          }

      value =0;
      err += devctl(fd, RESET, &value, sizeof(value), NULL);
      value = sample/100;
      err += devctl(fd, SET_SMPL_TIME, &value, sizeof(value), NULL);

      /*  Initialisation de la carte en boucle fermee */

      //axe 1
      value = 0;
      err += devctl(fd, SET_1, &value, sizeof(value), NULL);
      value = 0;
      err += devctl(fd, SET_ACTL_POS, &value, sizeof(value), NULL);
```

```
      err += devctl(fd, MTR_ON, &value, sizeof(value), NULL);
      err += devctl(fd, SET_PRFL_VEL, &value, sizeof(value), NULL);
//profile de trajectoire ds le mode velocity contouring
      err += devctl(fd, SET_KP, &kp_pmd, sizeof(kp_pmd), NULL);
      err += devctl(fd, SET_KD, &kd_pmd, sizeof(kd_pmd), NULL);
      err += devctl(fd, SET_KI, &ki_pmd, sizeof(ki_pmd), NULL);
      err += devctl(fd, SET_KVFF, &kvff_pmd, sizeof(kvff_pmd),NULL);
      err += devctl(fd, SET_AUTO_UPDATE_OFF, &value,
sizeof(value),NULL); //pour que l"axe reste dans le mode de
trajectoire setté
      value = 11800;       //TODO Hardcode value
      err += devctl(fd, SET_POS_ERR, &value, sizeof(value), NULL);
//valeur erreur max
      err += devctl(fd, SET_AUTO_STOP_ON, &value, sizeof(value),
NULL); //pour arreter moteur si erreur>erreur max
      err += devctl(fd, LMTS_OFF, &value, sizeof(value), NULL);
      err += devctl(fd, CLR_STATUS, &value, sizeof(value), NULL);
//clear tous les bits evenements
      err += devctl(fd, UPDATE, &value, sizeof(value), NULL);

      //axe 2
      err += devctl(fd, SET_2, &value, sizeof(value), NULL);
      value = 0;
      err += devctl(fd, SET_ACTL_POS, &value, sizeof(value), NULL);
      err += devctl(fd, MTR_ON, &value, sizeof(value), NULL);
      err += devctl(fd, SET_PRFL_VEL, &value, sizeof(value), NULL);
      err += devctl(fd, SET_KP, &kp_pmd, sizeof(kp_pmd), NULL);
      err += devctl(fd, SET_KD, &kd_pmd, sizeof(kd_pmd), NULL);
      err += devctl(fd, SET_KI, &ki_pmd, sizeof(ki_pmd), NULL);
      err += devctl(fd, SET_KVFF, &kvff_pmd, sizeof(kvff_pmd),NULL);
      err += devctl(fd, SET_AUTO_UPDATE_OFF, &value, sizeof(value),
NULL);
      value = 11800;       //TODO Hardcode value
      err += devctl(fd, SET_POS_ERR, &value, sizeof(value), NULL);
      err += devctl(fd, SET_AUTO_STOP_ON, &value, sizeof(value),
NULL);
      err += devctl(fd, LMTS_OFF, &value, sizeof(value), NULL);
      err += devctl(fd, CLR_STATUS, &value, sizeof(value), NULL);
      err += devctl(fd, UPDATE, &value, sizeof(value), NULL);

      /*  Fin de l'initialisation de la carte en boucle fermee */

      fichier<<"res = ["<<endl;

      /* Envoie de commandes dans le mode velocity contouring*/

      //envoie des commandes au premier axe
      value = 0;
      err += devctl(fd, SET_1, &value, sizeof(value), NULL);
      value = (long int) -wleft_pmd;
      err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
      value = (long int) -aleft_pmd;
      err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);

      //envoie des commandes au deuxieme axe
      value = 0;
      err += devctl(fd, SET_2, &value, sizeof(value), NULL);
      value = (long int) wright_pmd;
```

```
      err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
      value = (long int) aright_pmd;
      err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);

      value = 3;
      err += devctl(fd, MULTI_UPDATE, &value, sizeof(value), NULL);
//update des commandes

      do
         {
           //lecture du temps
           clock_gettime(CLOCK_REALTIME, &first);
           sec[i]=first.tv_sec;
           nsec[i]=first.tv_nsec;

           //calcul du temps entre deux échantillons
           temps[i]= (sec[i] - sec[i-1]) +
             (double)(nsec[i] - nsec[i-1])/(double)BILLION;

           //lecture des encodeurs
           err += devctl(fd, SET_1, &value, sizeof(value), NULL);
           err += devctl(fd, GET_ACTL_POS, &value, sizeof(value),
NULL);
           enc_left[i]= -value;

           err += devctl(fd, SET_2, &value, sizeof(value), NULL);
           err += devctl(fd, GET_ACTL_POS, &value, sizeof(value),
NULL);
           enc_right[i]= value;

           //calcul des vitesses linéaire et angulaire
           vit_left[i]=(enc_left[i]-enc_left[i-1])*2*3.14159*R/
(11800*0.04);
           vit_right[i]=(enc_right[i]-enc_right[i-
1])*2*3.14159*R/(11800*0.04);
           vit_lin[i]=(vit_left[i]+vit_right[i])/2;
           vit_ang[i]=(-vit_left[i]+vit_right[i])/lg;

           //angle theta
           theta += vit_ang[i]* 0.04;
    /*     if ((theta>3.14159/2) || (theta<-3.14159/2)) //utile ou
non?
             {
                theta = theta - 2*3.14159*int(theta/3.14159);
             }*/

           pos_ang[i]=vit_ang[i]* temps[i];

           //position
           y_diff[i] = speed * sin(theta) * 0.04;
           y += y_diff[i];
           pos_y[i]=y;

           //commande
           omega_etoile= -1 * y /speed - 2*y_diff[i] /(speed* 0.04);
           ometoi[i]=omega_etoile;

           /* calcul des vitesses des roues gauche et droite*/
```

```
            wleft = (speed-(omega_etoile*lg)/2)/R;
            wright = (speed+(omega_etoile*lg)/2)/R;

            /*conversion des vitesses en counts par sample*/
            wleft_pmd=wleft * fang;
            wright_pmd=wright * fang;

            /* conversion des vitesses en format de la carte*/
            wleft_pmd = (long)((((long)fabs(wleft_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wleft_pmd) - (long)fabs(wleft_pmd)) *
65536) & 0x0000FFFF));
            wright_pmd= (long)((((long)fabs(wright_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wright_pmd) - (long)fabs(wright_pmd)) *
65536) & 0x0000FFFF));

            /*envoie des commandes dans le mode velocity contouring*/

            //envoie des commandes au premier axe
            value = 0;
            err += devctl(fd, SET_1, &value, sizeof(value), NULL);
            value = (long int) -wleft_pmd;
            err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
            value = (long int) -aleft_pmd;
            err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);

            //envoie des commandes au deuxieme axe
            value = 0;
            err += devctl(fd, SET_2, &value, sizeof(value), NULL);
            value = (long int) wright_pmd;
            err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
            value = (long int) aright_pmd;
            err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);

            value = 3;
            err += devctl(fd, MULTI_UPDATE, &value, sizeof(value),
NULL);

            while (delta<0.04)   //boucle d'attente
              {
                 //lecture du temps
                 clock_gettime(CLOCK_REALTIME, &second);
                 bissec=second.tv_sec;
                 bisnsec=second.tv_nsec;
                 delta=(bissec - sec[i]) + (double)(bisnsec -
nsec[i])/(double)BILLION;
              }

            delta=0;

            i++;
            cpt++;
          }
        while (enc_right[i-1]<pos_des_tics);

        // remise de la vitesse à 0
        err += devctl(fd, SET_1, &value, sizeof(value), NULL);
        value =  0x0;
        err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
```

```
        err += devctl(fd, SET_2, &value, sizeof(value), NULL);
        value = 0x0;
        err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);

        value = 3;
        err += devctl(fd, MULTI_UPDATE, &value, sizeof(value), NULL);

        //calcul du temps et écriture des donnees (temps, enc_left et
enc_right) dans un fichier
        for (int k=2;k<cpt;k++)
          {

            fichier<<enc_left[k]<<" "<<enc_right[k]<<" "<<temps[k]<<"
"<<pos_y[k]<<" "<<vit_ang[k]<<" "<<pos_ang[k]<<" ;"<<endl;
          }
        return 0;
    }
```

2.2. Code source du suivi de cercle

```
    #include <devctl.h>
    #include "acs_qnx.macro"
    #include <iostream>
    #include <fstream>
    #include <iomanip.h>
    #include <fcntl.h>
    #include <string>
    #include <time.h>
    #include <cmath>

    #define BILLION 1000000000

    int main ()
    {
      int fd;
      long int value;
      long int val;
      long int err = 0;

      int kp  = 35;
      int kd  = 15;
      int ki  = 0;
      double kvff = 1/5.3;
      int k1=1;
      int k2=2;

      int kp_pmd;
      int kd_pmd;
      int ki_pmd;
      double kvff_pmd;
      int k1_pmd;
      int k2_pmd;

      int sample = 400;        //valeur en usec
      double cntsperrad = 11800/(2*3.14159); //nb de tics par radian
```

```
double lg = 0.4;          //distance entre les 2 roues motrices
double lg_pmd;
double R = 0.0525;        //rayon des roues motrices
double R_pmd;

float speed = 0.5;        //vitesse linéaire en m/s
float speed_pmd;
float omega = 0.5;        //vitesse angulaire en rad/s
float omega_pmd;
float omega_etoile;
float acc = 1;            // accélération en m/s^2

long double wleft, wright, aleft, aright;
long double wleft_pmd, wright_pmd, aleft_pmd, aright_pmd;

int enc_left[2000];
int enc_right[2000];
float vit_left[2000];
float vit_right[2000];
float vit_lin[2000];
float vit_ang[2000];
float ometoi[2000];

int pos_des = 10;         //valeur en metres
int pos_des_tics;

double temps[2000];
double tempsdeux[2000];
double sec[2000];
double nsec[2000];
double bissec;
double bisnsec;
struct timespec last,first,second;
double avant, apres;

int i=1;
int k=1;
int cpt=0;
double delta = 0;

double dist=0;
long double theta;
double delta_theta;
long double beta;
long double theta_ref;
long double xd;
long double yd;
long double xc=0;
long double yc=0;
long double y_pmd;
long double y_diff[2000];
long double pos_ang[2000];
long double pos_x[2000];
long double pos_y[2000];

long double erreur[2000];
double Rc=1;
```

```
      //facteurs de conversion
      double fgain = 0.916;
      double flong=35772;           //nb de tics/m
      double fang=cntsperrad*sample*1e-6;   //0.75 nb de tics/sample
      double flin=35772*1878*sample*1e-6;   //nb de tics*tics/sample

      //initialisation
      pos_x[0]=1;
      pos_y[0]=0;
      theta=3.14159/2;
      vit_lin[0]=speed;
      vit_ang[0]=omega;
      temps[0]=0.04;
      enc_left[0]=0;
      enc_right[0]=0;
      erreur[0]=- (sqrt(pos_x[0]*pos_x[0]+pos_y[0]*pos_x[0])-Rc);

      aleft=acc;
      aright=acc;

      /*conversion des gains*/
      kp_pmd=kp*fgain;
      kd_pmd=kd*fgain;
      ki_pmd=ki*fgain;
      kvff_pmd=kvff*fgain;
      k1_pmd=k1*fgain;
      k2_pmd=k2*fgain;

      /*conversion des longueurs*/
      pos_des_tics = pos_des * flong; //35772 tics/m

      /* conversion de la vitesse du robot en rad/s */
      wleft = (speed-(omega*lg)/2)/R;
      wright = (speed+(omega*lg)/2)/R;

      /* conversion des vitesses en counts/sample */
      wleft_pmd = wleft*fang;
      wright_pmd = wright*fang;

      /* conversion en format de la carte*/
      wleft_pmd = (long)((((long)fabs(wleft_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wleft_pmd) - (long)fabs(wleft_pmd)) *
65536) & 0x0000FFFF));
      wright_pmd= (long)((((long)fabs(wright_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wright_pmd) - (long)fabs(wright_pmd)) *
65536) & 0x0000FFFF));

      /*conversion de l'accélération des roues en counts/sample^2 */
      aleft=(aleft/R)*cntsperrad*sample*sample*1e-12;
      aright=(aright/R)*cntsperrad*sample*sample*1e-12;

      /* conversion en format de la carte de l'accélération */
      aleft_pmd = (long)((((long)fabs(aleft) << 16) & 0xFFFF0000) +
((long)((fabs(aleft) - (long)fabs(aleft)) * 65536) & 0x0000FFFF));
      aright_pmd= (long)((((long)fabs(aright) << 16) & 0xFFFF0000) +
((long)((fabs(aright) - (long)fabs(aright)) * 65536) & 0x0000FFFF));
```

```
      ofstream fichier;

      fichier.open ("suivi_cercle.m");

      if ((fd = open("/dev/ctrl", O_RDONLY)) < 0)  // FIXME:
HARDCODE VALUE
          {
          cout<<"Error when trying to open /dev/ctrl in
Controller_acs"<<endl;
          return -1;
          }

      value =0;
      err += devctl(fd, RESET, &value, sizeof(value), NULL);
      value = sample/100;
      err += devctl(fd, SET_SMPL_TIME, &value, sizeof(value), NULL);

      /*  Initialisation de la carte en boucle fermee */

      //axe 1
      value = 0;
      err += devctl(fd, SET_1, &value, sizeof(value), NULL);
      value = 0;
      err += devctl(fd, SET_ACTL_POS, &value, sizeof(value), NULL);
      err += devctl(fd, MTR_ON, &value, sizeof(value), NULL);
      err += devctl(fd, SET_PRFL_VEL, &value, sizeof(value), NULL);
//profile de trajectoire ds le mode velocity contouring
      err += devctl(fd, SET_KP, &kp_pmd, sizeof(kp_pmd), NULL);
      err += devctl(fd, SET_KD, &kd_pmd, sizeof(kd_pmd), NULL);
      err += devctl(fd, SET_KI, &ki_pmd, sizeof(ki_pmd), NULL);
      err += devctl(fd, SET_KVFF, &kvff_pmd, sizeof(kvff_pmd),
NULL);
      err += devctl(fd, SET_AUTO_UPDATE_OFF, &value,  sizeof(value),
NULL); //pour que l"axe reste dans le mode de trajectoire setté
      value = 11800;       //TODO Hardcode value
      err += devctl(fd, SET_POS_ERR, &value, sizeof(value), NULL);
//valeur erreur max
      err += devctl(fd, SET_AUTO_STOP_ON, &value, sizeof(value),
NULL); //pour arreter moteur si erreur>erreur max
      err += devctl(fd, LMTS_OFF, &value, sizeof(value), NULL);
      err += devctl(fd, CLR_STATUS, &value, sizeof(value), NULL);
//clear tous les bits evenements
      err += devctl(fd, UPDATE, &value, sizeof(value), NULL);

      //axe 2
      err += devctl(fd, SET_2, &value, sizeof(value), NULL);
      value = 0;
      err += devctl(fd, SET_ACTL_POS, &value, sizeof(value), NULL);
      err += devctl(fd, MTR_ON, &value, sizeof(value), NULL);
      err += devctl(fd, SET_PRFL_VEL, &value, sizeof(value), NULL);
      err += devctl(fd, SET_KP, &kp_pmd, sizeof(kp_pmd), NULL);
      err += devctl(fd, SET_KD, &kd_pmd, sizeof(kd_pmd), NULL);
      err += devctl(fd, SET_KI, &ki_pmd, sizeof(ki_pmd), NULL);
      err += devctl(fd, SET_KVFF, &kvff_pmd, sizeof(kvff_pmd),
NULL);
      err += devctl(fd, SET_AUTO_UPDATE_OFF, &value, sizeof(value),
NULL);
      value = 11800;       //TODO Hardcode value
```

```
      err += devctl(fd, SET_POS_ERR, &value, sizeof(value), NULL);
      err += devctl(fd, SET_AUTO_STOP_ON, &value, sizeof(value),
NULL);
      err += devctl(fd, LMTS_OFF, &value, sizeof(value), NULL);
      err += devctl(fd, CLR_STATUS, &value, sizeof(value), NULL);
      err += devctl(fd, UPDATE, &value, sizeof(value), NULL);

      /*  Fin de l'initialisation de la carte en boucle fermee */

      fichier<<"res = ["<<endl;

      /* Envoie de commandes dans le mode de velocity contouring*/

      //envoie des commandes au premier axe
      value = 0;
      err += devctl(fd, SET_1, &value, sizeof(value), NULL);
      value = (long int) -wleft_pmd;
      err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
      value = (long int) -aleft_pmd;
      err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);

      //envoie des commandes au deuxieme axe
      value = 0;
      err += devctl(fd, SET_2, &value, sizeof(value), NULL);
      value = (long int) wright_pmd;
      err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
      value = (long int) aright_pmd;
      err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);

      value = 3;
      err += devctl(fd, MULTI_UPDATE, &value, sizeof(value), NULL);

      do
         {
           //lecture du temps
           clock_gettime(CLOCK_REALTIME, &first);
           sec[i]=first.tv_sec;
           nsec[i]=first.tv_nsec;

           //calcul du temps entre deux échantillons
           temps[i]= (sec[i] - sec[i-1]) +
             (double)(nsec[i] - nsec[i-1])/(double)BILLION;

           //lecture des encodeurs
           err += devctl(fd, SET_1, &value, sizeof(value), NULL);
           err += devctl(fd, GET_ACTL_POS, &value, sizeof(value),
NULL);
           enc_left[i]= -value;

           err += devctl(fd, SET_2, &value, sizeof(value), NULL);
           err += devctl(fd, GET_ACTL_POS, &value, sizeof(value),
NULL);
           enc_right[i]= value;

           //calcul des vitesses linéaire et angulaire
           vit_left[i]=(enc_left[i]-enc_left[i-1])*2*3.14159*R/
(11800*0.04);
```

```
            vit_right[i]=(enc_right[i]-enc_right[i-1])*2*3.14159*R/
(11800*0.04);
            vit_lin[i]=(vit_left[i]+vit_right[i])/2;
            vit_ang[i]=(-vit_left[i]+vit_right[i])/lg;

            //angle theta
            theta += vit_ang[i]* 0.04;
        /*  cout<<"theta avant "<<theta<<endl;          //utile?
            if (theta>3.14159)
              theta = theta - 3.14159*int(theta/3.14159);
            if (theta<-3.14159)
              theta = theta + 3.14159*int(theta/3.14159);*/

            pos_ang[i]=vit_ang[i]* temps[i];

            //position
            pos_x[i]=pos_x[i-1] + speed * cos(theta) * 0.04;
            pos_y[i]=pos_y[i-1] + speed * sin(theta) * 0.04;

            //calcul de l'erreur latérale

            beta=atan2((pos_y[i]-yc),(pos_x[i]-xc));
            theta_ref=beta + 3.14159/2;

            xd=Rc*cos(beta)+xc;
            yd=Rc*sin(beta)+yc;
            erreur[i]=(xd-pos_x[i])*sin(theta_ref)+(pos_y[i]-yd)*
cos(theta_ref);

            //commande
            omega_etoile= omega-1*erreur[i]/speed -2*(erreur[i]-
erreur[i-1])/(speed * 0.04);

            /* calcul des vitesses des roues gauche et droite*/
            wleft = (speed-(omega_etoile*lg)/2)/R;
            wright = (speed+(omega_etoile*lg)/2)/R;

            /*conversion en counts par sample*/
            wleft_pmd=wleft * fang;
            wright_pmd=wright * fang;

            /* conversion en format de la carte*/
            wleft_pmd = (long)((((long)fabs(wleft_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wleft_pmd) - (long)fabs(wleft_pmd)) *
65536) & 0x0000FFFF));
            wright_pmd= (long)((((long)fabs(wright_pmd) << 16) &
0xFFFF0000) + ((long)((fabs(wright_pmd) - (long)fabs(wright_pmd)) *
65536) & 0x0000FFFF));

            /*envoi de commandes dans le mode velocity contouring*/

            //envoie des commandes au premier axe
            value = 0;
            err += devctl(fd, SET_1, &value, sizeof(value), NULL);
            value = (long int) -wleft_pmd;
            err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
            value = (long int) -aleft_pmd;
            err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);
```

```
            //envoie des commandes au deuxieme axe
            value = 0;
            err += devctl(fd, SET_2, &value, sizeof(value), NULL);
            value = (long int) wright_pmd;
            err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);
            value = (long int) aright_pmd;
            err += devctl(fd, SET_ACC, &value, sizeof(value), NULL);

            value = 3;
            err += devctl(fd, MULTI_UPDATE, &value, sizeof(value),
NULL);

            while (delta<0.04)   //boucle d'attente
              {
                 //lecture du temps
                 clock_gettime(CLOCK_REALTIME, &second);
                 bissec=second.tv_sec;
                 bisnsec=second.tv_nsec;
                 delta=(bissec - sec[i]) + (double)(bisnsec -
nsec[i])/(double)BILLION;
              }

            delta=0;

            i++;
            cpt++;
          }
        while (enc_right[i-1]<pos_des_tics);

        // remise de la vitesse à 0
        err += devctl(fd, SET_1, &value, sizeof(value), NULL);
        value =  0x0;
        err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);

        err += devctl(fd, SET_2, &value, sizeof(value), NULL);
        value = 0x0;
        err += devctl(fd, SET_VEL, &value, sizeof(value), NULL);

        value = 3;
        err += devctl(fd, MULTI_UPDATE, &value, sizeof(value), NULL);

        //calcul du temps et écriture des donnees (temps, enc_left et
enc_right) dans un fichier
        for (int k=2;k<cpt;k++)
          {
            fichier<<enc_left[k]<<" "<<enc_right[k]<<" "<<temps[k]<<"
"<<pos_x[k]<<" "<<pos_y[k]<<" "<<vit_ang[k]<<" "<<erreur[k]<<"
;"<<endl;
          }
        return 0;
    }
```

3. ANNEXE C

Cette troisième annexe est disponible sur le CD. Il contient :

- le code source des programmes C++
- les simulations effectuées sur Simulink (Matlab 6.1)
- les fiches techniques des différentes composantes jugées importantes du robot
- le présent rapport