Université de Liège - Faculté des Sciences Appliquées
Institut Montefiore

Travail de fin d'études

Locomotion d'un robot mobile

Stéphane Lens, 3ELIN

Mai 2008

Mémoire présenté en vue de l'obtention du grade
d'Ingénieur Civil Informaticien

Année académique 2007-2008

Résumé

Dans le domaine de la robotique mobile, l'étude de la locomotion possède une place prépondérante. De nombreuses approches et solutions peuvent être envisagées et il convient d'apporter un soin particulier quant à leur sélection afin de garantir les performances du système final. Ce travail de fin d'études a pour objectif la conception complète du système de locomotion d'un robot mobile participant à un concours de robotique. La réalisation des cartes électroniques de commande pour les moteurs, ainsi que la synthèse des lois de contrôle sont étudiées en détails. Ces lois seront principalement divisées en deux phases : d'une part, la génération d'une trajectoire réalisable, et d'autre part, une régulation sur cette trajectoire grâce, entre autres, à un contrôle local de la vitesse des roues motrices. Ce travail se termine, enfin, par une implémentation pratique du système étudié et par une évaluation de ses performances.

Remerciements

C'est avec un réel plaisir que je tiens à remercier tous ceux, qui d'une façon ou d'une autre, ont contribué à la réalisation de ce travail de fin d'études.

Je remercie, tout d'abord, Monsieur le Professeur Bernard Boigelot pour avoir permis à de nombreux étudiants de prendre part à un concours de robotique aussi motivant que passionnant et pour m'avoir proposé de réaliser mon travail dans ce cadre attrayant. Je le remercie, également, pour ses nombreux conseils et son aide avisée.

Je remercie, ensuite, toutes les personnes ayant pris part au projet Eurobot 2008, en particulier Sébastien Piérard, Vincent Pierlot et Laurent Poirrier, sans qui mon travail aurait trouvé une application nettement moins intéressante.

Enfin, je n'oublie évidemment pas tous ceux qui, par leur soutien, m'ont aidé dans l'élaboration de ce travail, mes amis, mon frère et mes parents.

Table des matières

Table des figures

Chapitre 1

Introduction

Sommaire

1.1 Objectifs

Dans le domaine de la robotique mobile, l'étude de la locomotion a, bien entendu, une place prépondérante. De nombreuses approches et solutions peuvent être envisagées et il convient d'apporter un soin particulier quant à leur sélection afin de garantir les performances du système final.

Ce travail s'inscrit dans le cadre d'une participation à un concours de robotique mobile et a pour objectif la conception complète du mécanisme de locomotion du robot participant. La réalisation se devra d'être la plus générale possible afin de pouvoir être appliquée à d'autres robots mobiles se déplaçant sur une surface plane.

L'objectif final sera la possibilité de déplacer le robot d'une configuration A à une configuration B tout en restant dans un périmètre fixé, et ceci, de la manière la plus rapide et la plus précise possible.

On entend par conception complète de la locomotion, la réalisation de la partie électronique, de la partie contrôle de bas niveau (e.g. contrôle local d'un moteur) ainsi que de la partie contrôle de haut niveau (e.g. suivi de trajectoire), toutes trois nécessaires au déplacement du robot.

Dans un souci de modularité, on aura pour objectif secondaire de pouvoir utiliser l'électronique développée comme contrôleur générique de moteur.

1.2 Résumé du travail

La première étape du travail consiste en une étude préliminaire des différents types de robots mobiles pouvant être utilisés, ainsi que des différentes contraintes extérieures qu'impose le cadre du concours dans lequel ce travail s'inscrit. En effet, un certain nombre de contraintes mécaniques, électroniques et informatiques doivent être prises en compte.

Le robot utilisé est un robot à roues de type unicycle et sa locomotion est assurée par deux moteurs-réducteurs à courant continu sur lesquels les roues sont directement fixées. La logique principale du robot est assurée par un ordinateur embarqué faisant tourner un *OS* temps réel. Le contrôle direct des différents périphériques du robot (comme les actuateurs) est assuré par des cartes électroniques séparées, reliées entre elles par un bus I^2C. Une carte électronique spécifique assure l'interface entre le bus et l'ordinateur embarqué.

Vient ensuite l'étude du contrôle, après une formalisation claire de nos attentes, on cherche une méthode optimale sous la forme de lois de contrôle pour les réaliser. L'objectif étant d'amener le robot d'une configuration A à une configuration B de la manière la plus rapide possible tout en respectant le roulement sans glissement des roues et tout en restant dans un certain périmètre fixé.

Plusieurs solutions sont envisagées : une suite de rotations et de lignes droites, de l'asservissement en position, du suivi de chemin ou encore du suivi de trajectoire avec profil de vitesse.

La dernière solution est celle qui permet de réaliser au mieux nos objectifs et est étudiée en détail. Elle consiste globalement en un asservissement à deux niveaux :

- un asservissement local *PID* en vitesse de chaque moteur pouvant être réalisé par une carte électronique.

- une régulation sur une trajectoire pré-calculée supposant le contrôle parfait des moteurs pouvant être réalisée sur l'ordinateur embarqué.

Le calcul des trajectoires et des profils de vitesse associés est étudié en détail, il fait appel à la génération d'ensembles de lignes droites, d'arcs de cercle et de clothoïdes et permet d'obtenir des trajectoires de durée minimale, réalisables pour notre robot. Nous définissons, ensuite, une loi de

contrôle permettant de réguler le robot à partir d'une trajectoire réalisable calculée. Pour ce faire, nous essayons de minimiser l'erreur en orientation et l'erreur en distance du robot par rapport à la trajectoire.

L'étape suivante consiste en la réalisation d'une carte électronique de contrôle pour un moteur à courant continu. Celle-ci passe par le design d'un étage de puissance constitué principalement d'un pont en H et par la sélection de ses composants. Le choix du mode de pilotage du pont en H est discuté, le mode *Locked Anti-phase* est sélectionné car c'est le mode qui permettra de contrôler au mieux les phases d'accélération et de décélération du moteur.

Nous faisons, ensuite, le design de la partie logique avec le choix d'un microcontrôleur et des différents capteurs nécessaires au fonctionnement de la carte électronique. La carte doit, par exemple, être en mesure de détecter une situation dangereuse pour le moteur par une mesure du courant consommé.

Une fois la carte électronique réalisée et testée, nous en venons à l'étape d'implémentation du contrôle. Tout d'abord, le contrôle de bas niveau *PID* est codé dans le microcontrôleur et permet de contrôler localement en vitesse un moteur. Ensuite, un programme de haut niveau est établi en *Java* pour aider au paramétrage de la boucle *PID* et permet d'évaluer les performances de notre contrôleur.

La dernière étape, est la mise en oeuvre du suivi de trajectoire sur le robot. Le suivi est géré par l'ordinateur embarqué, et implémente directement la solution étudiée. Globalement, lorsque nous demandons au robot d'aller dans une certaine configuration, il calcule une trajectoire et un profil de vitesse entre sa configuration courante et la configuration demandée. Il impose, ensuite, le profil de vitesse à ses roues, tout en effectuant la régulation nécessaire.

Enfin, nous discutons de l'application de notre solution au concours Eurobot. Nous évaluons les performances du système et nous donnons une liste de pistes à suivre pour des améliorations futures.

Chapitre 2

Robots mobiles

Sommaire

2.1 Définition

Un robot mobile est une machine automatique capable de se mouvoir dans un environnement donné.

On regroupe sous cette appellation tous les robots autonomes (i.e. non télécommandés) capables de se déplacer, par opposition aux robots attachés à un point fixe, comme les robots manipulateurs en industrie.
Il existe plusieurs types de robots mobiles et ceux-ci sont, en général, classifiés selon leur type de locomotion (i.e. le milieu dans lequel ils évoluent ainsi que leur mode de propulsion).
Les robots mobiles évolueront donc sur terre, dans les airs ou encore sur ou sous eau. Les robots terrestres sont, par exemple, actionnés par des roues, des chenilles ou encore des pattes.

Pour la résolution de notre problème, nous allons nous concentrer sur l'étude des robots à roues. Ceux-ci sont, en effet, particulièrement adaptés à un environnement plan tel que celui rencontré dans le concours qui nous

intéresse. Ils ont une construction mécanique relativement simple et jouissent d'une certaine contrôlabilité.

2.2 Architecture globale

2.2.1 Classes de robots à roues

Il existe plusieurs classes de robots à roues déterminées, principalement, par la position et le nombre de roues utilisées.

Nous citerons ici les quatre classes principales de robots à roues [Bay08].

Robot unicycle

Un robot de type unicycle est actionné par deux roues indépendantes, il possède éventuellement des roues folles pour assurer sa stabilité. Son centre de rotation est situé sur l'axe reliant les deux roues motrices.

C'est un robot non-holonôme[1], en effet il est impossible de le déplacer dans une direction perpendiculaire aux roues de locomotion.

Sa commande peut être très simple, il est en effet assez facile de le déplacer d'un point à un autre par une suite de rotations simples et de lignes droites.

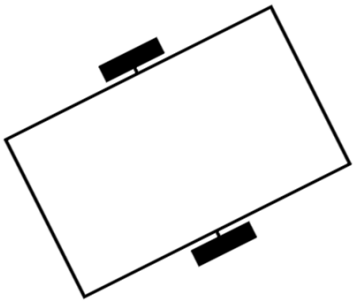

FIG. 2.1 – Robot de type unicycle

Robot tricycle

Un robot de type tricycle est constitué de deux roues fixes placées sur un même axe et d'une roue centrée orientable placée sur l'axe longitudinal. Le mouvement du robot est donné par la vitesse des deux roues fixes et par

[1]Non holonôme signifie que le système comporte une contrainte non intégrable (i.e. le système ne peut pas effectuer certains mouvements).

l'orientation de la roue orientable. Son centre de rotation est situé à l'intersection de l'axe contenant les roues fixes et de l'axe de la roue orientable.

C'est un robot non-holonôme. En effet, il est impossible de le déplacer dans une direction perpendiculaire aux roues fixes. Sa commande est plus compliquée. Il est en général impossible d'effectuer des rotations simples à cause d'un rayon de braquage limité de la roue orientable.

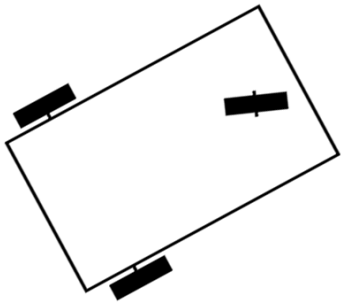

FIG. 2.2 – Robot de type tricycle

Robot voiture

Un robot de type voiture est semblable au tricycle, il est constitué de deux roues fixes placées sur un même axe et de deux roues centrées orientables placées elles aussi sur un même axe.

Le robot de type voiture est cependant plus stable puisqu'il possède un point d'appui supplémentaire.

Toutes les autres propriétés du robot voiture sont identiques au robot tricycle, le deuxième pouvant être ramené au premier en remplaçant les deux roues avant par une seule placée au centre de l'axe, et ceci de manière à laisser le centre de rotation inchangé.

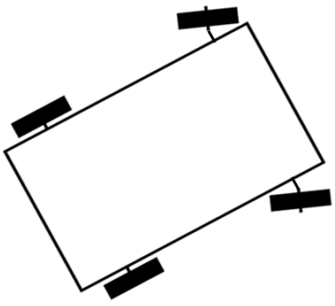

FIG. 2.3 – Robot de type voiture

Robot omnidirectionnel

Un robot omnidirectionnel est un robot qui peut se déplacer librement dans toutes les directions. Il est en général constitué de trois roues décentrées orientables placées en triangle équilatéral.

L'énorme avantage du robot omnidirectionnel est qu'il est holonôme puisqu'il peut se déplacer dans toutes les directions. Mais ceci se fait au dépend d'une complexité mécanique bien plus grande.

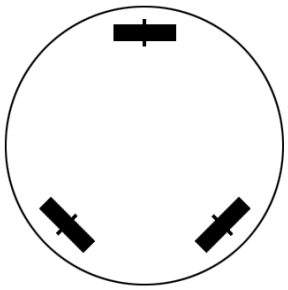

FIG. 2.4 – Robot de type omnidirectionnel

Comparaison des différents types

Nous pouvons observer dans le tableau ci-dessous un récapitulatif des avantages et des inconvénients des différents types de robots à roues.

Robot unicycle	− non-holonôme + stable + rotation sur soi-même + complexité mécanique faible
Robot tricycle	− non-holonôme − peu stable − pas de rotation sur soi-même + complexité mécanique modérée
Robot voiture	− non-holonôme + stable − pas de rotation sur soi-même + complexité mécanique modérée
Robot omnidirectionnel	+ holonôme + stable + rotation sur soi-même − complexité mécanique importante

2.2.2 Robots unicycles

Comme nous l'avons décrit à la sous-section précédente, les robots unicycles sont une classe de robots à roues. Ils représentent un modèle de robot mobile relativement simple et qui est, de ce fait, largement utilisé en pratique.

Nous allons décrire ici plus en détail les propriétés de commande et de localisation particulières du robot unicycle.

Commande

Dans des conditions de roulement sans glissement[2], les vitesses longitudinale et de rotation du robot sont liées aux vitesses de rotation angulaire des deux roues motrices ($\dot{\phi}_r$ et $\dot{\phi}_l$) par les relations suivantes :

Pour la vitesse longitudinale du robot v, on a :

$$v = \frac{v_r + v_l}{2} = \frac{r\left(\dot{\phi}_l - \dot{\phi}_r\right)}{2} \tag{2.1}$$

Pour la vitesse de rotation du robot ω, on a :

$$\omega = \dot{\theta} = \frac{v_r - v_l}{L} = -\frac{r\left(\dot{\phi}_r + \dot{\phi}_l\right)}{L} \tag{2.2}$$

avec r le rayon d'une roue et L la distance entre les deux roues motrices.

Localisation

Dans des conditions de roulement sans glissement, il est possible de déterminer la position du robot à l'instant t connaissant sa position à l'instant t_0.

En effet, on peut écrire :

$$\dot{\theta} = \omega \tag{2.3}$$

$$\dot{x} = v\cos\theta \tag{2.4}$$

$$\dot{y} = v\sin\theta \tag{2.5}$$

[2]On dit qu'un corps est en roulement sans glissement(r.s.g.) sur une surface lorsque la vitesse relative du corps par rapport à cette surface au point de contact est nulle. (i.e. le corps reste en contact avec la surface sans glisser)

On pourra donc déterminer la position du robot à l'instant t par intégration :

$$\theta(t) = \int_0^t \dot{\theta}(\tau)d\tau + \theta_0 = \int_0^t \omega(\tau)d\tau + \theta_0 \tag{2.6}$$

$$x(t) = \int_0^t \dot{x}(\tau)d\tau + x_0 = \int_0^t v(\tau)\cos\theta(\tau)d\tau + x_0 \tag{2.7}$$

$$x(t) = \int_0^t \dot{y}(\tau)d\tau + y_0 = \int_0^t v(\tau)\sin\theta(\tau)d\tau + y_0 \tag{2.8}$$

L'utilisation de capteurs (i.e encodeurs) sur les moteurs ou sur les roues, permet donc de déterminer la position d'un robot unicycle à tout instant, à la condition de connaître sa position initiale. Cette méthode est connue sous l'appellation d'*odométrie*.

En pratique, l'intégrale devra bien sûr être discrétisée. Nous procédons pour ce faire de la manière suivante : nous récupérons la valeur des capteurs à intervalles réguliers et nous faisons l'hypothèse que la vitesse du robot est constante entre deux mesures successives.
Nous veillerons donc à prendre des intervalles de temps suffisamment courts pour que cette hypothèse soit valable, en faisant toutefois attention à les garder suffisamment grands par rapport à la résolution des capteurs.

Si la vitesse du robot est constante, le robot se déplace sur un arc de cercle et on peut mettre à jour la position du robot à chaque nouvelle mesure de la façon suivante :

$$d_r = rp_r,\ d_l = rp_l \tag{2.9}$$

où r est le rayon d'une roue et p_r, p_l les variations (entre t_n et t_{n+1}) de position des roues exprimées en radians.

si $d_r = d_l$:

$$\theta(t_{n+1}) = \theta(t_n) \tag{2.10}$$

$$x(t_{n+1}) = x(t_n) - d_l \sin(\theta(t_n)) \tag{2.11}$$

$$y(t_{n+1}) = y(t_n) + d_l \cos(\theta(t_n)) \tag{2.12}$$

si $d_r \neq d_l$:

$$d\theta = \frac{d_r - d_l}{L} \tag{2.13}$$

$$a_1 = \sin(\theta(t_n))\frac{L(d_l + d_r)}{2(d_r - d_l)} \tag{2.14}$$

$$a_2 = \cos(\theta(t_n))\frac{L(d_l + d_r)}{2(d_r - d_l)} \tag{2.15}$$

et

$$\theta(t_{n+1}) = \theta(t_n) + d\theta \tag{2.16}$$

$$x(t_{n+1}) = x(t_n) - a_1 \sin(d\theta) + a_2 \cos(d\theta) - a_2 \tag{2.17}$$

$$y(t_{n+1}) = y(t_n) + a_1 \cos(d\theta) + a_2 \sin(d\theta) - a_1 \tag{2.18}$$

où L est la distance entre les deux roues motrices.

2.3 Architecture utilisée et contraintes impliquées

Comme, il a été dit dans le chapitre d'introduction, ce travail s'inscrit dans le cadre d'un concours de robotique, certains choix n'étaient donc pas entièrement libres.

Nous allons décrire, ci-après, les points clés de l'architecture globale du robot sur lequel notre étude va porter. Ceux-ci définissent en effet un certain nombre de contraintes dont nous devrons tenir compte dans la suite de notre travail.

2.3.1 Mécanique

Le robot utilisé est de type unicycle et la mécanique de sa plate-forme mobile est relativement simple.
La forme globale du robot est un cylindre d'une trentaine de centimètres de diamètre et la locomotion est assurée par deux roues motrices diamétralement opposées. Les roues motrices ont un diamètre de l'ordre d'une dizaine de centimètres et sont directement fixées sur les réducteurs des moteurs. La stabilité est assurée par une bille placée à l'arrière du robot.

Les moteurs utilisés sont des moteurs à courant continu 15V de 90W et le facteur de réduction est de 1 : 23 avec un rendement de 75%.

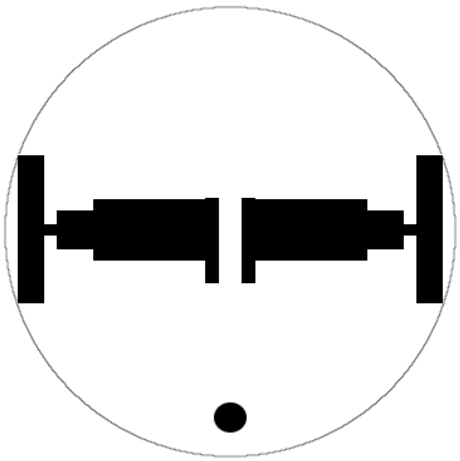

FIG. 2.5 – Vue de la disposition mécanique du robot utilisé

On voit dans le datasheet que le couple nominal du moteur est de 73,2mNm et que sa vitesse nominale est de 6270rpm.

Avec des roues de 10cm de diamètre et un robot d'une quinzaine de kilos, on pourra donc espérer :

$$V_{max} = \frac{6720rpm}{23} * \pi * 10cm = 1,42m/s \tag{2.19}$$

$$C_{roue_{max}} = 75\% * 23 * 73,2mNm = 1,26Nm \tag{2.20}$$

$$F_{roue_{max}} = \frac{C_{roue_{max}}}{r} = \frac{1,26Nm}{5cm} = 25,2N \tag{2.21}$$

$$Acc_{max} = \frac{2F_{roue_{max}}}{m} = \frac{2 * 25,2N}{15kg} = 3,36m/s^2 \tag{2.22}$$

2.3.2 Électronique

L'alimentation du robot est assurée par une batterie de 12V de 5Ah qui alimente aussi bien la carte mère embarquée, l'électronique de contrôle et les actuateurs.

Les cartes électroniques constituant le robot communiquent entres-elles via un bus de périphériques de type I^2C, l'interface entre ce bus et l'ordinateur embarqué est réalisé au moyen d'une carte électronique spécifique (HOST) qui se charge de la traduction $RS232 \leftrightarrow I^2C$.

Le bus de périphériques est constitué de 6 fils :
– deux fils nécessaires à l'I^2C. le bus
– deux fils d'alimentation.
– un fil pour un signal de *reset*.
– un fil pour un signal d'arrêtd'urgence des actuateurs.

La puissance fournie par le bus étant limitée à 1A, les actuateurs devront avoir leurs propres fils d'alimentation.

Une description complète du bus de périphériques ainsi que du protocole spécifique de communication utilisé pourra être trouvée dans [Boi05].

La carte électronique pour les moteurs devra donc gérer le protocole I^2C, ceci avec des temps de réponses relativement courts, afin de ne pas gêner les communications avec les autres cartes. En outre, la carte devra aussi être en mesure de gérer, par un mécanisme sûr, l'arrêt d'urgence des actuateurs. Remarque : l'arrêt d'urgence est une contrainte imposée par les règles du concours auquel le robot final va participer.

La taille des cartes électroniques, la position des connecteurs (bus, programmation, ...) ainsi que la hauteur maximale des composants est aussi imposée pour des raisons pratiques d'organisation du robot.

2.3.3 Informatique

Le robot est architecturé autour d'une carte mère de PC. Un noyau *RTAI*(*Linux temps réel*) est utilisé pour les tâches de haute priorité (comme l'adressage de l'ensemble des cartes électroniques cinquante fois par seconde). Un noyau *Linux* conventionnel, vu comme une tâche de basse priorité du noyau temps réel, est utilisé pour les tâches moins prioritaires.

L'architecture informatique utilisée est dérivée de [Lek03] et on y trouvera des informations plus détaillées.

Nous disposons donc dans le robot d'une grande puissance de calcul. Par contre, celle-ci ne pourra pas être utilisée pour toutes les tâches de contrôle.

Nous devons en effet tenir compte des contraintes suivantes du bus de périphériques :

– L'adressage d'une carte électronique se fait cinquante fois par seconde.
– La réponse à une requête est connue avec un cycle de retard.
– Il est impossible d'adresser deux cartes électroniques au même instant.

Donc, si nous envoyons un ordre au cycle t et que nous envoyons une requête pour observer l'effet de cet ordre au cycle $t + 1$, nous obtenons la réponse en $t + 2$.
Il s'écoule donc au minimum deux cycles (40ms) entre l'envoi d'un ordre et l'observation de l'effet de celui-ci !

Autre point important, si nous utilisons une carte électronique par moteur, les ordres envoyés à ces deux cartes ne pourront être synchronisés par la carte mère.
Il sera alors utile d'envisager une méthode de synchronisation sur les cartes elles-mêmes. En effet, imaginons que l'on veuille faire avancer le robot en ligne droite, l'ordre n'arrivant pas simultanément aux deux roues, le robot pivotera avant de commencer à avancer.

Chapitre 3

Contrôle

Sommaire

3.1 Définition du problème

Avant d'établir les lois de contrôle qui vont être utilisées, il convient de définir de manière détaillée les fonctionnalités attendues.

Remarque : Dans toute l'étude qui va suivre, nous considérerons seulement la classe de robots à roues de type unicycle, celle-ci étant celle qui nous intéresse pour notre problème particulier.

Problème principal :
– Soit le robot dans une certaine configuration[1] donnée A (v_A, $\omega_A = 0$, θ_A, x_A, y_A), et soit une configuration désirée B ($v \leq v_{B_{MAX}}$, $\omega_B = 0$, θ_B, x_B, y_B). Trouver une loi de contrôle permettant de passer de la

[1]Par configuration, on entend la valeur de l'ensemble des paramètres du robot : sa vitesse longitudinale v, sa vitesse de rotation ω, son orientation θ, sa position en x et y.

configuration A à la configuration B.

Contraintes :
- Le robot doit rester dans un certain périmètre fixé.
- Le dérapage des roues doit être évité (pour rester en roulement sans glissement).
- Le temps d'exécution doit être minimisé.
- Les déplacements doivent pouvoir être enchaînés.

Remarquons que la vitesse de rotation aux points A et B est imposée nulle, cette règle est peu contraignante et a été fixée pour faciliter la possibilité d'enchaînement des déplacements, le point B pouvant ainsi être facilement réutilisé comme nouveau point A'.

Remarquons aussi que la vitesse longitudinale du robot au point B n'est pas imposée, seule une limite maximale est fixée. En effet, vu le périmètre limité pour effectuer le déplacement, il n'existera pas toujours de solution nous permettant d'atteindre une vitesse imposée au point B (l'accélération du robot n'étant pas infinie).

Ces choix nous permettent de limiter les contraintes sur les trajectoires suivies et nous assurent de toujours trouver une solution au problème posé si, et seulement si, il existe à partir du point A une trajectoire de freinage permettant au robot de s'immobiliser dans la limite du périmètre imposé.

En effet :

- Condition nécessaire :
  Si une telle trajectoire n'existe pas, il n'existe pas de point à l'intérieur du périmètre où le robot peut avoir une vitesse $v = 0$ et donc, en particulier, le problème A vers B avec $v_{B_{MAX}} = 0$ n'a pas de solution.

- Condition suffisante :
  Si une telle trajectoire existe, notons A' l'endroit où le robot s'immobilise et $v_{A'} = 0$ sa vitesse en ce point. Il est toujours possible de trouver une trajectoire sous la forme d'une suite de rotations simples et de lignes droites amenant le robot en un point B (interne au périmètre) quelconque avec une vitesse finale nulle. Or, comme nous avons A vers A' par hypothèse, le problème A vers B a toujours une solution.

Nous aurons toujours comme règle de bonne pratique de nous assurer que les points d'arrivées respectent la condition d'existence d'une trajectoire de freinage. Ainsi, ils pourront être, à leur tour, réutilisés sans problème comme points de départ. De plus, nous nous assurerons d'être en tout instant dans

une situation où le robot peut s'arrêter sans sortir de son périmètre et donc d'être toujours dans une configuration non dangereuse.

3.2 Choix de la solution

Maintenant que les bases du problème ont été posées, il reste à déterminer la méthode que nous allons utiliser pour sa résolution.

Quatre approches ont été envisagées :

1. Suite de rotations et lignes droites :
   Comme nous l'avons vu à la section 2.2.1, un robot unicycle peut avoir une commande très simple constituée d'une suite de rotations simples et de lignes droites. Cette solution est souvent utilisée en pratique mais possède un énorme défaut : à chaque transition rotation $\leftrightarrow$ ligne droite, nous sommes obligés de passer par une position de repos $(v = 0, \omega = 0)$. Nous serons donc souvent bien loin de l'optimum quant à la minimisation du temps d'exécution.

2. Asservissement sur la position finale :
   Une autre solution souvent utilisée consiste à asservir la vitesse longitudinale et la vitesse de rotation du robot sur la position en x et y de l'objectif. La vitesse longitudinale du robot étant régulée par rapport à la distance entre la position courante du robot et la position finale, la vitesse de rotation étant, quant à elle, régulée par rapport à l'angle entre la direction courante du robot et la direction relative de l'objectif. Lorsque la position finale en x et y est atteinte, on effectue éventuellement une rotation simple afin d'atteindre la position angulaire désirée.
   Cette solution nous permet d'atteindre des vitesses plus grandes, la trajectoire suivie étant une courbe, et donc un temps d'exécution en général bien meilleur que pour la solution précédente.
   L'énorme défaut de cette approche est que nous n'avons aucune connaissance a priori sur la trajectoire qui sera empruntée par le robot, nous n'avons donc, par exemple, aucune garantie que celui-ci reste dans son périmètre imposé.

3. Suivi de chemin :
   Afin de garantir le chemin emprunté par le robot, la solution est bien évidemment d'effectuer l'asservissement sur ce même chemin. C'est un problème non trivial pour lequel plusieurs approches ont été proposées, on notera particulièrement une des plus connues proposée par

C. Samson dans [MS93].
Un des défauts de cette approche, est qu'elle ne fait aucune hypothèse sur la trajectoire à suivre. Or dans le problème qui nous occupe, si nous voulons rester en roulement sans glissement, la trajectoire doit respecter certaines propriétés de courbures et les accélérations des roues doivent être maîtrisées. Cette approche ne permet pas de tenir compte de ces particularités.

4. Stabilisation sur une trajectoire avec profil de vitesse :
Afin de rester en roulement sans glissement, nous fixons des contraintes d'accélération et de vitesse sur chacune des roues du robot. Afin de respecter ces contraintes, un profil de vitesse devra être calculé en même temps que la trajectoire.
Ce profil, consistant à donner, à tout endroit de la trajectoire, la vitesse de chacune des roues du robot, pourra être utilisé lors du contrôle. En effet, il peut être vu comme la définition d'un robot virtuel de référence. Le problème de contrôle pourra, dès lors, être ramené au problème de stabilisation du robot mobile par rapport à ce robot de référence.

L'approche retenue pour la suite de ce travail est la stabilisation sur une trajectoire avec profil de vitesse. En effet, c'est celle qui va nous permettre de résoudre notre problème en respectant toutes les contraintes et ceci avec un temps de parcours proche de l'optimum.

Afin de respecter les contraintes d'architecture vues à la section 2.3, nous devons aussi quelque peu réfléchir à la mise en oeuvre avant de poursuivre notre étude. En effet, nous avons vu que nous disposions d'un ordinateur embarqué, mais que celui-ci était affublé d'un temps de réponse non négligeable.

Le contrôle devra donc être divisé en deux parties :

- Un contrôle local en vitesse des moteurs, géré par une carte électronique. (En fait une carte électronique par moteur dans un souci de modularité.)

- La génération de trajectoire et la stabilisation sur celle-ci gérées par l'ordinateur embarqué. On considère, pour la stabilisation de trajectoire, que les vitesses sont directement transmises aux moteurs (i.e. les vitesses sont parfaitement régulées).

En effet, la régulation en vitesse est peu gourmande en ressources mais demande un temps de réponse relativement faible, la génération de trajec-

toire et la stabilisation étant elles trop lourdes que pour tourner sur un microcontrôleur. De plus, la vitesse étant déjà régulée localement, la stabilisation pourra se satisfaire de temps de réponse plus longs.

Cela nous permet aussi une plus grande modularité dans notre approche, les deux contrôles (vitesse et stabilisation) étant effectués, chacun, de manière totalement indépendante.

3.3 Régulation locale

3.3.1 Régulation *PID*

Le contrôleur *PID* (*Proportional Integral Derivative Controler*) est un système de contrôle en boucle fermée très largement utilisé en pratique notamment pour ses bonnes performances dans un grand nombre d'applications mais aussi pour sa relative facilité d'implémentation dans des systèmes embarqués.

Il est défini de la manière suivante :

Soit $r(t)$ un signal de référence et $y(t)$ la sortie observée du système, le signal de contrôle $u(t)$ du système est donné par la loi :

$$\begin{aligned} u(t) =& k_p(b_{sp}r(t) - y(t)) + k_i \int_0^t r(\tau) - y(\tau)d\tau \\ &+ k_d \left(b_{sd}\frac{dr(t)}{dt} - \frac{dy(t)}{dt} \right) \end{aligned} \tag{3.1}$$

Ce qui donne dans le domaine de Laplace :

$$\begin{aligned} U(s) =& k_p(b_{sp}R(s) - Y(s)) + \frac{k_i}{s}(R(s) - Y(s)) \\ &+ k_d s(b_{sd}R(s) - Y(s)) \end{aligned} \tag{3.2}$$

Les différents paramètres du contrôleur *PID* sont les suivants :

- k_p , le gain proportionnel
- k_i , le gain intégral
- k_d , le gain dérivatif
- b_{sp} , le poids du signal de référence proportionnel
- b_{sd} , le poids du signal de référence dérivatif

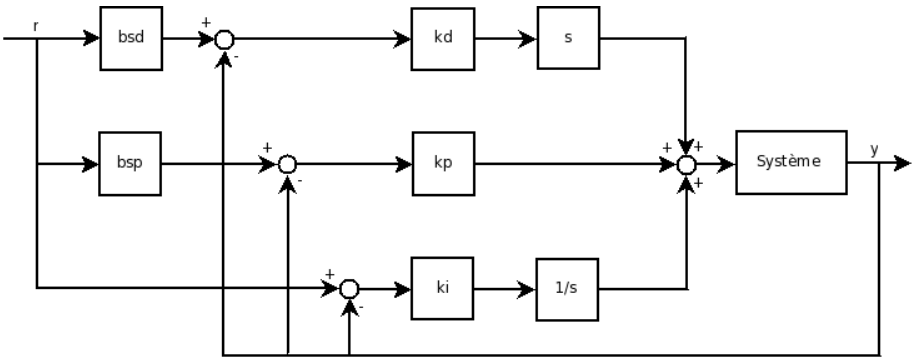

FIG. 3.1 – Bloc diagramme du contrôleur *PID*

Le rôle principal du gain k_p est la diminution du temps de réponse, mais une augmentation trop importante de celui-ci peut mener à une instabilité du système.

Le rôle principal du gain k_i est la suppression de l'erreur en régime stationnaire, une augmentation du gain permet une suppression plus rapide de l'erreur mais peut mener à des oscillations.

Démontrons la suppression de l'erreur en régime stationnaire si $k_i \neq 0$:

En régime stationnaire le signal de contrôle u_S est constant et l'erreur e_S est donc aussi constante. Il s'en suit que :

$$u_S = k_p(b_{sp}r_S - y_S) + k_i e_S t \tag{3.3}$$

Pour que l'égalité soit respectée, u_S étant constant, e_S doit être égal à zéro et il y a bien annulation de l'erreur en régime stationnaire.

Le rôle principal du gain k_d est la diminution des dépassements, le gain k_d agissant en quelque sorte par anticipation grâce à la présence de la dérivée. Remarquons qu'il faut être vigilant dans le cas d'une mesure bruitée du signal d'erreur car un des effets secondaires de la dérivation est justement d'augmenter ce bruit, la présence d'un gain k_d important peut donc mener à une instabilité.

Les poids b_{sp} et b_{sd} sont compris entre zéro et un et n'affectent pas le temps de réponse à des variations de charge du système, mais influencent la manière dont le système va réagir à des changements de signal de référence.

Integral windup

Un des problèmes majeurs pouvant survenir dans une boucle de contrôle *PID* pour laquelle $k_i \neq 0$ est le phénomène connu sous le nom d'*integral windup*.

Ce problème survient lorsque le système contrôlé peut entrer en saturation. En effet, lorsque le système sature (i.e. la sortie a atteint un maximum (resp. minimum)) mais que l'erreur est toujours positive (resp. négative), le terme intégral ne va cesser d'augmenter (resp. diminuer) tant que le système reste en saturation. La modification du signal de contrôle n'ayant plus d'effet sur la sortie, le terme intégral peut donc devenir très important. Ceci aura pour effet principal de nécessiter un changement de signe de l'erreur pendant un temps relativement conséquent avant que le terme intégral ne soit de nouveau nul.

Une méthode relativement simple pour supprimer ce phénomène est, par exemple, le blocage de l'intégration lorsque l'effet de celle-ci, via le signal de contrôle, cherche encore à augmenter (resp. diminuer) la sortie d'un système ayant atteint son maximum (resp. minimum). Cette méthode nécessite, bien entendu, un modèle de saturation du système contrôlé.

Des études plus détaillées du contrôleur *PID* pourront, par exemple, être trouvées dans [AM04] et dans [Sep07].

3.3.2 Régulation en vitesse

Le problème qui nous occupe est donc la régulation locale en vitesse d'un moteur à courant continu. Le contrôleur *PID* vu ci-dessus est particulièrement bien adapté à ce problème.

En fait, nous n'utiliserons en pratique qu'un contrôleur PI pour le contrôle en vitesse, la mesure de la vitesse étant en général trop bruitée que pour utiliser un terme dérivatif.

Le signal de contrôle sera exprimé comme la tension à appliquer aux bornes du moteur et sera donné par la loi suivante :

$$u(t) = k_p(b_{sp}v_{ref}(t) - v_{obs}(t)) + k_i \int_0^t v_{ref}(\tau) - v_{obs}(\tau)d\tau \tag{3.4}$$

avec v_{ref} la vitesse désirée et v_{obs} la vitesse mesurée du moteur.

Les différents paramètres sont fonction des caractéristiques propres du moteur utilisé.

Remarquons qu'il sera nécessaire d'appliquer une méthode pour supprimer le phénomène d'*integral windup*, la tension, pouvant être appliquée aux bornes du moteur, étant limitée.

3.3.3 Régulation en position

Le problème de la régulation locale en position d'un moteur à courant continu n'est pas ce qui nous occupe a priori. Mais dans un souci de généralité, il a été décidé que la carte électronique soit aussi dotée de cette fonctionnalité. Nous verrons, par ailleurs, à la section 3.5 que ce choix n'a pas été vain.

Un contrôleur *PID* pourra aussi être utilisé pour la résolution de ce problème.

Le signal de contrôle sera exprimé comme la tension à appliquer aux bornes du moteur et sera donné par la loi suivante :

$$\begin{aligned} u(t) =& k_p(p_{ref}(t) - p_{obs}(t)) + k_i \int_0^t p_{ref}(\tau) - p_{obs}(\tau) d\tau \\ &+ k_d \left(b_{sd} \frac{dp_{ref}(t)}{dt} - \frac{dp_{obs}(t)}{dt} \right) \end{aligned} \tag{3.5}$$

avec p_{ref} la position désirée et p_{obs} la position mesurée du moteur.

Les différents paramètres sont de nouveau fonction des caractéristiques propres du moteur utilisé.

Remarquons que nous avons forcé le terme b_{sp} à un. Il n'y avait en effet aucune raison d'influencer plus ou moins une position par rapport à une autre.

Remarquons qu'il sera aussi nécessaire d'appliquer une méthode pour supprimer le phénomène d'*integral windup* pour la même raison évoquée au point précédent.

En plus du simple contrôle en position, la carte électronique à été dotée de la possibilité de gérer des rampes d'accélération et de décélération lorsque ce mode de contrôle est utilisé. Lorsqu'une nouvelle position de référence est donnée, celle-ci n'est, en fait, répercutée que progressivement au contrôleur *PID* de manière à respecter les rampes imposées.

3.4 Suivi de trajectoire

3.4.1 Génération de trajectoire

Afin de passer d'une configuration A à une configuration B tout en respectant les différentes contraintes imposées (notamment roulement sans glissement des roues et respect d'un périmètre fixé), nous avons décidé de mettre en oeuvre du suivi de trajectoire. Il convient donc, d'abord, de savoir comment nous allons calculer cette trajectoire.

Pour rappel, le problème qui nous occupe consiste à amener le robot d'une configuration A (v_A, $\omega_A = 0$, θ_A, x_A, y_A), à une configuration B ($v \leq v_{B_{MAX}}$, $\omega_B = 0$, θ_B, x_B, y_B).

Remarque préalable : nous ferons pour l'instant abstraction des vitesses longitudinales possibles au départ et à l'arrivée, celles-ci interviendront dans un deuxième temps lors du calcul du profil de vitesse à la section 3.4.2.

La méthode employée pour générer les trajectoires est inspirée de [LNRL06] et de [Rib02], nous en donnons ici une approche intuitive.

Tout d'abord, remarquons que nous avons imposé des vitesses de rotation nulles au départ et à l'arrivée, ceci nous permet donc déjà de fixer la courbure initiale et finale de notre trajectoire à zéro, ce qui va faciliter notre étude.

Intuitivement, pour aller d'une configuration A à une configuration B, la première solution envisagée est d'effectuer une rotation sur soi-même, suivie d'une ligne droite et puis enfin d'une seconde rotation.
Cette solution est certainement la plus simple mais n'est pas nécessairement la meilleure. En effet, elle nous oblige à passer par des positions de repos lors des deux transitions. (Remarque : si la vitesse initiale n'est pas nulle, cette solution est même impossible)

Afin d'améliorer notre solution, on pourrait, par exemple, imaginer de remplacer les rotations par des arcs de cercle, ainsi on ne serait plus obligé de s'arrêter lors des transitions.

Prenons deux configurations A et B et plaçons les dans un système d'axes. Choisissons $r1$ le rayon de l'arc de cercle qui part de A et $r2$ le rayon de l'arc de cercle qui arrive en B.
Nous remarquons directement à la figure 3.2 que nous avons deux possibilités pour placer les cercles en chaque point. Ce qui nous fait donc 4 combinaisons possibles.

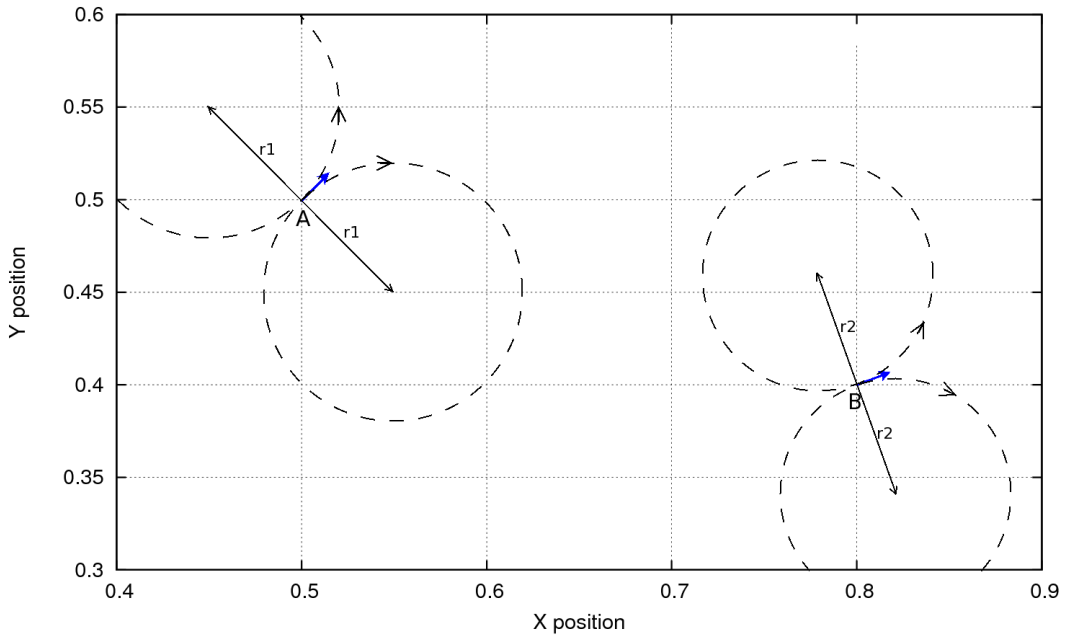

FIG. 3.2 – Génération de trajectoire - Configurations et cercles associés

Ensuite, pour chaque combinaison, nous cherchons une tangente commune aux deux cercles. C'est le chemin que nous emprunterons pour passer d'un arc de cercle à l'autre.

Outre les combinaisons incompatibles pour lesquelles il est impossible de trouver une tangente (deux cercles inscrits l'un dans l'autre par exemple), nous pouvons, dès à présent, éliminer certaines combinaisons. Celles amenant des trajectoires avec une rotation totale de plus de 2π rad, ou, les trajectoires enchaînant deux rotations de plus de π rad dans des sens opposés sont, par exemple, à proscrire. En effet, elles ne rencontreront probablement pas notre objectif de minimisation du temps d'exécution.

Pour toutes les autres combinaisons, nous calculerons une trajectoire. La méthode utilisée pour sélectionner la meilleure d'entre-elles sera vue à la section suivante.

Nous voyons à la figure 3.3 un choix de combinaison particulier et la tangente associée.

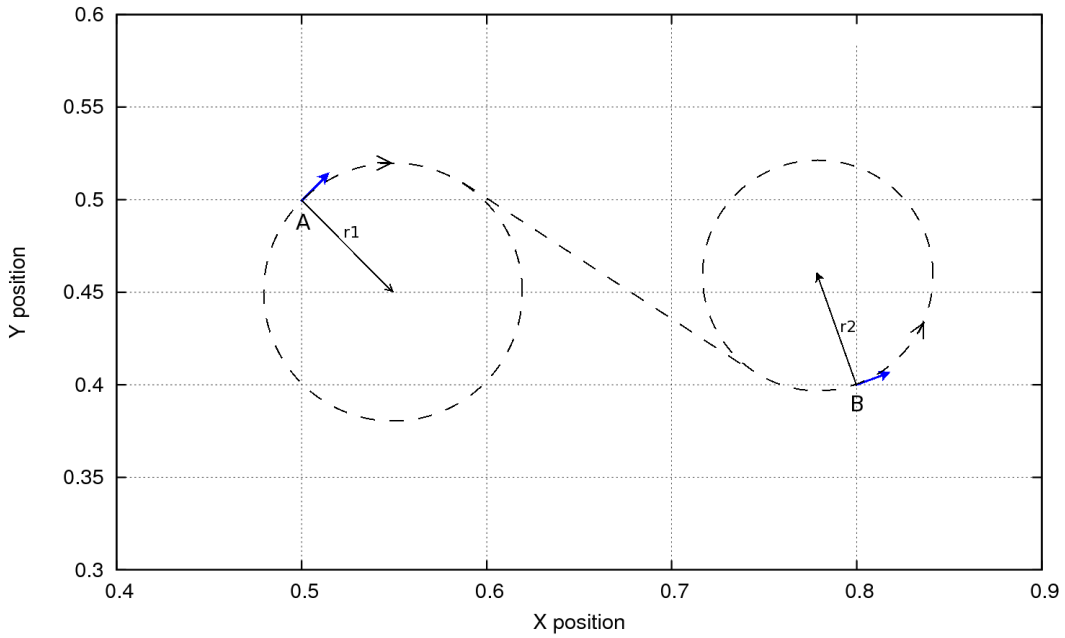

FIG. 3.3 – Génération de trajectoire - Tangente aux deux cercles

Le chemin (représenté à la figure 3.4) constitué du premier arc de cercle, de la tangente et du second arc de cercle forme donc à priori la trajectoire recherchée.

Malheureusement, notre problème n'est pas encore totalement résolu car cette trajectoire ne permet pas de respecter la contrainte de roulement sans glissement des roues que nous nous étions aussi fixée.

En effet, observons de plus près la courbure de notre trajectoire et plus particulièrement la transition entre la tangente et l'arc de cercle.
La courbure le long de la tangente est nulle et la courbure sur l'arc de cercle est constante et non nulle. Si nous voulons effectivement que notre robot parcoure cette trajectoire, les roues du robot lors de la transition devront subir une accélération infinie. Ce qui est physiquement impossible et nous aurons, au mieux, un dérapage des roues.

Nous allons, pour régler ce problème, faire appel à une construction mathématique bien connue des ingénieurs civils spécialisés dans la construction d'autoroutes : la clothoïde (ou spirale de Cornu).

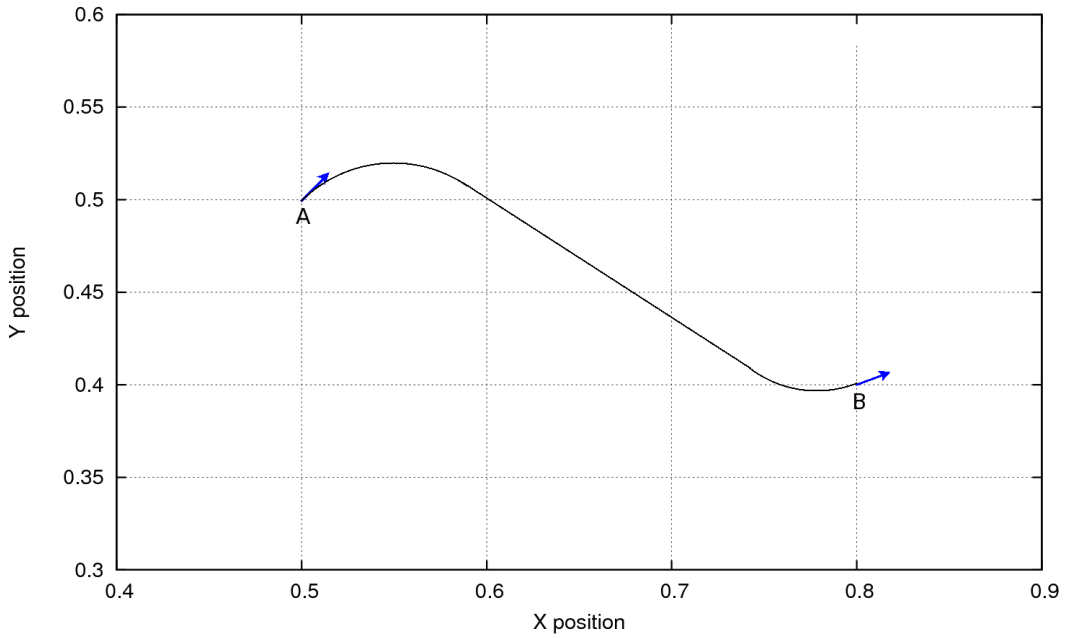

FIG. 3.4 – Génération de trajectoire - Trajectoire (premier essai)

Une clothoïde est définie par une équation paramétrique du type :

$$\begin{cases} x(t) = k \int_0^t \sin(u^2) du \\ y(t) = k \int_0^t \cos(u^2) du \end{cases} \tag{3.6}$$

La clothoïde a la particularité d'avoir une courbure proportionnelle à son abscisse curviligne. Ce qui se traduit pratiquement par le fait que lorsque l'on parcourt une telle courbe à vitesse constante, la courbure varie de manière proportionnelle au temps.

Nous pourrons donc raccorder une ligne droite à un arc de cercle par l'intermédiaire d'une clothoïde de sorte que la courbure varie de manière continue le long de la courbe.

Ces propriétés intéressantes font que les clothoïdes sont très souvent utilisées pour le tracé d'autoroutes ou encore de chemins de fer.

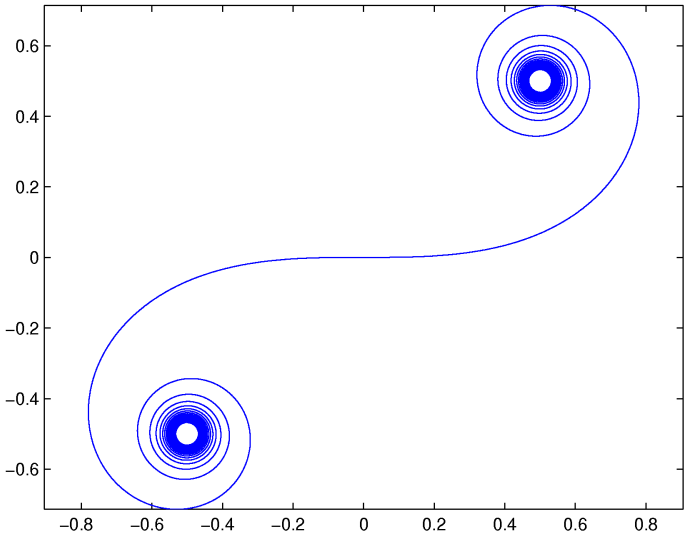

FIG. 3.5 – Clothoïde (ou spirale de Cornu)

Pour notre trajectoire, nous allons remplacer les deux arcs de cercles problématiques par des ensembles “clothoïde - arc de cercle - clothoïde”. Cela va nous permettre de passer d'une courbure nulle à une courbure constante, puis, de nouveau, à une courbure nulle, d'une manière continue.

Notre trajectoire finale sera donc constituée de la manière suivante :

clothoïde - arc de cercle - clothoïde - ligne droite - clothoïde - arc de cercle - clothoïde

Nous voyons à la figure 3.6 la trajectoire finale générée en rouge et la trajectoire sans clothoïde en pointillé pour comparaison.

La dernière contrainte, dont nous devons encore tenir compte, est celle qui impose que le robot reste dans un certain périmètre donné. Ceci sera vérifié simplement de la façon suivante : les ensembles clothoïdes et arcs de cercle doivent être inclus au périmètre (avec éventuellement une marge de sécurité correspondant au rayon du robot).
Nous testerons soit l'absence d'intersection entre les différentes courbes et le périmètre, soit, et c'est la méthode que nous avons retenue, nous discrétisons chaque courbe et ensuite testons si tous ses points sont bien inclus au périmètre.

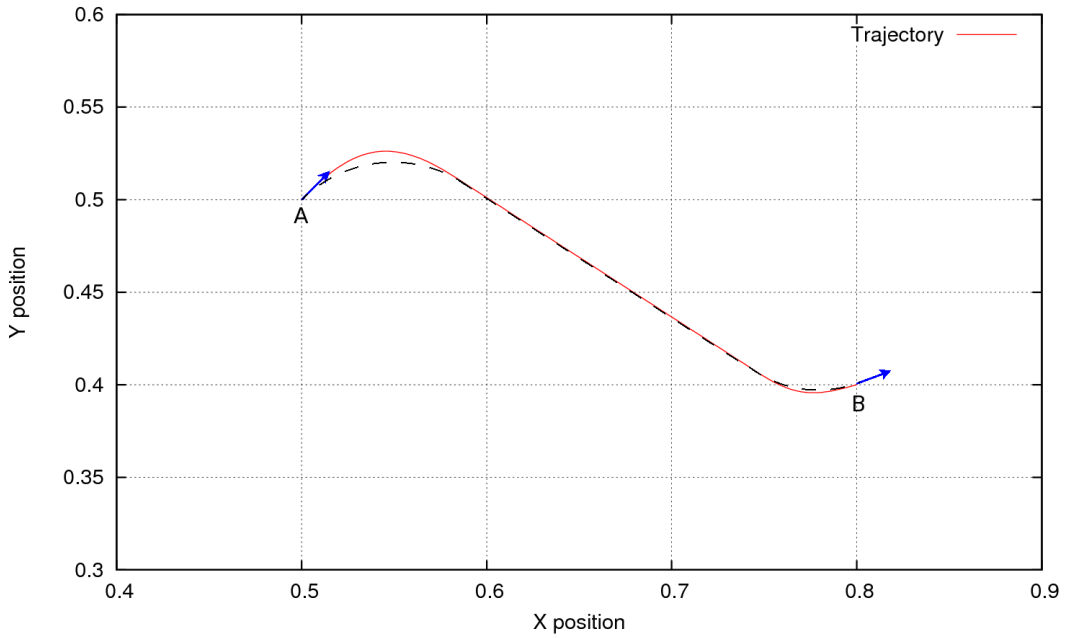

FIG. 3.6 – Génération de trajectoire - Trajectoire finale

Si la contrainte n'est pas respectée, la combinaison de cercles est éliminée et ne fournit pas de trajectoire. Si aucune combinaison de cercles ne permet de trouver une trajectoire compatible, nous serons amenés à générer une trajectoire avec des rayons $r1$ et $r2$ différents.

La manière dont nous allons sélectionner les deux rayons ainsi que la combinaison associée sera vue à la section suivante.

Remarquons que si nous prenons $r1 = 0$ et $r2 = 0$, nous dégénèrons vers la solution naïve du début.

Nous avons donc maintenant un moyen de déterminer une trajectoire réalisable pour notre robot permettant de passer d'un point A à un point B pour $r1$ et $r2$ donnés.

3.4.2 Profil de vitesse

Une fois la trajectoire déterminée au moyen de la méthode vue à la section précédente, il faut maintenant lui adjoindre un profil de vitesse. Ce

profil nous permettra, entre autre, de savoir si cette trajectoire est compatible avec la vitesse initiale du robot dont nous n'avons pas encore tenu compte.

Pour établir notre profil de vitesse, nous avons besoin de quatre paramètres :

1. La vitesse maximale d'une roue.
2. L'accélération maximale d'une roue.
3. La vitesse maximale du robot.
4. L'accélération maximale du robot.

Afin de maximiser la vitesse et l'accélération du robot en ligne droite, nous prendrons les deux paramètres de vitesse et les deux paramètres d'accélération égaux.

Notre profil de vitesse est défini de la manière suivante : nous déterminons la vitesse longitudinale maximale du robot en chaque point telle que si la suite de vitesses calculées est appliquée à un robot suivant cette trajectoire, on minimise le temps de parcours tout en respectant les contraintes de vitesse et d'accélération sur le robot et sur les roues.

Ainsi défini, ce profil nous permettra de garantir, en tout instant, le roulement sans glissement des deux roues (à la condition de choisir correctement nos paramètres de vitesse et d'accélération maximales).

Pour déterminer notre profil de vitesse en chaque point de la courbe, nous déterminons la vitesse maximale compatible, d'une part, avec les contraintes en vitesse et en accélération, et, d'autre part, avec les trois paramètres supplémentaires suivants : la vitesse calculée au point précédent, la courbure de la trajectoire au point courant et la variation de courbure entre le point courant et le point suivant.
Cette vitesse est donc le résultat d'un problème d'optimisation et est telle que si on l'applique à un robot en ce point de la trajectoire, les contraintes de vitesse et d'accélération sont respectées pour le robot lui-même mais aussi pour chaque roue indépendamment.
Si une telle vitesse n'existe pas, celle calculée au point précédent était manifestement trop importante et la vitesse maximale que nous fixons alors ne tient compte que de la contrainte en vitesse et des deux paramètres qui sont : la courbure de la trajectoire au point courant et la variation de courbure entre le point courant et le point suivant. Nous avons alors une discontinuité dans notre profil de vitesse.

Pour régler ce problème, nous procédons en fait en deux passes distinctes, l'une parcourant la trajectoire dans son sens normal et qui nous permet de tenir compte des contraintes d'accélération fixées par la trajectoire, l'autre parcourant la trajectoire dans le sens inverse et qui nous permet de tenir compte des contraintes de décélération.

La vitesse initiale pour la première passe est choisie égale à la vitesse initiale de notre problème, la vitesse initiale de la seconde passe est quant à elle choisie égale à la vitesse finale trouvée lors de la première passe.
Le profil de vitesse sera alors constitué du minimum en chaque point pour les deux passes.

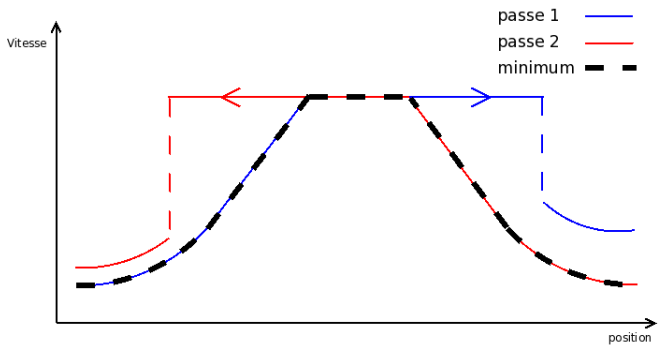

FIG. 3.7 – Profil de vitesse - Les deux passes et le minimum

Remarquons que nous avons maintenant un moyen simple pour tester si une trajectoire est compatible avec notre vitesse initiale imposée. Il suffit simplement de regarder si celle-ci est bien inférieure ou égale à la vitesse finale donnée par la seconde passe. Si tel n'est pas le cas, la trajectoire ne pourra pas être suivie tout en respectant les contraintes fixées.

Remarquons aussi que le profil de vitesse trouvé par cette méthode est bien continu. Les profils donnés par les deux passes sont continus par morceaux, mais les discontinuités interviennent, par définition, toujours sur le profil le plus rapide et ne seront donc, jamais, répercutées sur le minimum des deux profils. En effet, nous aurions discontinuité en un point si nous y étions trop rapides. La passe inverse, tenant compte des contraintes opposées, aura toujours en ce point une vitesse inférieure puisque compatible.

Remarquons, enfin, qu'à partir de notre profil de vitesse longitudinale et de la courbure en chaque point, on peut déterminer un deuxième profil de vitesse équivalent donnant la vitesse de la roue gauche et de la roue droite en chaque point. Ce deuxième profil, plus commode, sera utilisé à la section 3.4.3 pour effectuer le suivi à proprement dit.

Nous savons, à présent, comment déterminer une trajectoire réalisable (si elle existe) pour passer d'une configuration A à une configuration B avec $r1$ et $r2$ donnés. Nous allons maintenant voir comment utiliser le profil en vitesse pour fixer $r1$ et $r2$ de manière à minimiser notre temps de parcours.

En utilisant les informations de distance de la trajectoire et les informations de vitesse du profil, nous pouvons mesurer le temps de parcours d'un robot virtuel qui suivrait ce trajet.
Nous allons utiliser cette information de temps, pour nous permettre, si pas d'atteindre la trajectoire optimale, au moins de nous en approcher au maximum.

Premièrement, pour une paire $r1$ et $r2$ donnée, on peut déjà ne garder que la trajectoire qui a le plus petit temps de parcours au cas où ces rayons nous en donnent plusieurs.
Pour déterminer les rayons, on utilisera la procédure suivante : nous choisissons d'abord $r1$ et $r2$ égaux et relativement grands, ensuite diminuons les rayons progressivement jusqu'à d'atteindre un minimum pour le temps de parcours.(Le choix du rayon à réduire et du facteur de réduction se fait de manière heuristique.)

Une autre possibilité, encore non envisagée mais tout aussi importante pour diminuer le temps de parcours, est la possibilité d'effectuer notre déplacement en marche arrière. En effet, dans toute notre étude nous avons considéré que le robot se déplaçait toujours en marche avant. Cependant vu la construction du robot, la marche arrière est tout à fait symétrique à la marche avant (i.e. vitesse et commandabilité identique) et peut donc aussi être utilisée. Pour déterminer notre trajectoire finale, nous effectuerons donc tout notre développement deux fois : pour la marche avant et pour la marche arrière. Nous choisirons ensuite la trajectoire de durée minimum.

Le programme utilisé pour la génération de la trajectoire optimale et de son profil de vitesse associé pourra être trouvé sur le cd-rom joint à ce mémoire dans le répertoire “traj/”.

Ce programme est le fruit d'une collaboration avec d'autres membres de l'équipe participant au projet *Eurobot* 2008. Il met en oeuvre les techniques vues dans cette section et dans la section précédente. Il prend en entrée deux configurations et génère en sortie (si possible) une trajectoire réalisable discrétisée sous la forme d'un ensemble de configurations permettant le passage entre les deux configurations d'entrée.

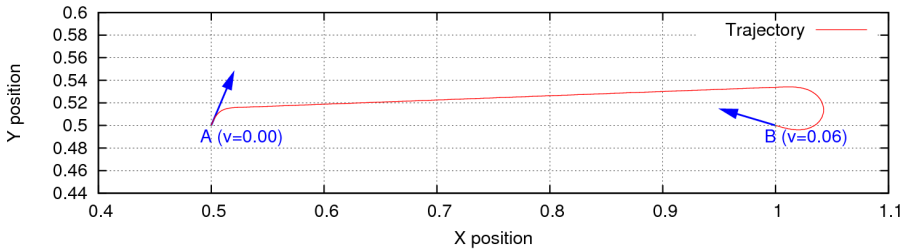

FIG. 3.8 – Exemple 1 - Configurations avec vitesse initiale nulle et vitesse finale libre

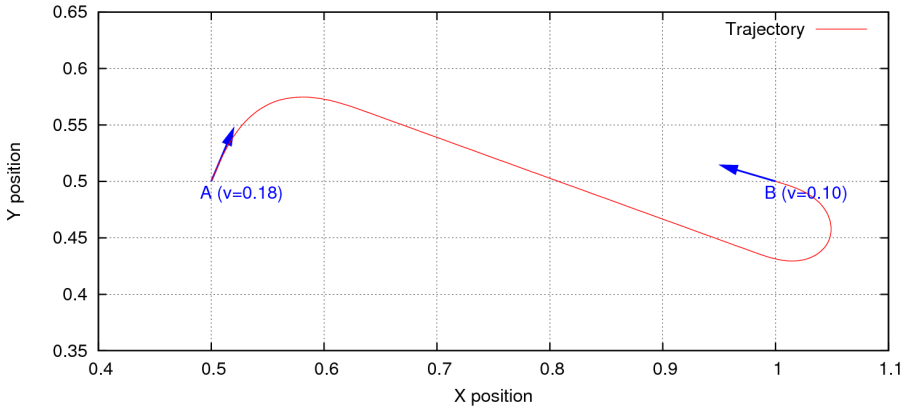

FIG. 3.9 – Exemple 2 - Les mêmes configurations qu'à la figure précédente mais avec une vitesse initiale imposée

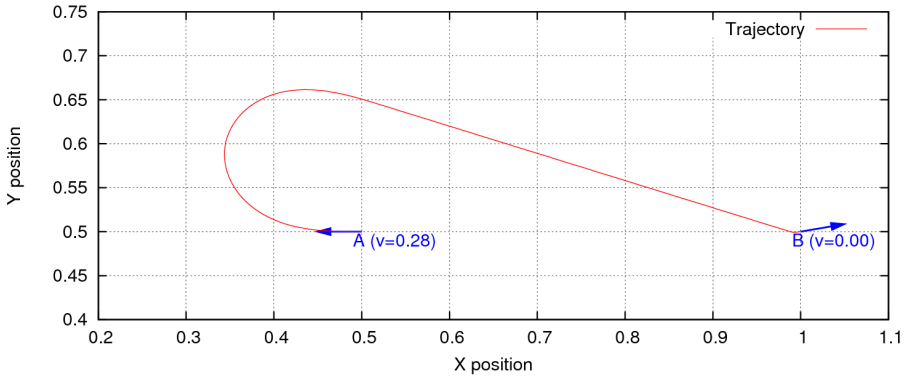

FIG. 3.10 – Exemple 3 - Configurations avec vitesse initiale imposée et vitesse finale nulle

3.4.3 Régulation sur la trajectoire

Le deuxième problème qui nous occupe est le suivant : pour une trajectoire réalisable donnée et un profil de vitesse associé, déterminer la commande à donner au robot pour qu'il effectue le suivi de cette trajectoire.

Remarque préalable : le profil de vitesse que nous utiliserons dans cette section n'est pas le profil de vitesse longitudinale mais son équivalent vitesse roue gauche et vitesse roue droite.

Dans un monde idéal, la commande à envoyer au robot pour qu'il suive la trajectoire désirée serait constituée uniquement de notre profil de vitesse. Ce ne sera bien entendu pas le cas, d'une part, parce que le contrôle en vitesse des roues n'est pas parfait et, d'autre part, parce que le robot peut subir des perturbations.
Il nous faut donc établir une loi de commande qui permette de tenir compte des perturbations éventuelles.

La commande utilisée est inspirée de [Gau99], qui présente une version simplifiée de la commande pour un suivi de chemin de C. Samson [MS93]. La différence est que nous n'allons pas essayer ici de suivre de manière géométrique un chemin, mais essayer de nous stabiliser autour d'une trajectoire décrite par un robot virtuel de référence.

Premièrement, nous avons v_{prof} et ω_{prof}, les vitesses longitudinale et de rotation du robot données par le profil de vitesse :

$$v_{prof} = \frac{v_{l_{prof}} + v_{r_{prof}}}{2} \tag{3.7}$$

$$\omega_{prof} = \frac{v_{r_{prof}} - v_{l_{prof}}}{L} \tag{3.8}$$

Ensuite, soit P, le point courant de la trajectoire, notons d la distance signée[2] entre ce point et le centre du robot, et notons θ_e l'erreur d'orientation du robot (i.e. la différence entre l'orientation réelle du robot et celle prévue sur la trajectoire au point P).

La correction à apporter à la vitesse de rotation du robot sera donnée par la loi suivante :

$$\omega_c = -k_1 v_{prof} d - k_2 \left| v_{prof} \right| \theta_e \tag{3.9}$$

avec k_1 et k_2 deux constantes positives.

[2]Par distance signée, on entend la distance conventionnelle à laquelle on adjoint un signe : positif lorsque le point P de la trajectoire est situé à la droite du robot et négatif lorsqu'il est situé à la gauche.

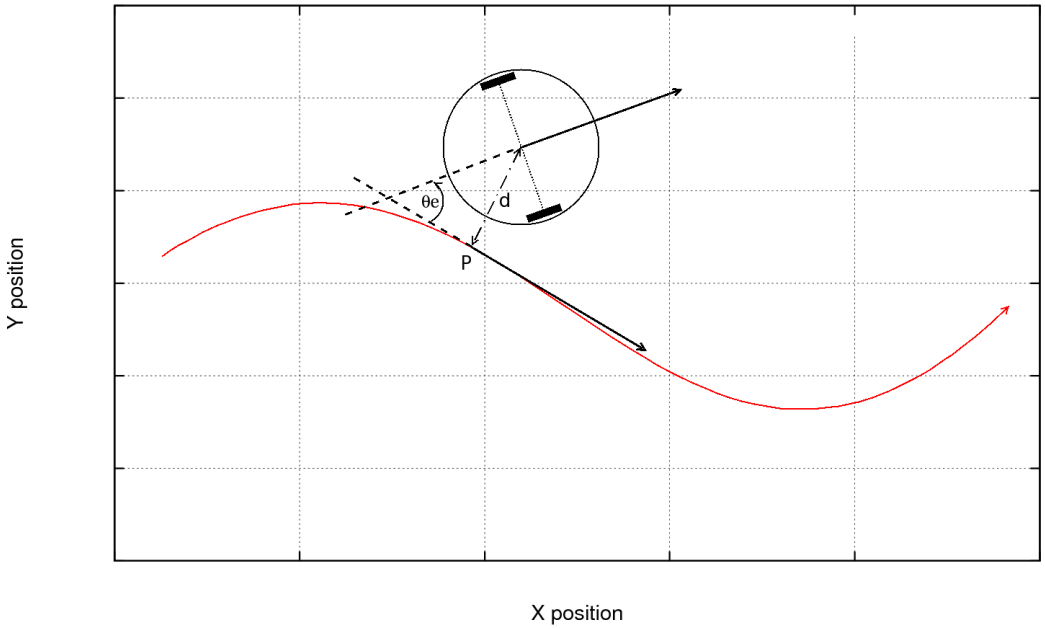

FIG. 3.11 – Régulation sur la trajectoire

Le but de cette correction étant de réduire l'erreur en distance et l'erreur en orientation du robot.

Les vitesses à donner au robot seront donc :

$$v = v_{prof} \tag{3.10}$$

$$\omega = \omega_{prof} + \omega_c \tag{3.11}$$

et les vitesses à envoyer aux roues :

$$v_l = v - \frac{L}{2}\omega,\ v_r = v + \frac{L}{2}\omega \tag{3.12}$$

où L est la distance entre les deux roues motrices.

Nous voyons aussi dans [Gau99] que les valeurs de k_1 et de k_2 peuvent être obtenues lors du suivi d'une trajectoire en ligne droite de la manière suivante :

$$k_1 = \xi^2,\ k_2 = \zeta\xi \tag{3.13}$$

où ξ la pulsation propre du système et ζ son amortissement.

Nous avons donc maintenant une commande qui nous permet de corriger les erreurs de suivi dues aux perturbations et qui nous permet, en théorie, d'assurer une convergence géométrique vers la trajectoire de référence.

Cette commande ne nous permet, par contre, plus de respecter notre contrainte de roulement sans glissement. En effet, celle-ci était respectée si on appliquait scrupuleusement les vitesses données par notre profil, ce qui n'est ici plus le cas.
Pour nous tirer de ce mauvais pas, nous allons procéder de la manière suivante : nous n'allons pas utiliser les vrais paramètres maximaux pour la génération de trajectoire, mais des paramètres réduits. Ceci va nous permettre d'avoir une certaine marge de manoeuvre pour effectuer notre correction. Les vrais paramètres seront, quant à eux, utilisés par la suite, afin de filtrer les vitesses réellement envoyées aux roues.

La dernière question qui peut se poser pour la régulation, est de savoir comment déterminer le point P courant de notre trajectoire de référence. L'approche que nous présentons est la suivante : P est choisi parmi l'ensemble de points constitué de l'ancien point courant et tous les points suivants de la trajectoire. Celui-ci étant le premier point (repéré dans le sens de parcours) qui satisfait un minimum local de la fonction erreur :

$$f_e(p) = a_1|d(p)| + a_2\theta_e(p) \tag{3.14}$$

avec a_1 et a_2 des fonctions de la courbure au point P.

Les fonctions a_1 et a_2 sont choisies de manière à donner une contribution maximale à l'erreur en distance lorsque la courbure est nulle, et une contribution maximale à l'erreur en orientation lorsque la courbure est infinie (i.e. une rotation sur soi-même).

On choisira par exemple :

$$a_1(\gamma) = \frac{2}{\pi}\arctan\left(\frac{1}{\frac{L}{2}|\gamma|}\right) \; si\ \gamma > 0\ et\ a_1 = 1\ si\ \gamma = 0 \tag{3.15}$$

$$a_2(\gamma) = \frac{2}{\pi}\arctan\left(\frac{L}{2}|\gamma|\right) \tag{3.16}$$

où γ est la courbure et L est la distance entre les deux roues motrices.

Ces fonctions donnent :
- Pour une ligne droite : $a_1 = 1$ et $a_2 = 0$
- Pour une rotation simple : $a_1 = 0$ et $a_2 = 1$
- Pour une rotation de rayon $\frac{L}{2}$: $a_1 = 0.5$ et $a_2 = 0.5$

Cette approche s'est montrée nettement plus satisfaisante que l'approche, souvent rencontrée, tenant compte uniquement de l'erreur en distance. Ceci particulièrement dans les portions de trajectoire à forte courbure.

3.5 Positionnement précis

La méthode de stabilisation sur une trajectoire vue à la section précédente fonctionne très bien, mais peut parfois s'avouer légèrement imprécise sur la position finale réellement atteinte. Cette imprécision se manifeste, en général, surtout sur la position angulaire du robot à l'arrivée.

Cela ne pose aucun problème dans le cas d'enchaînements de mouvements, mais cela peut parfois être gênant lorsque la vitesse à l'arrivée est nulle et que la position finale à besoin d'être très précise (si le robot doit effectuer une manipulation par exemple).

Pour corriger ce petit défaut nous allons faire appel à la possibilité de contrôle en position des moteurs.

Remarquons que nous pouvons utiliser les rampes d'accélérations et de décélérations fournies lors du contrôle en position afin de respecter la contrainte de roulement sans glissement des roues.

Méthode naïve

Trois cas peuvent se présenter :

1. Une légère erreur dans la position angulaire

2. Une légère erreur dans la direction du robot, couplée ou non à une légère erreur de position angulaire

3. Une légère erreur dans une direction perpendiculaire à la direction du robot, couplée ou non aux deux autres erreurs.

Dans le premier cas, on effectuera juste une légère rotation du robot.

Dans le second cas, un léger déplacement en ligne droite suivi d'une légère rotation du robot.

Et enfin dans le troisième cas, une rotation suivie d'un léger déplacement en ligne droite, elle même suivie d'une deuxième rotation.

Pour une rotation $d\theta$ nous enverrons comme variations de position aux roues :

$$p_r = \frac{Ld\theta}{4\pi r} 2\pi = \frac{Ld\theta}{2r} \tag{3.17}$$

$$p_l = -\frac{Ld\theta}{4\pi r} 2\pi = -\frac{Ld\theta}{2r} \tag{3.18}$$

avec r le rayon d'une roue et L la distance entre les deux roues motrices et p_r, p_l les variations de position exprimées en radians

Pour un déplacment dy dans la direction du robot nous enverrons comme variations de position aux roues :

$$p_r = p_l = \frac{dy}{2\pi r} 2\pi = \frac{dy}{r} \tag{3.19}$$

avec r le rayon d'une roue et p_r, p_l les variations de position exprimées en radians

Dans la pratique, le troisième cas n'est pour ainsi dire jamais utilisé (sauf si l'erreur est importante) car il est trop contraignant en temps d'exécution.

Méthode de l'*optimum*

Une approche différente consiste à ne pas vouloir annuler l'erreur à tout prix, mais à calculer les variations de position à envoyer aux roues afin de minimiser celle-ci. Cela va nous permettre de nous approcher de notre objectif en un seul mouvement (et donc dans un temps d'exécution très court).

Plaçons nous dans un repère centré sur le robot, l'axe y pointant dans la direction du robot et l'axe x dans la direction perpendiculaire.

Nous pouvons noter e_y, e_x et e_θ les erreurs de position du robot dans ce système d'axes, et donner un poids plus ou moins important à chaque type d'erreur suivant l'importance qu'on leur accorde.

Le problème d'optimisation consiste alors, par exemple, à trouver le vecteur déplacement $\boldsymbol{d} = \{d_r, d_l\}$ tel que :

$$arg \min_{\boldsymbol{d}} K_y(e_y - d_y)^2 + K_x(e_x - d_x)^2 + K_\theta(e_\theta - d_\theta)^2 \tag{3.20}$$

avec

si $d_r = d_l$:

$$d_\theta = 0 \tag{3.21}$$

$$d_y = d_r = d_l \tag{3.22}$$

$$d_x = 0 \tag{3.23}$$

si $d_r \neq d_l$:

$$d_\theta = \frac{d_r - d_l}{L} \tag{3.24}$$

$$d_y = \sin\left(\frac{d_r - d_l}{L}\right)\frac{L(d_l + d_r)}{2(d_r - d_l)} \tag{3.25}$$

$$d_x = \cos\left(\frac{d_r - d_l}{L}\right)\frac{L(d_l + d_r)}{2(d_r - d_l)} - \frac{L(d_l + d_r)}{2(d_r - d_l)} \tag{3.26}$$

où L est la distance entre les deux roues motrices.

Ces équations sont dérivées du fait que le mouvement réalisé par le robot est un arc de cercle.

Ce problème peut, par exemple, être résolu par l'algorithme de Levenberg-Marquardt (voir Annexe D) pour lequel nous utiliserons le vecteur initial $\boldsymbol{d} = \{0, 0\}$.

Les variations de position à envoyer aux roues seront alors :

$$p_r = \frac{d_r}{r},\ p_l = \frac{d_l}{r} \tag{3.27}$$

où r est le rayon d'une roue et p_r, p_l les variations de position exprimées en radians.

Remarque : les variations de position données par cette méthode n'étant pas égales ou opposées, nous devrons jouer sur les pentes d'accélération et de décélération des deux roues afin que les déplacements commencent et se terminent plus ou moins simultanément.

Remarquons que dans le cas où la minimisation de l'erreur ne serait pas trouvée suffisante par rapport à la précision désirée, il est envisageable d'effectuer la méthode plusieurs fois de suite (éventuellement en modifiant les poids donnés aux erreurs pour insister plus sur une erreur ou sur une autre selon les itérations).
Ce procédé sera toujours plus efficace en terme de temps d'exécution que la méthode naïve qui ne permettait que des rotations et des lignes droites.

Cette solution n'a pas encore pu être testée en pratique, la méthode précédente s'étant avérée suffisante pour notre utilisation.

Chapitre 4

Electronique

Sommaire

4.1 Définition du problème

Avant de passer à la réalisation pratique de l'électronique, il convient de faire un rappel des différentes fonctionnalités attendues.

La carte électronique doit être en mesure de commander un moteur en boucle ouverte ou en boucle fermée (régulation en vitesse ou en position). Le moteur à commander, est un moteur à courant continu 15V de 90W, ses caractéristiques donnent $45A$ comme courant de démarrage et $4A$ comme courant nominal.

L'alimentation de la partie logique est fournie par un bus de périphériques (voir 2.3.2), l'alimentation de la partie puissance devant elle être séparée. Remarque : les masses des deux alimentations sont connectées entre elles en un point unique extérieurement à la carte électronique.

Toutes les communications de la carte avec l'extérieur doivent se faire via un bus I^2C, la carte doit en outre être capable de gérer un signal d'arrêt d'ur-

gence du moteur via un mécanisme matériel sûr. La carte doit aussi être en mesure de détecter une situation dangereuse pour le moteur et l'électronique, comme un blocage des roues par une mesure du courant consommé.

La mesure de la position relative du moteur est fournie pas un encodeur incrémental monté directement sur l'axe du moteur.

4.2 Puissance

4.2.1 Pont en H

Un des montages les plus utilisés pour la commande d'un moteur à courant continu est le variateur de courant continu bidirectionnel ou pont en H (voir entre autres [Büh91] et [Gen05]). Ce montage est représenté à la figure suivante.

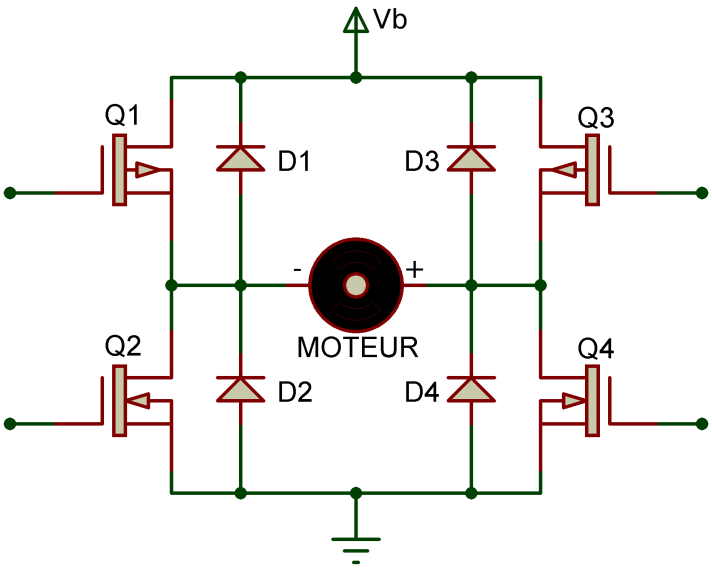

FIG. 4.1 – Schéma général du pont en H

Le fonctionnement de base du pont en H est relativement simple, il permet d'alimenter le moteur dans un sens ou dans l'autre, rendant ainsi possible son changement de sens de rotation.

En effet, si Q2 et Q3 conduisent, la borne moins du moteur sera connectée à la masse et la borne plus à la tension d'alimentation faisant ainsi tourner le moteur dans son sens de rotation normal.
Si maintenant ce sont Q1 et Q4 qui conduisent, la borne moins du moteur

sera connectée à la tension d'alimentation et la borne plus à la masse faisant alors tourner le moteur en sens inverse.

Pour faire varier la vitesse du moteur, il faut appliquer une tension variable aux bornes de celui-ci. Pour ce faire, la commutation des transistors est commandée par un signal PWM (*Pulse With Modulation*) de fréquence relativement élevée ($> 10KHz$). Le moteur passe ainsi de manière très rapide d'état alimenté à non alimenté et vice-versa.
La tension moyenne vue par le moteur (charge inductive) sera alors déterminée par le rapport cyclique (i.e. le rapport entre le temps ON et le temps OFF) du signal PWM utilisé.

Les diodes D1 à D4 sont appelées diodes de roue libre et sont nécessaires au bon fonctionnement du pont en H. Lorsque le moteur est alimenté (i.e. Q2 et Q3 conduisent ou Q1 et Q4 conduisent) ces diodes n'ont aucun effet. Elles servent uniquement lorsque le moteur passe d'un état alimenté à un état non alimenté. En effet, dans ce cas, les transistors vont arrêter brusquement de conduire le courant et celui-ci pourra alors passer par les diodes de roue libre. Cela permet de limiter les pics de tension inverse aux bornes du moteur et ainsi d'éviter la destruction des composants électroniques.

Remarquons, que les transistors Q1 et Q2 ou Q3 et Q4 ne doivent jamais conduire simultanément sous peine de court-circuit franc et de destruction des transistors.

Le pont en H peut être commandé de plusieurs façons différentes suivant la manière dont les commutations des transistors sont enchaînées. Nous allons étudier ci-après les deux modes de commande les plus connus.

Les informations concernant les modes de contrôle particuliers des ponts en H sont principalement tirées de notes d'applications de pont-H intégrés, en particulier, nous citerons celle du *LMD18200* de National Semiconductor [Reg99].

Sign - Magnitude

Dans le mode *Sign - Magnitude*, pour une direction donnée, le signal PWM fait varier les transistors entre deux phases.
Dans la première phase, un des transistors de la partie haute du pont et le transistor opposé de la partie basse conduisent et alimentent le moteur. Dans la deuxième phase, le transistor de la partie basse est coupé mettant alors le moteur en roue libre.

Lors de la transition de la première phase à la deuxième phase, le courant ne pouvant plus passer par le transistor bas, passe par la diode haute correspondante.

La figure 4.2 représente le diagramme temporel du mode *Sign - Magnitude*. Les signaux de commutation des transistors, les tensions aux bornes du moteur et le courant passant dans le moteur y sont représentés.

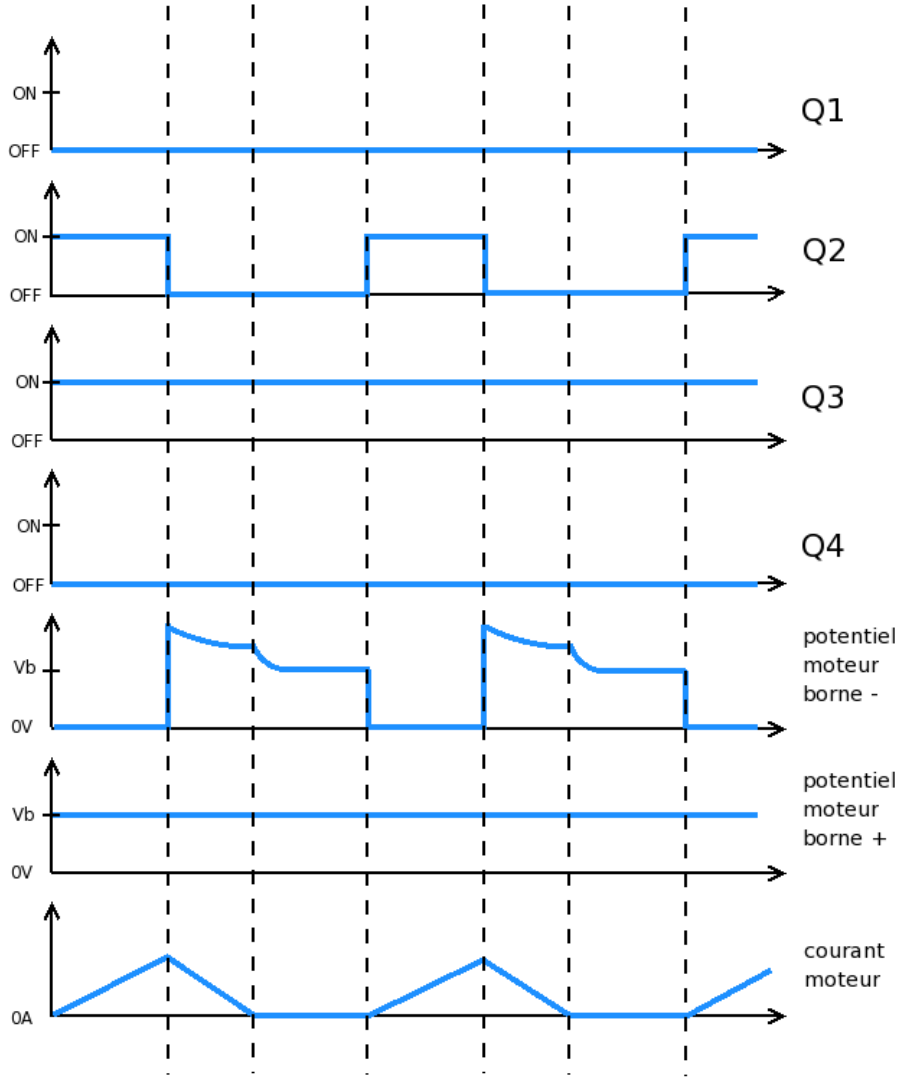

FIG. 4.2 – Diagramme temporel *Sign - Magnitude*

Locked Anti-phase

Dans le mode *Locked Anti-phase*, aucune distinction n'est faite quant à la direction de rotation du moteur, les deux directions étant contrôlées directement par le signal *PWM*. Ce mode est aussi divisé en deux phases.
Dans la première phase, un des transistors de la partie haute du pont et le transistor opposé de la partie basse conduisent et alimentent le moteur dans un certain sens.
Dans la deuxième phase, ce sont les deux autres transistors qui conduisent et qui alimentent alors le moteur dans le sens inverse.
Le moteur sera donc connecté en permanence à une source de faible impédance.

Il est à remarquer que pour mettre le moteur à l'arrêt, un rapport cyclique de 50% doit être utilisé.

La figure 4.3 représente le diagramme temporel du mode *Locked Anti-phase*. Les signaux de commutation des transistors, les tensions aux bornes du moteur et le courant passant dans le moteur y sont représentés.

Comparaison des deux modes

Bien que le mode *Sign - Magnitude* soit très souvent utilisé, c'est loin d'être la solution idéale pour notre utilisation. En effet, il est particulièrement mal adapté au contrôle en boucle fermée car il ne permet pas d'avoir un contrôle total du moteur. Le moteur étant en roue libre pendant une partie du temps, il sera en fait impossible de le freiner activement. La seule solution pour réaliser un freinage dans ce mode est de court-circuiter les bornes du moteur.

Le mode *Locked Anti-phase* permet, lui, de contrôler parfaitement l'accélération et la décélération du moteur. En effet, le moteur est toujours connecté à une source de faible impédance et donc la tension aux bornes du moteur est toujours imposée. La tension moyenne vue par le moteur sera, dès lors, directement proportionnelle au rapport cyclique de la *PWM*. Ce qui est capital pour notre contrôle en boucle fermée.

Le principal problème du *Locked Anti-phase* est qu'il nécessite de travailler avec des fréquences de *PWM* bien plus élevées que pour le *Sign - Magnitude*, ceci afin de lisser correctement le courant passant dans le moteur. (En pratique, une centaine de KHz pour le *Locked Anti-phase* alors qu'une dizaine suffit largement pour le *Sign - Magnitude*)

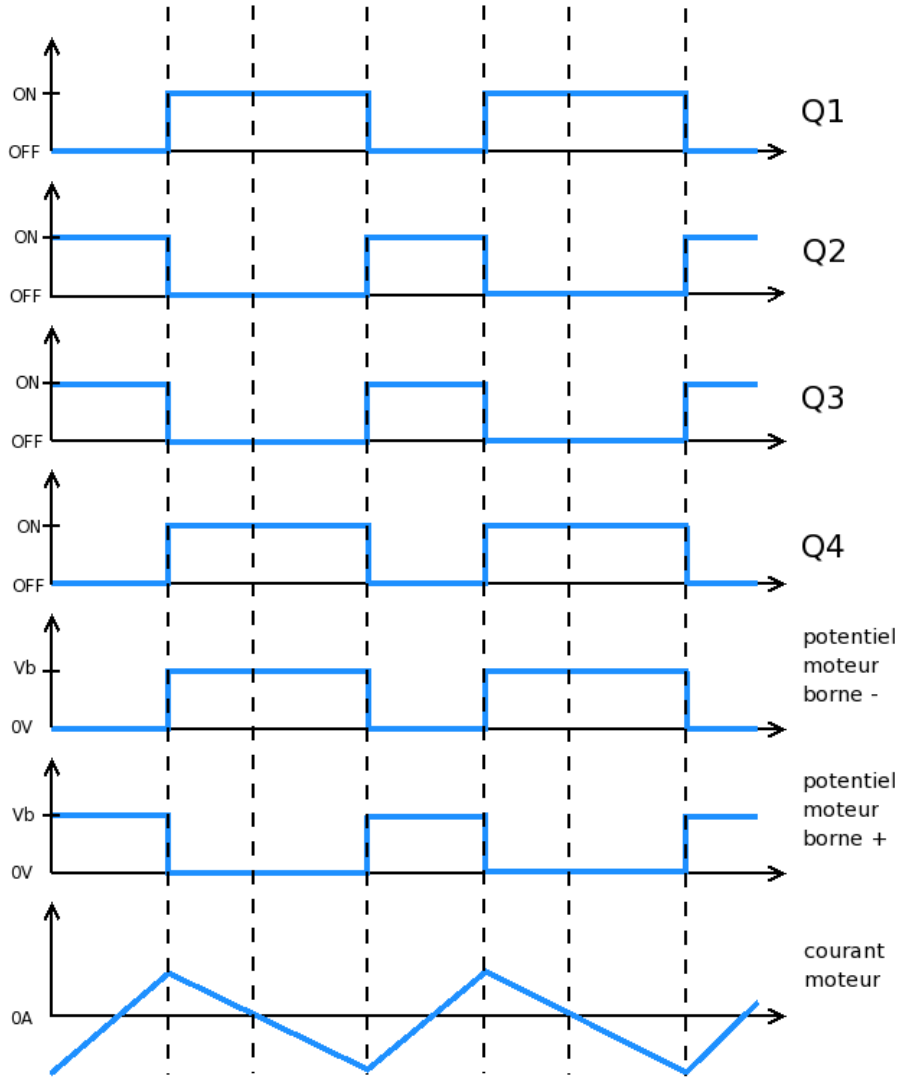

FIG. 4.3 – Diagramme temporel *Locked Anti-phase*

Les temps de commutation des transistors devront donc être très courts afin de limiter les pertes.

Le contrôle en boucle fermée étant un des points clés de notre application, c'est le mode *Locked Anti-phase* qui a été retenu.
Un effort tout particulier sera donc fait dans la suite de notre *design* afin de tenir compte des fréquences de commutation relativement élevées qui seront utilisées.

4.2.2 Design

Maintenant que le schéma général est connu, il convient de sélectionner correctement les composants nécessaires à sa réalisation.

La possibilité d'utiliser un pont en H intégré (i.e. tout en un) à été presque immédiatement écartée. Trois critères ont dicté ce choix :

- La difficulté de trouver des composants intégrés permettant des courants nominaux importants. Ceux-ci sont en général tous limités à 2 voir $3A$.
- Le manque de liberté sur la commande des éléments du pont.
- La dissipation thermique souvent importante de ces composants.

Nous allons donc essayer d'utiliser un maximum d'éléments discrets pour notre design.

Transistors

Comme nous devons travailler à relativement haute fréquence, le choix de la technologie est rapidement résolu : nous allons utiliser des transistors *MOSFET*s.

Le deuxième choix que nous devons faire concerne le type de transistor que nous allons utiliser pour la partie haute du pont en H.
En effet, bien qu'il serait plus facile d'utiliser des *MOSFET*s à canal P pour cette partie, il pourrait être plus intéressant d'utiliser des *MOSFET*s à canal N à la place. Ceux-ci étant, d'une part, moins onéreux, et ayant, d'autre part, de bien meilleures performances ($R_{DS(on)}$[1] plus faible par exemple).
En outre, cela nous permettra d'avoir des caractéristiques de commutation semblables pour les deux parties du pont, ce qui est intéressant pour le mode de commande *Locked Anti-phase* que nous allons utiliser.

L'utilisation de transistors à canal N n'a, par contre, pas que des avantages, puisque nous allons devoir trouver un moyen de fournir, par moment, à la gâchette du transistor, une tension supérieure à la tension d'alimentation.

[1] La résistance drain-source en saturation($R_{DS(on)}$) est la résistance, entre le drain et la source, interne au *MOSFET* qui s'oppose au passage du courant lorsque celui-ci est à l'état passant. Cette résistance est nulle dans un composant idéal.

Il ne nous reste plus qu'à choisir le transistor à canal-N que nous allons utiliser.

Celui-ci devra avoir une $R_{DS(on)}$ la plus faible possible afin de limiter les pertes par dissipation et afin d'autoriser des courants importants.

De plus, les transistors *MOSFET* possèdent une capacité parasite entre leur gâchette et leur source qui doit être chargée ou déchargée à chaque commutation. Le transistor devra donc, aussi, avoir une charge de gâchette la plus faible possible pour limiter le temps de commutation. Ceci est nécessaire, d'une part, pour pouvoir suivre la fréquence, et d'autre part, pour limiter les pertes que nous avons pendant les commutations.

Le transistor que nous avons sélectionné est le *IRF1302* de chez *International Rectifier*.

Les caractéristiques maximums données par le *datasheet* sont (à $25°C$) :

- $V_{(BR)DSS} = 20V$
- $V_{GS} = 20V$
- $V_{GS(th)} = 4V$
- $R_{DS(on)} = 4m\Omega$
- $Q_G = 120nC$

Ces valeurs sont idéales pour notre utilisation. En effet la tension maximale utilisée pour l'alimentation des moteurs sera de 13,8V (batterie en charge), les tensions $V_{(BR)DSS}$ et V_{GS} maximales ne poseront donc pas de problème. Il suffira, en outre, que V_{GS} dépasse 4V pour que le transistor entre en saturation (état passant).

Concernant la dissipation thermique, une $R_{DS(on)}$ de $4m\Omega$ nous donnera $64mW(4A * 4A * 0,004\Omega)$ comme puissance dissipée dans le transistor pour un courant de $4A$. Ou, si on prend le problème en sens inverse, comme le *package TO-220* a une résistance thermique de $62°C/W$ et que la température maximale de fonctionnement du transistor est de $175°C$, la dissipation maximale sera de $1,5W((75°C - 80°C)/62°C/W)$ si la température ambiante ne dépasse pas les $80°C$. Nous pourrons donc commuter au maximum $19A(\sqrt{1,5W/0,004\Omega})$ dans notre pont en H sans utiliser de radiateur, ce qui est largement suffisant pour notre application.

La charge de gâchette devrait, elle aussi, être suffisamment faible pour nous permettre de commuter à des fréquences de l'ordre de la centaine de KHz sans induire de perte trop importante.

Diodes de roue libre

Les *MOSFET* possèdent déjà par construction une diode entre leur drain et leur source.

Ces diodes pourraient être suffisantes, mais par sécurité, nous avons préféré leur adjoindre des diodes externes plus rapides. En effet, dans le mode *Locked Anti-phase*, les diodes de roue libre ne doivent conduire du courant que pendant les phases de commutation des transistors. Les diodes devront réagir le plus rapidement possible pour limiter au maximum les pics de tension inverse aux bornes du moteur pendant ces courtes périodes.

La diode que nous avons sélectionnée est la *MUR420* de chez *ON Semiconductor*.

Elle possède un temps de recouvrement maximum de l'ordre de $25ns$, ce qui est quatre fois plus rapide que la diode interne du *IRF1302* et devrait être largement suffisant pour notre application.

Drivers

Comme nous l'avons vu plus haut, un transistor *MOSFET* possède une capacité parasite en entrée. Lorsque nous voudrons faire commuter ce transistor, nous devrons d'abord charger ou décharger cette capacité avant que la commutation n'aie effectivement lieu.

Le circuit de contrôle devra donc être capable d'imposer un courant suffisamment important pour pouvoir charger ou décharger cette capacité le plus rapidement possible. Les $25mA$ trouvés en entrée/sortie d'un microcontrôleur seront donc très loin d'être suffisants.

Afin de fournir le courant nécessaire de la manière la plus optimale possible, nous allons utiliser un circuit dédié appelé *driver* de *MOSFET*.

Le driver que nous avons sélectionné est le *IR2010* de chez *International Rectifier*.

Ce composant permet de commuter à la fois un des transistors de la partie basse du pont en H mais aussi un des transistors de la partie haute. Il devrait, en outre, être capable d'imposer un courant de $3A$ à la gâchette des transistors et ainsi d'assurer des commutations rapides.

Comme conseillé dans les notes d'applications du constructeur, nous avons placé une faible résistance et une diode en parallèle entre le driver

et la gâchette du transistor. Ceci permet d'augmenter légèrement le temps de commutation de saturation sans modifier le temps de commutation de coupure et donc ainsi d'assurer des temps équivalents pour les deux commutations (ce qui sera utile pour le mode de contrôle *Locked Anti-phase*). La résistance permet, en outre, d'atténuer les oscillations parasites qui pourraient survenir à la gâchette du transistor.

Alimentation de la partie haute

Le dernier problème qu'il nous reste à résoudre dans notre design est la manière dont nous allons fournir une alimentation suffisante aux transistors de la partie haute du pont.

Une première possibilité serait d'utiliser un condensateur dit de *bootstrap* comme proposé dans le *datasheet* du *driver*.

On voit à la figure 4.4 le schéma du montage correspondant.

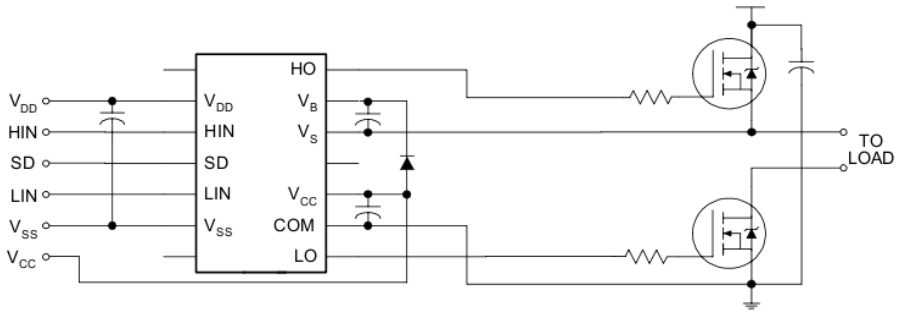

FIG. 4.4 – Montage typique du *IR2010* - Source : *IR2010 Datasheet*

Le fonctionnement est relativement simple :

- Lorsque le transistor haut est coupé, V_S est à la masse et le condensateur de *bootstrap* (entre V_B et V_S) est chargé à la tension d'alimentation V_{CC} au travers de la diode de *bootstrap*.

- Lorsque le transistor passe en saturation, nous avons une tension V_{CC} aux bornes du condensateur de *bootstrap* et donc une tension V_S+V_{CC} en V_B, ce qui permet de maintenir le transistor ouvert.

Ce montage, bien qu'élégant, est loin d'être parfait pour notre utilisation. En effet, le condensateur de *bootstrap* se décharge au cours du temps et ne permet pas de garder le transistor haut en saturation indéfiniment. En outre, le temps passé en mode coupé doit être suffisamment long pour

permettre la charge du condensateur.

Ce montage nous impose donc, en pratique, des contraintes sur le rapport cyclique de notre *PWM*. Pour un pont en H complet piloté en *Locked Anti-phase*, on ne pourra, par exemple, pas utiliser un rapport cyclique inférieur à 10-15% et supérieur à 85-90%. Ce qui est une sérieuse limitation.

Une deuxième possibilité, qui est celle que nous avons testé en pratique, est de se servir un convertisseur DC-DC isolé. C'est la possibilité la plus coûteuse, mais c'est certainement la plus simple et la plus performante.
Le convertisseur que nous avons sélectionné est le *TEL2-1212* de chez *Traco Power*, il prend en entrée une tension comprise entre 9 et $18V$ et donne en sortie une tension de $12V$.

Une troisième possibilité, serait d'employer une pompe de charge, celle-ci fonctionne de manière similaire au condensateur de *bootstrap*, mais utilise un oscillateur externe pour maintenir la tension aux bornes du condensateur. Ce montage est nettement moins onéreux que le précédent mais complexifie le schéma. Cette possibilité n'a pas été mise en pratique, le montage actuel s'étant montré suffisamment satisfaisant. Il pourrait, par contre, être utile d'en étudier les performances dans une mise à jour éventuelle, notamment pour la réduction substantielle du coût qu'elle pourrait entraîner.

Schéma

On trouvera à la figure 4.5 le schéma final de notre étage de puissance (Pont en H).

Les pins MOT_0V et MOT_12V représentent l'alimentation du moteur, les pins MOT1 et MOT2 représentent elles, les bornes du moteur.

On remarquera, par exemple, les *drivers* U1 et U2, ainsi que les alimentations isolées U3 et U4 utilisées pour alimenter la partie haute du pont.

4.3 Logique

Une fois le design de la partie puissance réalisé, il reste à élaborer celui de la partie logique qui va être chargée de commander le pont H. Ceci passera par la sélection d'un microcontrôleur et d'un certain nombre de capteurs.

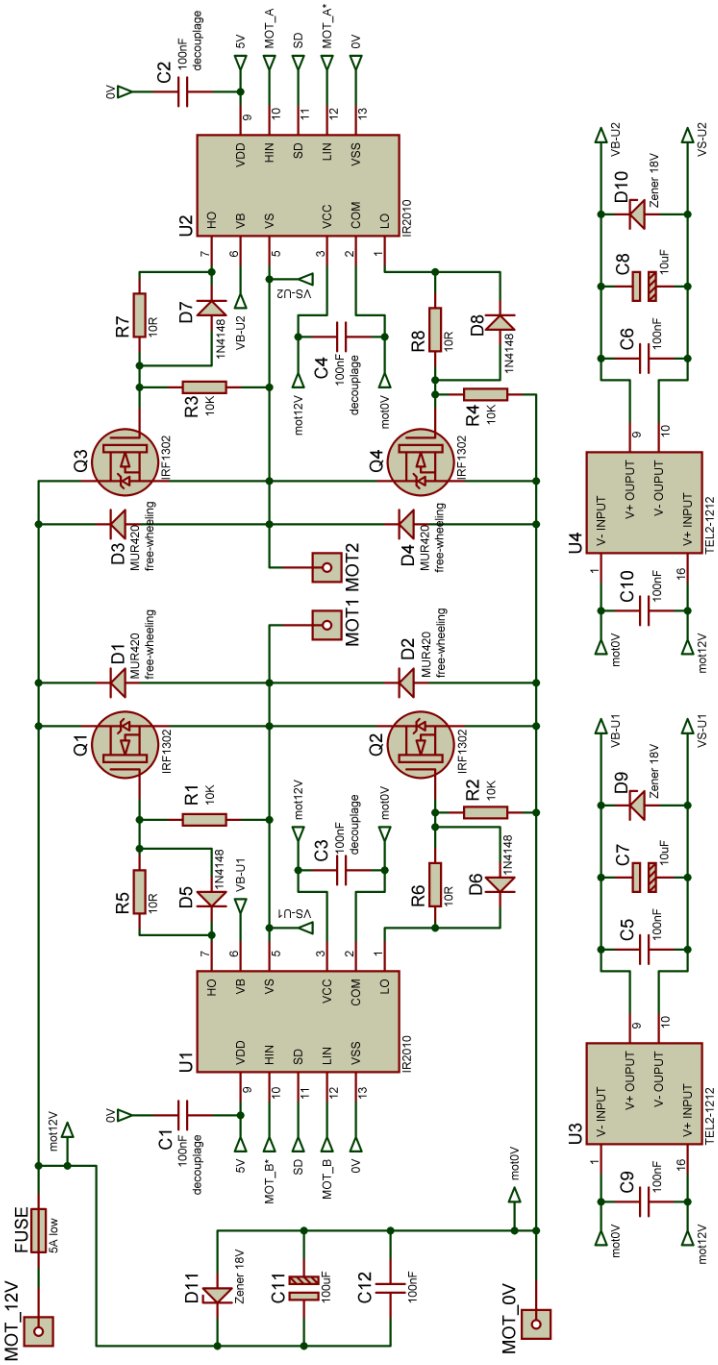

FIG. 4.5 – Schéma du Pont en H utilisé

La partie logique doit être en mesure d'effectuer un certain nombre de tâches :

- Générer des signaux PWM de fréquence de l'ordre d'une centaine de KHz avec une résolution acceptable du rapport cyclique.
- Gérer un signal d'arrêt d'urgence.
- Effectuer des conversions analogique-digital pour la mesure du courant consommé et de la tension de l'alimentation.
- Récupérer les informations d'un encodeur de quadrature.
- Effectuer une régulation numérique de type PID.
- Gérer le protocole I^2C.

4.3.1 Microcontrôleur

Pour effectuer toutes les tâches nécessaires, notre choix s'est porté sur un microcontrôleur de chez *Microchip* de la famille *18F* : le *18F2431*.

Les caractéristiques de ce microcontrôleur répondent à toutes nos attentes. Nous citerons par exemple :

- un module PWM de plusieurs canaux avec sorties complémentaires, et une entrée hardware de protection qui pourra être utilisée pour l'arrêt d'urgence.
- un module pour la conversion analogique-digital sur 10bits de cinq canaux, pouvant effectuer plusieurs conversions simultanées.
- une interface pour un encodeur de quadrature.
- un module I^2C esclave.
- de nombreux *timers*.
- une gestion des interruptions.
- 8Kmots de mémoire programme, 256bytes de ROM et 768bytes de RAM.

– 10*MIPS* (*Million Instructions Per Second*) si il est utilisé avec un quartz de 10Mhz et une *PLL* 4X.

– 24 entrées/sorties.

La plupart des tâches pourront donc être gérées directement par ce composant, et nous n'aurons pas besoin de beaucoup plus qu'une alimentation stabilisée de $5V$ et un quartz de 10Mhz pour terminer notre circuit.

4.3.2 Capteurs

Encodeur de quadrature

Nous allons faire ici une brève description du capteur. L'encodeur de quadrature monté sur nos moteurs est un *HEDL-5540* de 500 pas. Celui-ci permet d'obtenir une information sur la position du moteur de manière incrémentale. En effet, si on connaît le nombre d'impulsions par tour, il suffit de compter le nombre d'impulsions données par le capteur pour connaître la position relative du moteur par rapport à sa position de départ.

L'encodeur est dit de quadrature, parce qu'il possède en fait deux sorties A et B déphasées de 90°. Nous avons quatre états différents pour les deux signaux, qui s'enchaînent dans un certain ordre suivant le sens de rotation. On peut décoder ces deux signaux pour générer les impulsions et pour connaître le sens de rotation du moteur. La résolution lors du décodage sera quatre fois supérieure au nombre de pas du capteur (pour chaque pas nous avons quatre transitions d'état), et on pourra espérer pour le *HEDL-5540* une précision au 1/2000ème de tour.

On observe dans les tableaux ci-dessous les transitions entre les différents états des signaux A et B pour une rotation en sens normal et en sens inverse.

Sens normal de rotation		
État	A	B
1	0	0
2	0	1
3	1	1
4	1	0

Sens inverse de rotation		
État	A	B
1	1	0
2	1	1
3	0	1
4	0	0

Le décodage est relativement simple et peut être réalisé par une petite *CPLD*, mais comme nous venons de le voir, nous allons utiliser directement

un module dédié du microcontrôleur pour cette tâche.

Remarquons que l'encodeur de quadrature utilisé sur nos moteurs est muni d'un *line driver* différentiel afin de garantir une meilleure immunité aux parasites (qui peuvent être nombreux à proximité du moteur) pendant la transmission. Nous devrons donc utiliser sur notre carte un *line receiver* correspondant (le *26LS32*) afin de remettre le signal en forme, avant sa connexion au microcontrôleur.

Courant consommé

Afin d'être en mesure de détecter une situation dangereuse pour le moteur, comme un blocage de celui-ci, nous allons faire une mesure du courant consommé.

Pour ne pas modifier le schéma du pont en H, et afin de limiter au maximum les perturbations, nous avons décidé d'utiliser un capteur de courant à effet Hall. Celui-ci nous donne, en sortie, une tension proportionnelle au courant le traversant.

Le capteur utilisé est le *ACS754-050* de chez *Allegro*. Il permet de mesurer un courant dans les deux sens jusque $50A$, peut être alimenté en $5V$, et fourni en sortie une tension, centrée en $V_{CC}/2$, proportionnelle au courant. La sensibilité du capteur est de l'ordre de $40mV/A$.

La sortie du capteur est ensuite connectée à l'entrée d'un convertisseur analogique-digital du microcontrôleur. L'information fournie par le capteur sera largement suffisante pour nous permettre de vérifier le courant consommé par le moteur.

Pour éviter les parasites, nous avons en plus placé en sortie du capteur un filtre RC passe-bas avec une fréquence de coupure légèrement inférieure à la bande passante du capteur.

Alimentation du moteur

Avec notre pont en H, la tension aux bornes du moteur est proportionnelle au rapport cyclique de la *PWM*. Cela nous permet de fixer un pourcentage de la tension d'alimentation aux bornes du moteur.
Le problème est que pour fixer exactement une tension, nous devons, avant tout, connaître la tension d'alimentation. Celle-ci pouvant fluctuer, nous avons décidé de la mesurer. Si nous avions fixé cette valeur sans la mesurer,

la réponse de notre contrôleur *PID* aurait été fonction de la charge de la batterie, ce qui aurait été gênant.

Pour effectuer notre mesure, nous allons utiliser un simple diviseur potentiomètrique en entrée d'un convertisseur analogique-digital du microcontrôleur.

4.4 Schéma complet

Le schéma complet, la liste des composants utilisés, ainsi que le *layout* de la carte électronique réalisée pourront être trouvés à l'annexe A.

Une version électronique du schéma et du routage pourront aussi être trouvés sur le cd-rom joint à ce mémoire dans le répertoire "LOCO/schema/".

4.5 Résultats

Une fois la carte électronique réalisée physiquement (on peut trouver des photographies de la carte à la figure 4.6 et à la figure 4.7), nous allons tenter d'évaluer ses performances.

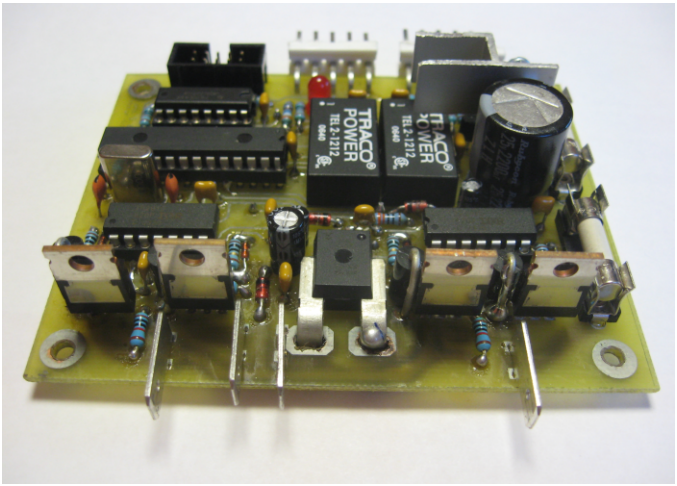

FIG. 4.6 – Carte électronique - face avant

Remarque : nous ne nous occuperons uniquement ici que des performances intrinsèques du pont en H. Les performances du contrôleur *PID* seront évaluées au chapitre suivant.

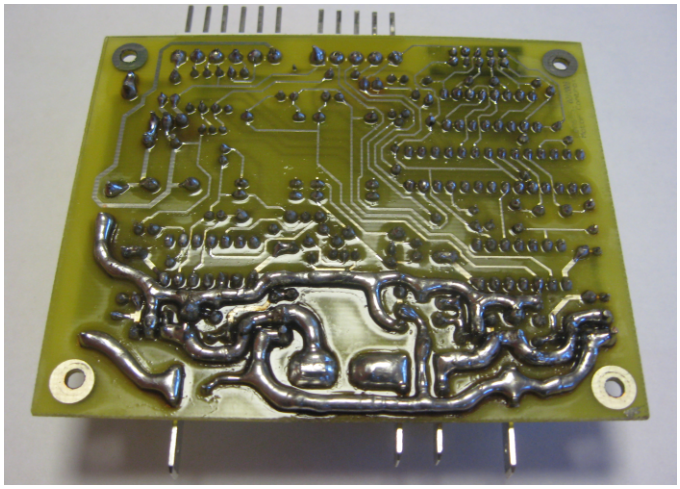

FIG. 4.7 – Carte électronique - face arrière

La fréquence choisie pour la *PWM* qui va contrôler le pont en H est de 76kHz et nous permet d'avoir une résolution de 9 bits pour le rapport cyclique ce qui devrait être suffisant.

La figure 4.8 montre la tension aux bornes du moteur, le signal est bien régulier et est conforme à nos attentes. Le moteur est alimenté dans un sens et puis dans l'autre en alternance comme le prévoit le mode *Locked Anti-phase*.

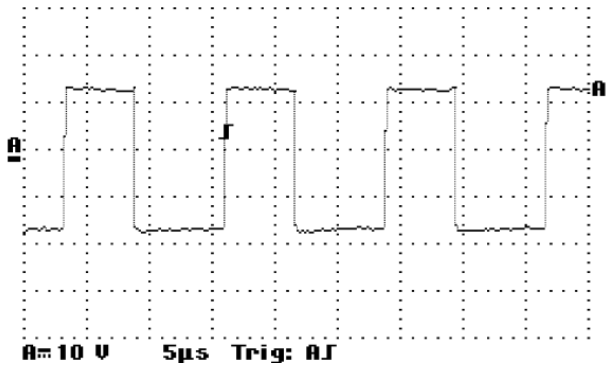

FIG. 4.8 – Tension aux bornes du moteur

En outre, aucun échauffement sensible des composants n'est observé (transistors, diodes de roue libre) et ce, même après un très long temps de fonctionnement.

Le système se comporte donc de manière très satisfaisante, à l'exception

du moteur qui lui chauffe fortement.

Si nous analysons le courant traversant le moteur à la figure 4.9, nous observons que celui-ci varie fortement entre chaque commutation. Malgré la fréquence élevée de notre *PWM*, l'inductance du moteur($0.085mH$) ne semble pas suffisante pour lisser correctement le courant. Nous avons donc décidé d'adoucir les pentes du courant en ajoutant une inductance de $330\mu H$ supplémentaire en série avec le moteur.

Remarque : le courant dans le moteur est obtenu en mesurant la tension de sortie de la sonde de Hall présente sur la carte.

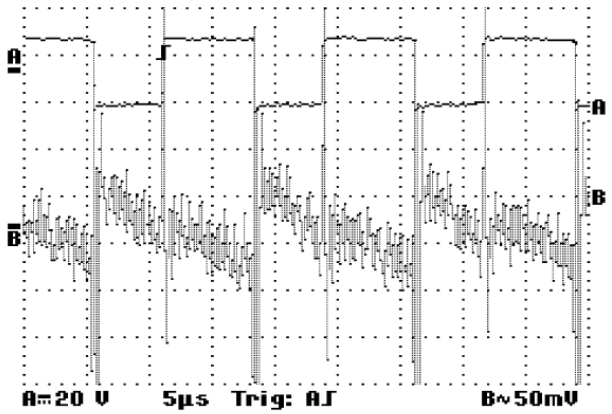

FIG. 4.9 – Analyse du courant - Tension aux bornes du moteur en A - Courant dans le moteur en B ($40mv/A$)

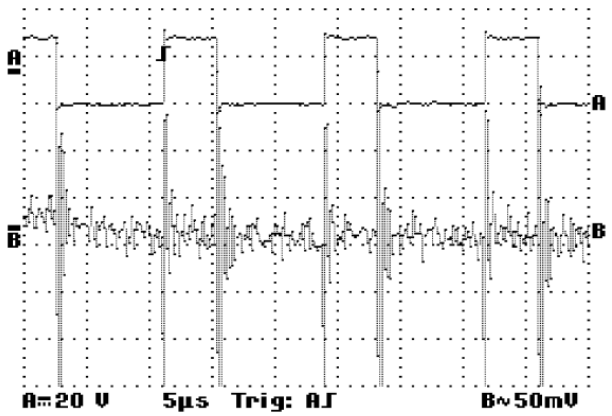

FIG. 4.10 – Analyse du courant avec une inductance supplémentaire - Tension aux bornes du moteur en A - Courant dans le moteur en B (40mv/A)

Nous remarquons, après l'ajout de cette inductance, à la figure 4.10 que le courant dans le moteur est maintenant beaucoup mieux lissé.
L'impact direct de cet ajout est que le moteur ne chauffe maintenant plus du tout. Notre pont en H se comporte parfaitement et ses performances sont sans commune mesure avec les systèmes utilisés lors des participations précédentes au concours de robotique.

Pour être exhaustif, nous allons aussi regarder de plus près la tension V_{gs} des transistors lors des commutations, afin d'évaluer leur durée.

Nous verrons dans les quatre figures suivantes, l'observation de cette tension pour dans l'ordre :
- une mise en saturation d'un transistor de la partie basse du pont.
- une mise en coupure d'un transistor de la partie basse du pont.
- une mise en saturation d'un transistor de la partie haute du pont.
- une mise en coupure d'un transistor de la haute basse du pont.

Toutes les commutations sont inférieures à 100ns (fin du palier), ce qui est conforme à nos attentes pour ce transistor.

Nous remarquerons aussi que les durées des commutations pour les mises en saturation et en coupure sont équivalentes, comme nous l'avions espéré. Il n'a donc pas été nécessaire d'ajouter un *dead-time* supplémentaire entre les commutations des deux transistors d'une branche du pont. En effet, celles-ci se coordonnent efficacement et ne provoquent donc pas de court-circuit.

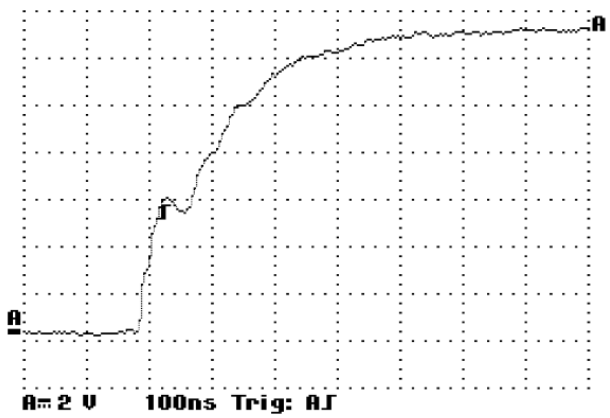

FIG. 4.11 – V_{gs} pour la mise en saturation d'un transistor bas

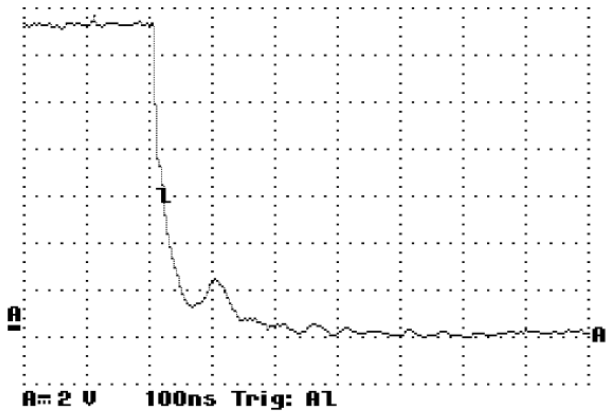

FIG. 4.12 – V_{gs} pour la mise en coupure d'un transistor bas

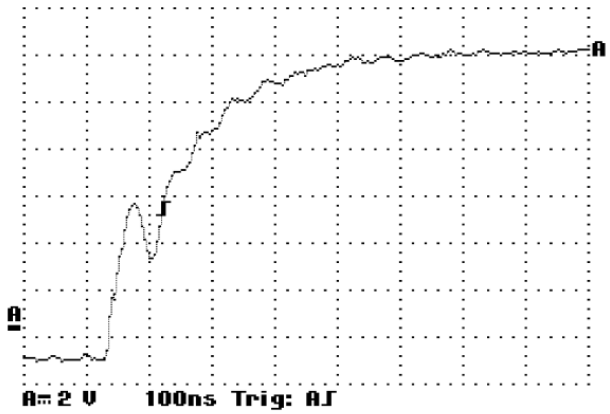

FIG. 4.13 – V_{gs} pour la mise en saturation d'un transistor haut

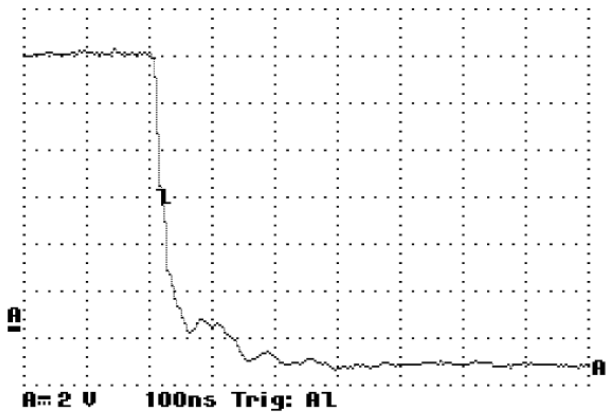

FIG. 4.14 – V_{gs} pour la mise en coupure d'un transistor haut

Chapitre 5

Aspects logiciels

Sommaire

5.1 Microcontrôleur

5.1.1 Définition du problème

Une fois la carte électronique réalisée, il convient d'écrire le programme du microcontrôleur.

Celui-ci devra répondre à plusieurs attentes :

- Pouvoir commander le moteur en boucle ouverte.
- Pouvoir contrôler le moteur en vitesse.
- Pouvoir contrôler le moteur en position avec éventuellement des rampes d'accélération.

- Maintenir une mesure odométrique de la distance angulaire parcourue par la roue motrice.

- Pouvoir recevoir des ordres et répondre à des requêtes via le bus I^2C.

- Arrêter le moteur en cas de situation dangereuse (blocage par exemple).

- Exécuter les ordres de locomotion à des instants spécifiés pour permettre la synchronisation de plusieurs cartes.

Afin de faciliter l'implémentation de la régulation numérique, le programme du microcontrôleur a été écrit en *C* et a été compilé grâce au compilateur *C18* pour *PIC 18F* de chez *Microchip*.
Une implémentation directe en assembleur, aurait complexifié inutilement la compréhension du programme tout en n'apportant pas un gain de temps d'exécution suffisamment appréciable.

Le programme du microcontrôleur représente environ 2700 lignes de code *C*, celui-ci pourra être trouvé sur le cd-rom joint à ce mémoire dans le répertoire "LOCO/programme/".

5.1.2 Implémentation du contrôle

Contraintes

Pour la réalisation de ce programme, nous allons devoir faire face à un certain nombre de contraintes.

La boucle de régulation numérique doit être exécutée à intervalles réguliers avec une fréquence importante (plusieurs centaines de Hz). Le temps d'exécution de celle-ci devra donc être le plus court possible.

Afin d'effectuer les calculs numériques des boucles de régulation le plus rapidement possible, plusieurs astuces ont été utilisées :

- Toutes les variables du programme sont soit globales, soit *static*. Cela permet d'y accéder plus rapidement. (Celles-ci étant dans la *stack* et pas sur la *pile*). Nous avons remarqué que cette astuce a une énorme influence sur le temps d'exécution, ceci est dû à l'architecture particulière du microcontrôleur. Remarque : aucune des fonctions n'a besoin d'être ré-entrante, avoir des variables globales ou *static* ne pose donc aucun problème.

- On utilise le moins possible d'appels de fonction, on préférera l'utilisation de *macros* qui ne nécessitent pas de changement de contexte sur la pile.

- On utilise du calcul en virgule flottante seulement lorsque c'est absolument nécessaire, tous les autres calculs étant effectués en nombres entiers.

- On évite le plus possible les divisions très coûteuses en temps, et on les remplace par des multiplications lorsque c'est possible. On remplacera par exemple une division par 5 par une multiplication par 0.2.

- Le microcontrôleur ayant une architecture de 8 bits, on veille à choisir correctement les types des variables. On préférera utiliser par exemple le type particulier *short long* de 24 bits plutôt que le type *long* de 32bits, si 24bits sont suffisants pour stocker notre variable.

- On essaye d'utiliser au maximum des opérations binaires (décalage, ...) plutôt que des opérations arithmétiques lorsque c'est possible.

Remarquons qu'il a aussi été tenté de remplacer les calculs en virgule flottante par des calculs en virgule fixe. Mais cette modification n'a pas apporté de gain substantiel suffisant que pour être retenue.

Une autre contrainte, est que d'autres opérations plus prioritaires peuvent survenir pendant la régulation (une communication I^2C par exemple). La boucle doit donc pouvoir être interrompue.

Notre boucle sera donc appelée régulièrement par une interruption *timer* et seule la lecture du capteur de position sera effectuée dans celle-ci. Toutes les autres opérations de la boucle seront effectuées dans le corps du programme.

Il convient toutefois de faire attention à la modification des variables pouvant être lues dans d'autres routines d'interruption. Celles-ci n'étant pas toujours sur 8 bits, toutes les modifications s'y rapportant ne seront pas atomiques. Il conviendra donc de désactiver temporairement les interruptions à chaque fois qu'elles sont modifiées, afin d'éviter des corruptions.

Contrôle en vitesse

La régulation en vitesse consiste en l'implémentation de la formule

$$\begin{aligned} u(t) =& k_p(b_{sp}v_{ref}(t) - v_{obs}(t)) + k_i \int_0^t v_{ref}(\tau) - v_{obs}(\tau)d\tau \\ &+ k_d \left(b_{sd}\frac{dv_{ref}(t)}{dt} - \frac{dv_{obs}(t)}{dt} \right) \end{aligned} \tag{5.1}$$

avec v_{ref} la vitesse désirée et v_{obs} la vitesse mesurée du moteur.

La discrétisation de l'intégrale est faite par approximations trapézoïdales, et celle de la dérivée par différences finies.

Pour l'intégrale :

$$\begin{aligned} CI(n+1) =& CI(n) \\ &+ k_i \frac{(v_{ref}(n+1) - v_{obs}(n+1)) + (v_{ref}(n) - v_{obs}(n))}{2F} \end{aligned} \tag{5.2}$$

Pour la dérivée :

$$CD(n+1) = Fk_d \left(b_{sd} \left(v_{ref}(n+1) - v_{ref}(n) \right) - \left(v_{obs}(n+1) - v_{obs}(n) \right) \right) \tag{5.3}$$

où F est la fréquence du contrôleur.

Pour supprimer l'*integral windup*, nous utilisons simplement la méthode suivante : nous bloquons l'intégration lorsque l'effet de celle-ci, via le signal de contrôle, cherche encore à augmenter (resp. diminuer) la sortie du système ayant atteint son maximum (resp. minimum). Le maximum étant la génération d'un signal PWM de 100% de rapport cyclique, le minimum un signal PWM de 0% de rapport cyclique.

Afin de diminuer le bruit présent sur la mesure de la vitesse, nous lui avons appliqué un léger filtre à réponse impulsionnelle infinie.

$$vit(t+1) = \frac{vit_{obs}(t+1) + vit(t)}{2} \tag{5.4}$$

Contrôle en position

La régulation en position consiste en l'implémentation de la formule

$$\begin{aligned} u(t) =& k_p(p_{ref}(t) - p_{obs}(t)) + k_i \int_0^t p_{ref}(\tau) - p_{obs}(\tau)d\tau \\ &+ k_d \left(b_{sd}\frac{dp_{ref}(t)}{dt} - \frac{dp_{obs}(t)}{dt} \right) \end{aligned} \tag{5.5}$$

avec p_{ref} la position désirée et p_{obs} la position mesurée du moteur.

La discrétisation de l'intégrale et de la dérivée, ainsi que la suppression de l'*integral windup*, sont implémentées de la même manière que pour le contrôle en vitesse.

La position du moteur étant cyclique par nature, il convient de faire attention lors du calcul des erreurs, nous veillerons à toujours faire en sorte que celles-ci soient inférieures à un demi-tour, grâce, par exemple, à un calcul de *modulo*. Sans quoi, le contrôleur pourrait avoir un comportement imprévu.

Pour la gestion des rampes d'accélération, nous procédons de la manière suivante : lorsqu'une nouvelle position de référence est donnée, celle-ci n'est, en fait, répercutée que progressivement au contrôleur *PID* de manière à respecter les rampes imposées.

Pour calculer les rampes, nous avons besoin de deux paramètres : une accélération maximale et une vitesse maximale.

La nouvelle valeur de position à envoyer au contrôleur à chaque étape est calculée comme suit :

$$p_{ref}(n+1) = p_{ref}(n) + vit(n+1) \tag{5.6}$$

où

$$vit(n+1) = \min(vit_a(n+1), vit_b(n+1), vit\_max) \tag{5.7}$$

avec

$$vit_a(n+1) = vit(n) + acc\_max \tag{5.8}$$

et

$$vit_b(n+1) = \frac{\sqrt{(8p_{dist}(n+1) + acc\_max)acc\_max} - acc\_max}{2} \tag{5.9}$$

avec p_{dist} la différence de position restant encore à parcourir avant d'atteindre la position finale de référence.

Nous remarquons que vit_b représente, en fait, la vitesse maximum possible à la distance p_{dist} de l'objectif, et qui permet d'atteindre la position finale de référence, avec une vitesse nulle, tout en respectant la décélération maximale.

La racine carrée est calculée pour des nombres entiers grâce à une approximation de Newton.

Nous ne calculerons la racine carrée que si c'est vraiment nécessaire (celle-ci étant gourmande en ressources).

En effet, si

$$p_{dist}(n+1) \geq \frac{vit\_max\left(\frac{vit\_max}{acc\_max}+1\right)}{2} \tag{5.10}$$

On sera toujours capable de respecter la pente de décélération, même à vitesse maximale, et il n'est pas nécessaire de calculer vit_b.

Dans notre implémentation finale, afin que les calculs des rampes (principalement le calcul de la racine carrée) n'influencent pas l'exécution de la boucle de régulation, ceux-ci ont été répartits sur plusieurs exécutions de la boucle.

5.1.3 Communications

Une liste des ordres et des requêtes I^2C pourra être trouvée à l'annexe B.

Notre gestion de l'I^2C est relativement classique et est basée sur le protocole [Boi05].
L'implémentation est générique et pourra être réutilisée pour d'autres cartes périphériques du robot. Un effort particulier a été donné afin de minimiser la durée des routines d'interruption.

Nous remarquerons aussi, qu'un certain nombre de fonctionnalités liées aux communications ont été ajoutées à la carte afin d'améliorer sa robustesse.

Nous citerons par exemple la sauvegarde des paramètres de régulation dans la mémoire *ROM* du *PIC* afin de pouvoir les récupérer après un redémarrage imprévu de la carte. Ceci nous permet de reprendre un fonctionnement normal, même si l'ordinateur embarqué ne nous renvoie pas les paramètres.

Nous citerons aussi, la présence d'un chien de garde sur la réception des ordres de locomotion. Si nous ne recevons pas d'ordre pendant plus de deux secondes, nous considérons que l'ordinateur ne répond plus et nous arrêtons immédiatement le moteur. Cela permet, par exemple, d'éviter les accidents en cas de plantage du programme principal du robot.

5.1.4 Gestion des erreurs

Une *LED* placée sur la carte électronique permet d'avoir un *monitoring* des erreurs pouvant survenir. Une tâche spécifique à la gestion des erreurs est appelée régulièrement et permet de faire clignoter la *LED* à des fréquences différentes selon le type d'erreur rencontrée.

C'est aussi à cette tâche que revient la responsabilité de mesurer le courant consommé par le moteur et de détecter des sur-courants. Nous procédons pour ceci de la manière suivante, nous calculons une moyenne de la valeur absolue du courant mesuré, lorsque celle-ci dépasse un certain seuil spécifié, le moteur est mis en roue libre pendant une demi-seconde et puis essaie de reprendre son activité normale. Ceci nous permet de protéger le moteur en cas de blocage par exemple.

5.2 Paramètrage du contrôle

5.2.1 Définition du problème

Nous possédons maintenant une carte électronique permettant a priori le contrôle en vitesse et en position d'un moteur à courant continu. Afin d'en tester les performances, et surtout, afin de choisir de manière optimale les paramètres de la boucle de contrôle, un logiciel, permettant d'obtenir les courbes de vitesse ou de position du moteur en fonction de courbes de référence, serait fort utile. Ce logiciel devrait permettre la modification des paramètres de contrôle ainsi que l'observation en temps réel de leur influence sur la réponse du système.

5.2.2 Driver temps réel

Afin d'utiliser la carte électronique développée dans des conditions similaires à celles du robot final, il était important d'adresser celle-ci cinquante fois par seconde tout en utilisant la même carte électronique $RS232 \leftrightarrow I^2C$ (HOST).

Le programme se voulant général et utilisable sur n'importe quel ordinateur, il était impensable d'utiliser un noyau temps réel pour effectuer la tâche d'adressage. Un driver spécifique s'approchant au plus des spécifications temps-réel à donc été développé.

Plusieurs astuces ont été utilisées afin d'atteindre une précision d'adressage de l'ordre de cinq micro-secondes :

1. Le driver profite des facilités offertes par l'ordonnanceur du noyau *Linux*. Le processus utilise un ordonnancement *FIFO* avec une priorité maximale. Cela signifie que dès que le processus est prêt, tout autre processus de priorité inférieure est interrompu. Cela signifie aussi que ce processus ne peut jamais être préempté. La seule contrainte que ce mécanisme impose est, que le processus possède des droits *root* afin d'être en mesure de modifier sa politique d'ordonnancement.

2. Une fonction *sleep* spécifique a été développée afin d'atteindre une précision supérieure sur le moment où la tâche reprend effectivement son exécution. Cette fonction effectue, en fait, un *sleep* standard d'une milli-seconde plus court que la durée totale puis une boucle de type *busy-waiting* sur la durée restante. La priorité du processus combinée à cette fonction, assure en général la reprise de l'exécution normale à moins de cinq micro-secondes près sur les ordinateurs où elle a été testée.

3. Toutes les tâches de mise en forme du protocole qui dépendent du nombre et de la taille des messages à envoyer au cours du cycle suivant sont effectuées une demi milli-seconde avant l'envoi effectif des données sur le port série. De cette manière, seul le moment précis de l'envoi des données est assuré au moyen du *sleep* défini ci-dessus.

Toutes ces petites astuces ont permis d'atteindre une précision d'adressage quasi similaire à celle du robot final. Mais la difficulté avec laquelle celle-ci a été acquise, démontre que le choix d'une architecture temps-réel sur un système autonome complexe n'est pas vain.

La communication entre le driver et le processus utilisateur se fait au moyen de files de messages *IPC*, ce qui permet au processus utilisateur d'être totalement indépendant du driver. C'est au driver que revient la charge de la synchronisation. Pour se faire, le driver envoie un message de synchronisation déterminé à chaque cycle afin de spécifier qu'il est prêt à recevoir les ordres et/ou requêtes suivants.

L'implémentation se veut aussi générale que possible et pourra être réutilisée pour adresser n'importe quelle carte respectant le protocole.

5.2.3 Client *Java*

La gestion de l'adressage étant gérée par un driver spécifique, il reste à développer l'interface utilisateur.

Celle-ci a été réalisée en *Java*, ce choix à été principalement guidé par la facilité de développement d'interfaces graphiques dans ce langage, mais aussi grâce à la disponibilité de la libraire *JFreeChart*[1] pour la gestion des diagrammes.

Une des particularités de *Java* est qu'il tourne dans une machine virtuelle. Ce qui ne permet pas l'utilisation des files de messages *IPC*. Il a donc fallu développer une libraire *JNI* (*Java native interface*) pour permettre l'utilisation de celles-ci.

N.B. : D'autres choix pour la communication entre un processus *Java* et un processus C auraient pu être envisagés comme par exemple l'utilisation de *sockets TCP/UDP*. Mais la combinaison files *IPC* - *JNI* nous a semblé relativement intéressante.

Le logiciel final permet d'adresser n'importe quelle carte moteur et permet l'affichage en temps réel de la tension de l'alimentation, du courant consommé, de l'estimation de la tension aux bornes du moteur ainsi que de la vitesse ou la position du moteur selon le mode choisi. Le logiciel dispose, en outre, d'un générateur de signal de référence afin de tester les performances du contrôle.

L'ensemble driver, *JNI* et interface *Java* représente environ 6000 lignes de code.

Le programme complet pourra être trouvé sur le cd-rom joint à ce mémoire dans le répertoire "locoParams/".

[1] *JFreeChart* est une librairie *Java* sous licence *LGPL* qui permet l'inclusion de diagrammes divers dans des applications graphiques.
Source : `http://www.jfree.org/jfreechart/`

FIG. 5.1 – Vue d'ensemble de l'interface *Java* pour le réglage des paramètres de contrôle

5.2.4 Méthodes de paramétrage

Modélisation

La première méthode que l'on peut utiliser est basée sur la modélisation du système que l'on veut contrôler.

Nous voyons dans [AM04] et dans [Sep07] que si la dynamique du système peut être approximée par un système du premier ordre avec une fonction de transfert :

$$G(s) = \frac{K}{\tau + s} \tag{5.11}$$

Il est possible de déterminer les paramètres d'une boucle PI en fonction des caractéristiques de réponse du système que nous voulons obtenir.

En effet la fonction de transfert référence → sortie en boucle fermée (PI) du système final sera donnée par :

$$G_{y,r}(s) = \frac{K(k_p b_{sp} s + k_i)}{\tau s^2 + (Kk_p + 1)\, s + Kk_i} \tag{5.12}$$

et nous aurons deux paramètres (k_p et k_i) pour choisir la position des pôles du système en boucle fermée.

De la même manière, pour un système du second ordre avec une fonction de transfert :

$$G(s) = \frac{b}{s^2 + a_1 s + a_2} \tag{5.13}$$

La fonction de transfert référence → sortie en boucle fermée (PID) du système final sera donnée par :

$$G_{y,r}(s) = \frac{b(k_d b_{sd} s^2 + k_p b_{sp} s + k_i)}{s^3 + (a_1 + bk_d)s^2 + (a_2 + bk_p)s + bk_i} \tag{5.14}$$

et nous aurons trois paramètres (k_p, k_i et k_d) pour choisir la position des pôles du système en boucle fermée.

La modélisation du moteur à courant continu pour le contrôle en vitesse PI et le contrôle en position PID est donnée à l'annexe C.

Ziegler-Nichols

Une autre méthode bien connue est celle énoncée par Ziegler et Nichols dans [ZN42].

Elle se base sur les propriétés de réponse fréquentielle du système étudié.

Nous utilisons une boucle de rétroaction avec gain proportionnel uniquement, et nous augmentons celui-ci jusqu'à atteindre la limite de stabilité du système. Nous notons alors Kc le gain critique du système et Pc la période de l'oscillation observée.

Les paramètres de contrôle sont ensuite donnés par la table suivante :

P	$0.5Kc$	–	–
PI	$0.45Kc$	$1.2Kp/Pc$	–
PID	$0.6Kc$	$2Kp/Pc$	$KpPc/8$

Nous pourrions même envisager d'automatiser la méthode dans notre programme. En effet, beaucoup de méthodes de paramétrage automatique sont basées sur Ziegler-Nichols, et la mise en oeuvre ne devrait pas être trop compliquée. Cette possibilité n'a, par contre, pas été testée.

Apprentissage

Nous pourrions aussi envisager de trouver les paramètres de contrôle par apprentissage.

En essayant de minimiser la somme des carrés des distances entre le signal de référence et la sortie observée pour un échelon de durée n par

exemple. Pour éviter le plus possible les dépassements, un poids important au sein de la somme pourrait être donné à ceux-ci.

Pour résumer, nous cherchons l'ensemble des paramètres $\boldsymbol{p}$ tels que :

$$arg\min_{\boldsymbol{p}} \sum_{i=0}^{n} \left((\alpha - 1) \frac{sign\left(r(i) - y(i|\boldsymbol{p})\right)}{-2} + 1 \right) \left(r(i) - y(i|\boldsymbol{p})\right)^2, \alpha > 1 \tag{5.15}$$

Ce problème peut, par exemple, être résolu par l'algorithme de Levenberg-Marquardt (voir Annexe D).

La seule contrainte est que cette méthode ne permet de trouver qu'un minimum local de la fonction. Il sera peut-être donc nécessaire de fournir un vecteur $\boldsymbol{p}$ initial relativement proche de la solution recherchée. Celui-ci peut, par exemple, être fourni par la méthode de Ziegler-Nichols automatisée.

Cette approche n'a malheureusement pas pu être testée faute de temps, mais il pourrait être intéressant d'observer ses performances ainsi que son réel intérêt pratique.

5.3 Résultats

Nous allons donner ici quelques résultats de contrôle en vitesse et en position, afin d'évaluer les performances du système construit. La fréquence de la boucle de régulation sur la carte électronique est de 500Hz et la fréquence d'envoi des ordres à la carte est de 50Hz.

On observera sur les figures suivantes, des exemples de régulation en vitesse, en position et en position avec rampes d'accélération. Les paramètres de la boucle *PID* ayant étant réglés ou non.

La régulation est tout à fait conforme à nos attentes, et on peut raisonnablement estimer que les différentes réalisations fonctionnent correctement.

Remarque : un délai d'un cycle (20ms) peut être observé sur les différentes réponses, ceci est tout à fait normal et est inhérent au protocole utilisé (voir section 2.3).

FIG. 5.2 – Résultats pour un contrôle en vitesse - boucle non paramétrée

FIG. 5.3 – Résultats pour un contrôle en vitesse - boucle paramétrée

FIG. 5.4 – Résultats pour un contrôle en position - boucle non paramétrée

FIG. 5.5 – Résultats pour un contrôle en position - boucle paramétrée

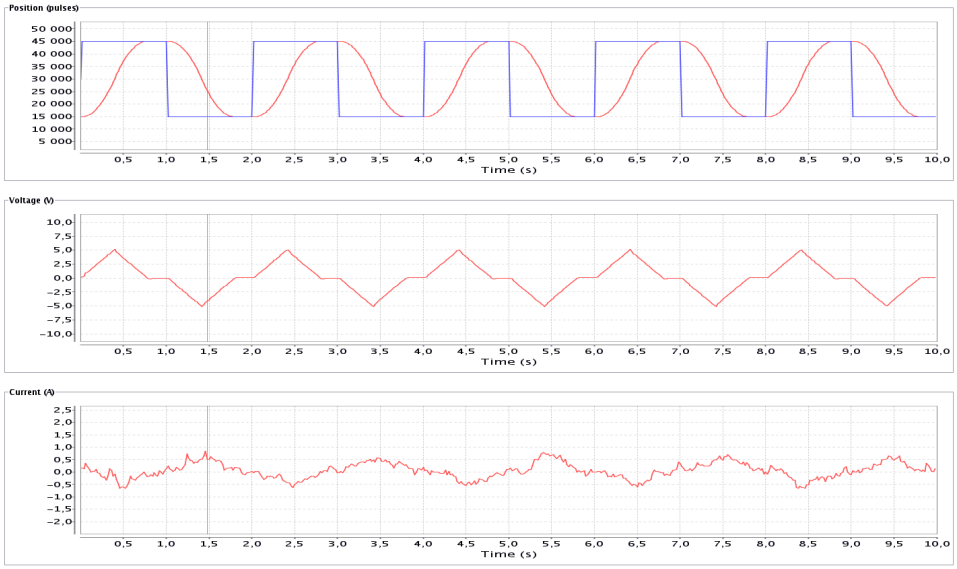

FIG. 5.6 – Résultats pour un contrôle en position avec rampes d'accélération

Chapitre 6

Implémentation

Sommaire

6.1 Définition du problème

Nous allons maintenant passer à l'implémentation des lois de contrôle sur le robot participant au concours de robotique.

Le robot est présenté aux figures 6.1 et 6.2 et a été entièrement réalisé par des membres de l'équipe du projet Eurobot 2008.

Après l'installation dans le robot de deux exemplaires de la carte électronique développée, nous allons nous occuper de leur adressage par l'ordinateur embarqué.

Deux choses sont encore à implémenter pour terminer notre travail, la localisation du robot par odométrie et la gestion des déplacements par suivi de trajectoire pré-calculée.

6.2 Localisation

Comme nous l'avons vu à la section 2.2.2, nous pouvons utiliser l'odométrie pour connaître la position du robot. L'implémentation de celle-ci est très simple, nous demandons aux deux cartes électroniques la position de

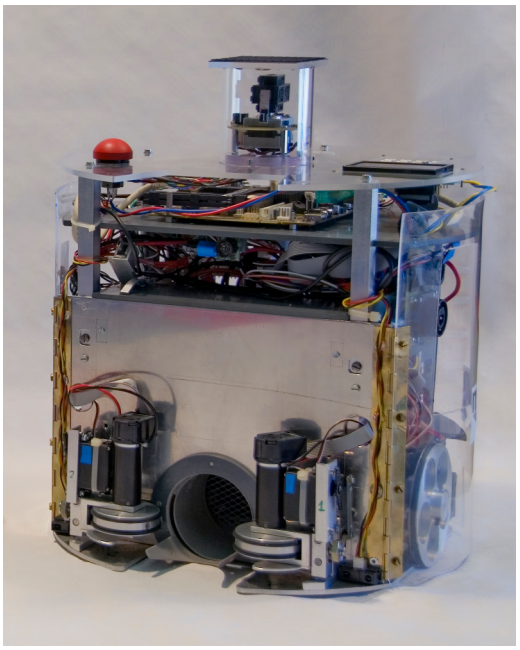

FIG. 6.1 – Le robot - “Frida” - Vue avant

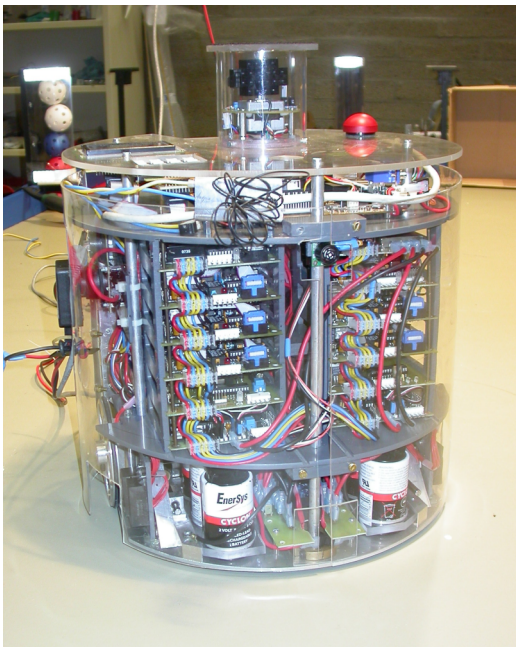

FIG. 6.2 – Le robot - “Frida” - Vue arrière

leur moteur et nous appliquons l'algorithme étudié pour connaître la position actuelle du robot.

Malheureusement, l'information qui en résulte n'est pas parfaite et se dégrade au cours du temps. En outre, cette méthode ne nous permet pas de connaître la position de départ et celle-ci devra être fournie de manière différente. En effet, le principe du calcul odométrique est basé sur les variations de position du robot et les erreurs systématiques s'ajoutent au cours du temps, ce qui dégrade la précision de la localisation.

Afin d'avoir un calcul d'odométrie précis en minimisant les erreurs systématiques, il convient de calibrer précisément le diamètre des roues de locomotion et la distance entre celles-ci.
Pour ce faire, nous pouvons comparer la mesure d'odomètrie avec la position réelle du robot après des déplacements simples, comme des lignes droites, des rotations ou des déplacements en carré. Nous remarquerons par exemple qu'il est impossible de réaliser deux roues parfaitement identiques, et qu'en tenir compte a un impact important sur la précision de l'orientation.

Afin de palier à ces lacunes, la position exacte du robot ne reposera donc pas que sur l'odométrie, mais sur plusieurs sources différentes (capteurs de distance, triangulation par balises, odométrie, ..), toutes ces informations sont traitées et filtrées afin de nous donner une estimation précise de la position du robot à tout instant.

Afin de diminuer les erreurs systématiques inhérentes à l'odométrie, nous avions aussi imaginé dans un premier temps d'utiliser des capteurs de quadrature supplémentaires sur deux petites roues libres montées sur amortisseurs et placées sur le même axe que les roues motrices. Ces roues libres pouvant être plus petites et subissant moins de contraintes que les roues motrices (pas de dérapage par exemple), nous auraient certainement permis d'effectuer un calcul odométrique entaché de moins d'erreurs systématiques. Malheureusement, cette solution a été abandonnée pour des raisons de difficulté de construction mécanique. Il pourrait malgré tout être intéressant d'en valider l'efficacité lors d'une participation future.

Malgré tous ces problèmes, la mesure fournie par les encodeurs des moteurs s'est révélée très satisfaisante, ceci principalement grâce au fait que nous garantissons normalement un roulement sans glissement des roues de locomotion, la mesure d'odométrie a donc pu être notre principale source de localisation, principalement lors des déplacements.

6.3 Déplacement

Une fois les lois connues et le programme de génération de trajectoire écrit, l'implémentation du suivi est aussi relativement simple.

Lorsque le robot est à l'arrêt et qu'il reçoit une nouvelle configuration de destination, ou lorsque le robot termine une trajectoire et que sa liste de points de passage n'est pas vide, nous calculons une trajectoire entre la configuration courante et la configuration suivante, puis nous effectuons un suivi de celle-ci. Afin de permettre le suivi, les trajectoires sont discrétisées de manière telle à avoir une différence maximum d'orientation ou de position entre deux points de la trajectoire fixée.

Malheureusement notre calcul de trajectoire n'est pas parfait et ne donne pas toujours de résultat. Afin de palier à ce problème, lorsque nous ne parvenons pas à trouver une solution, nous nous lançons dans une trajectoire de freinage et puis nous réessayons de générer une nouvelle trajectoire à intervalles réguliers. Les problèmes dans la génération de trajectoire n'apparaissant que si la vitesse initiale n'est pas nulle (en effet si la vitesse est nulle nous pouvons toujours nous ramener à une suite de rotation simple et de lignes droites), comme nous effectuons une trajectoire de freinage, nous finirons toujours pas nous retrouver dans une configuration non problématique.

Remarquons que les problèmes que nous avons lors de la génération de trajectoire ne sont pas dus au principe en lui même, mais aux heuristiques que nous avons utilisées pour ne pas parcourir toutes les combinaisons possibles des rayons $r1$ et $r2$. Afin de palier à ce problème, il pourrait être intéressant d'en trouver de nouvelles.

Un autre problème qui peut survenir pendant notre déplacement est la présence d'un adversaire sur notre trajectoire. Celle-ci sera détectée par exemple par des capteurs de distance. Lorsque l'adversaire se trouve sur notre trajectoire, nous arrêtons le suivi en cours et nous calculons une trajectoire de freinage qui devrait nous permettre de l'éviter. C'est ensuite à la stratégie de plus haut niveau de nous donner de nouveaux points de passage pour continuer notre déplacement.

6.4 Canon et Barillet

La carte électronique de contrôle moteur à été développée dans une optique de généricité. Elle peut donc être utilisée avec des moteurs de puissance différente et dans un but qui peut être totalement différent de la locomotion

d'un robot.

Le robot qui a participé au concours contient en fait trois exemplaires supplémentaires de la carte en plus des deux cartes nécessaires à la locomotion.

Nous utilisons deux de ces cartes pour piloter les roues d'un canon. Le canon est constitué d'un mécanisme relativement simple, deux roues tournant à grandes vitesses dans des sens opposés entre lesquelles nous amenons une balle afin de l'éjecter. Une des clés, afin d'effectuer un tir balistique est la précision sur la vitesse des roues lorsque la balle est éjectée. Tâche pour laquelle les cartes électroniques se sont parfaitement illustrées, nous permettant d'avoir un tir relativement précis.

La dernière carte servait à contrôler en position un barillet contenant les balles à éjecter, il fallait positionner les loges contenant les balles de manière précise devant l'entrée du canon avant un tir ou lors d'un ramassage. De nouveau la carte s'est très bien comportée, de plus le système de rampes d'accélération développé pour le contrôle en position a permis de ne pas trop stresser le réducteur à vis sans fin sur lequel était placé le moteur du barillet.

Ces deux applications supplémentaires démontrent donc, d'une belle manière, l'utilisation générale qui peut être faite de la carte.

6.5 Résultats

Une fois l'implémentation terminée, nous pouvons évaluer les performances du système.

Nous verrons ci-après (à la figure 6.3) un exemple de suivi de trajectoire réalisé par notre robot, on observe que le robot effectue bien une rotation sur place, comme demandé, et qu'il suit ensuite relativement bien la trajectoire.

Nous sommes relativement satisfaits des résultats, les trajectoires étant toujours globalement bien suivies. La précision finale en fin de trajectoire n'est par contre pas toujours suffisamment satisfaisante dans certains cas, et nous utiliserons alors un léger contrôle en position comme discuté à la section 3.5.

Les déplacements sont impressionnants et leur durée est bien inférieure aux durées que nous obtiendrions si nous nous limitions uniquement à des rotations et à des lignes droites. Nous n'observons, en outre, aucun dérapage

significatif des roues de locomotion, ce qui est important pour notre calcul d'odométrie.

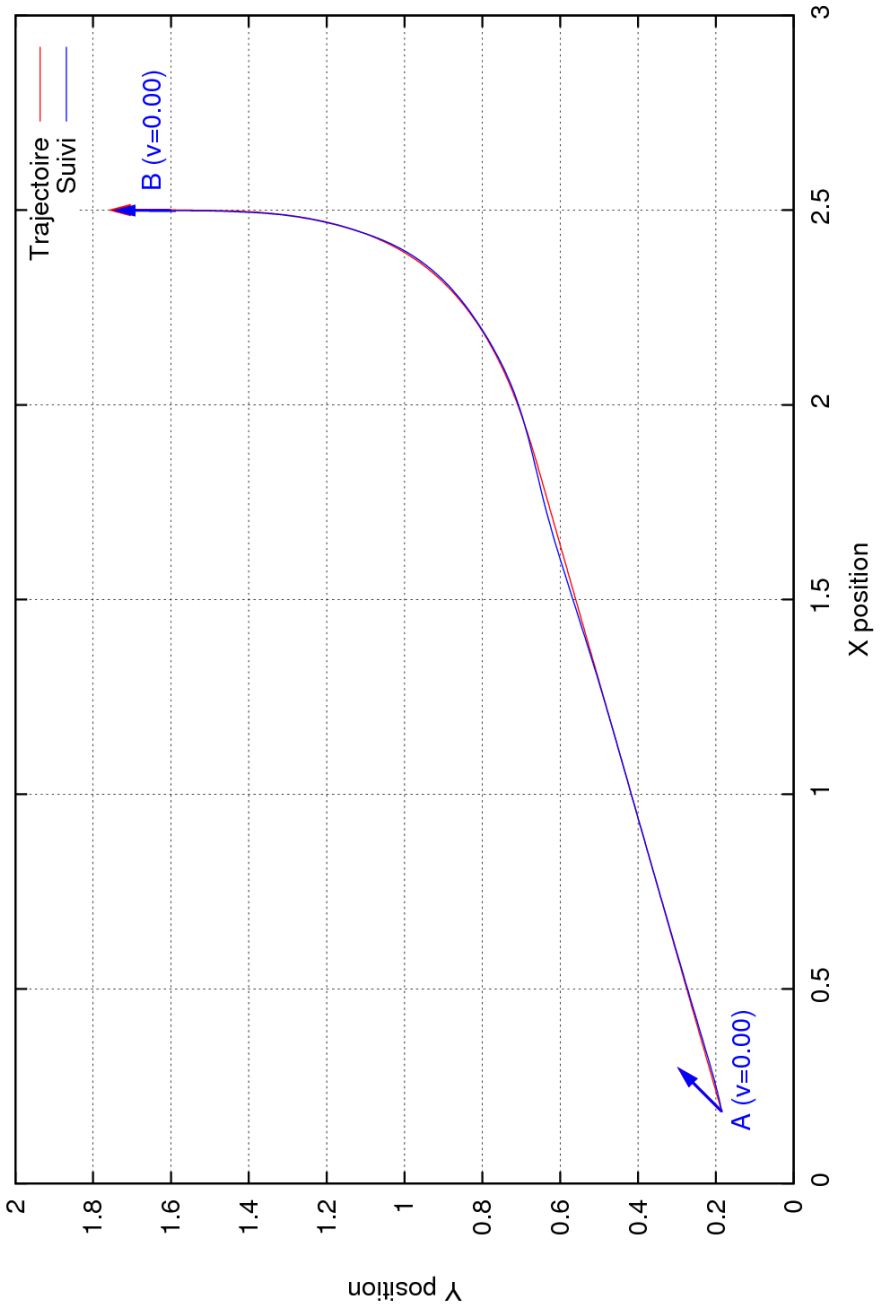

FIG. 6.3 – Suivi de trajectoire - En rouge, la trajectoire de référence précalculée - En bleu, la trajectoire réellement suivie par le robot

Chapitre 7

Conclusions

Sommaire

7.1 Application au concours Eurobot

Comme il a été déjà répété de nombreuses fois, ce travail s'inscrit dans le cadre d'une participation à un concours de robotique mobile.

La participation à ce concours est un projet pluridisciplinaire de grande envergure impliquant une quinzaine de personnes. Il nous a permis d'appliquer et de tester notre travail en situation réelle et a donc été une énorme source de motivation.

Eurobot[1] est un concours de robotique amateur européen rassemblant principalement des équipes d'étudiants et des clubs indépendants. Le concours est constitué de qualifications nationales puis d'une finale internationale accueillant cette année plus de 27 pays.

Chaque match oppose deux robots autonomes se déplaçant sur une surface de trois mètres sur deux pendant une durée de nonante secondes. Le vainqueur étant le robot qui marque le plus de points.

Le thème définissant le contage des points change chaque année, pour l'année 2008 il s'intitule "Mission to Mars". En résumé, les robots doivent

[1]Des informations détaillées du concours pourront être trouvées sur le site officiel : `http://www.eurobot.org/`

collecter des balles de couleurs différentes (des morceaux de roche martienne) et les déposer dans une rigole ou dans des paniers en hauteur. Chaque balle de sa couleur donnant deux points, chaque balle blanche un point, et chaque balle de sa couleur entourée par deux balles blanches dans la rigole donnant droit à un bonus supplémentaire.

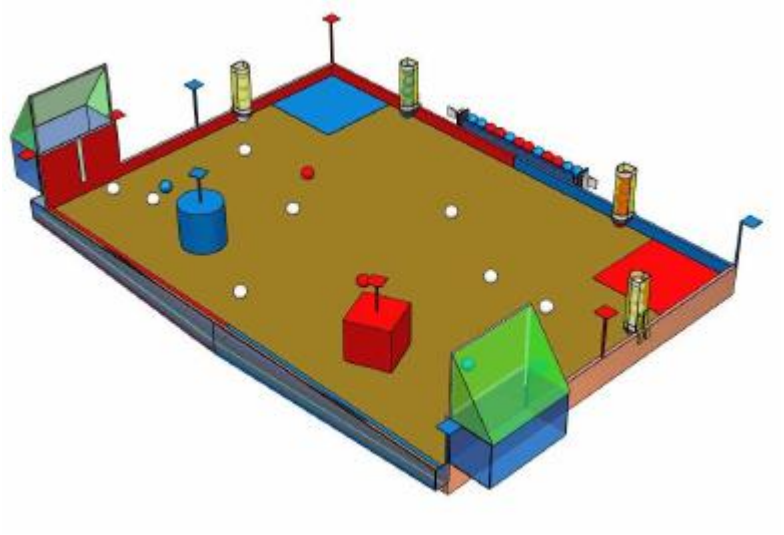

FIG. 7.1 – Table Eurobot 2008 - Source : `http://www.eurobot.org`

Une description complète des règles pourra être trouvée dans [Eur07].

Le bilan est positif puisque nous avons terminé deuxièmes sur une vingtaine d'équipes à la finale belge, et que nous sommes qualifiés pour la finale européenne.

Globalement, jusqu'à présent, le système de locomotion s'est très bien comporté et aucun problème majeur n'est à signaler au niveau de sa fiabilité.

Le concours nous a aussi permis d'évaluer les performances de notre système de locomotion en comparaison aux autres participants. Il ressort que notre système de locomotion est certainement un des plus complexes(pour un robot à roues) envisagés lors du concours.

Il nous a permis de rivaliser en terme de rapidité et de précision de déplacement avec les meilleures équipes.

7.2 Résultats

Tout au long de ce travail, nous avons été amenés à réaliser le système de locomotion complet d'un robot mobile.

Le robot mobile utilisé est un robot à roues de type unicycle dont la locomotion est assurée par deux moteurs à courant continu.

Afin de trouver une méthode générale permettant d'amener le robot d'une configuration donnée à une autre, nous avons été amenés à développer et à implémenter un certain nombre de lois de contrôle qui régissent le déplacement du robot.
Ces lois consistent principalement, en la génération préalable d'une trajectoire réalisable et d'un profil de vitesse associé (i.e. respectant des contraintes de courbure, d'accélération, de périmètre, ...), et en un suivi de celle-ci sur base, du profil de vitesse, et de la minimisation de l'erreur en orientation et de l'erreur en distance du robot par rapport à la trajectoire.

Les différentes trajectoires peuvent, en outre, être enchaînées sans nécessiter un arrêt complet du robot. Ce qui nous permet d'effectuer des trajectoires complexes avec des points de passage imposés d'une manière simple et efficace.

Le suivi de trajectoire suppose que les vitesses sont directement transmises aux moteurs. Pour ce faire, une carte électronique pour la commande et le contrôle d'un moteur à courant continu a été développée. Elle est constituée, d'une part, d'un pont en H discret piloté en *Locked Anti-phase*, et d'autre part, d'une partie logique responsable de l'asservissement du moteur en vitesse ou en position grâce, notamment, à une boucle de régulation *PID*.

Le système complet a ensuite été testé, et ses performances ont pu être évaluées lors du concours de robotique Eurobot.

Les performances atteintes par ce système sont sans commune mesure avec celles obtenues par les systèmes développés lors des participations précédentes au concours de robotique. Le système n'est par contre pas encore parfait, mais permet déjà de gérer avec une très bonne efficacité l'ensemble des déplacements du robot.

Un effort particulier a aussi été réalisé quand à la généricité de la carte électronique développée. Celle-ci a déjà été éprouvée, puisque le robot utilisé compte en plus des deux cartes nécessaires à la locomotion, trois cartes supplémentaires qui permettent de gérer en vitesse ou en position les autres moteurs à courant continu nécessaires au fonctionnement du robot.

7.3 Améliorations possibles

Comme nous l'avons dit, le système n'est pas encore parfait et certaines choses pourraient encore être améliorées.

Au niveau de la génération des trajectoires, il subsiste encore certains cas pathologiques pour lesquels notre programme ne retourne pas de solutions. Ceci est principalement dû aux heuristiques utilisées pour le choix des rayons $r1$ et $r2$. Il pourrait être intéressant d'essayer d'en développer de meilleures, permettant de toujours atteindre une solution, sans, pour autant, faire une recherche systématique parmi toutes les combinaisons possibles.

Dans la génération de trajectoire toujours, notre solution ne nous permet pour l'instant de tenir compte que d'un périmètre éventuel. Il pourrait être intéressant de pouvoir en plus gérer des obstacles (comme le robot adverse) à l'intérieur de ce périmètre. Cela pourrait par exemple être réalisé à l'aide d'un algorithme de *path-finding*. Celui-ci permettrait de calculer un certain nombre de points de passage pour atteindre notre objectif final tout en évitant les obstacles. La génération des trajectoires entre ces différents points de passage, et leur enchaînement, pouvant ensuite être réalisée par la méthode que nous avons étudiée.

Les deux défauts principaux de la carte électronique sont son encombrement et son coût. Il pourrait être intéressant pour palier à ces deux problèmes, de remplacer les convertisseurs DC-DC isolés par des pompes de charge, et de remplacer les composants actuels par des composants de surface. Il pourrait ainsi être possible de réduire substantiellement le coût de la carte et de réduire par un facteur deux son encombrement.

Bibliographie

[AM04] K.J. Aström and R. Muray. Feedback systems : An Introduction for Scientists and Engineers, 2004.

[Bay08] B. Bayle. Robotique mobile, 2008.

[Boi05] B. Boigelot. La plate-forme robotique Glutobor III : Organisation des composants électroniques et informatiques, 2005.

[Büh91] H. Bühler. *Convertisseurs statiques*. Presses Polytechniques et Universitaires Romandes, PPUR, 1991.

[CR93] J.L. Crowley and P. Reignier. Asynchronous Control of Rotation and Translation for a Robot Vehicle. *Journal of Robotics and Autonomous Systems*, 1993.

[Cro95] J.L. Crowley. Mathematical foundations of navigation and perception for an autonomous mobile robot. *Workshop on Reasoning with Uncertainty in Robotics*, pages 4–6, 1995.

[Eur07] Eurobot 2008, Mission to Mars, Rules 2008 - Revision 1, 2007.

[Gau99] E. Gauthier. *Utilisation des réseaux de neurones artificiels pour la commande d'un véhicule autonome*. PhD thesis, Thèse de Doctorat, Inst. Nat. Polytechnique de Grenoble, 1999.

[Gen05] A. Genon. ELEC032 : Électronique de puissance, 2005.

[Lek03] M. Lekeux. Création d'une architecture générique de robot mobile. Master's thesis, Travail de fin d'études, Université de Liège, 2003.

[LNRL06] L. Labakhua, U. Nunes, R. Rodrigues, and F.S. Leite. Smooth trajectory planning for fully automated passengers vehicles. 2006.

[MS93] A. Micaelli and C. Samson. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. *Research Report*, 2097, 1993.

[Ran04] A. Ranganathan. The Levenberg-Marquardt Algorithm, 2004.

[Reg99] T. Regan. A DMOS 3A 55V H-Bridge The LMD18200. *National Semiconductor Application Note*, 868, 1999.

[Rib02] M. I. Ribeiro. Smooth path planning. 2002.

[Sep07] R. Sepulchre. SYST003 : Analyse et synthèse des systèmes, 2007.

[ZN42] JG Ziegler and NB Nichols. Optimal settings for automatic controllers. *Trans. ASME*, 64(11) :759–768, 1942.

Annexe A

Schéma de la carte électronique

Liste des composants

20 Résistances		
Quantité	Références	Valeur
5	R1-R4, R12	10K
4	R5-R8	10R
3	R9-R11	100R
5	R13, R15-R17, R19	1K
1	R14	220R
1	R18	3K9
1	R20	392R

20 Condensateurs		
Quantité	Références	Valeur
11	C1-C6, C9, C10, C12, C19, C20	100nF
2	C7, C8	10uF / 25V
1	C15	10uF / 25V / tantale
1	C11	100uF / 25V
1	C13	2200uF / 25V
1	C14	470nF
2	C17, C18	22pF
1	C21	10nF

7 Circuits intégrés		
Quantité	Références	Valeur
2	U1, U2	IR2010
2	U3, U4	TEL2-1212
1	U5	LM7805
1	U6	26LS32
1	U7	ACS754

5 Transistors		
Quantité	Références	Valeur
4	Q1-Q4	IRF1302
1	Q5	2N3904

12 Diodes		
Quantité	Références	Valeur
4	D1-D4	MUR420
4	D5-D8	1N4148
3	D9-D11	Zener 18V (BZX85-C18)
1	D12	1N4001

12 Divers		
Quantité	Références	Valeur
1	PIC	PIC 18F2431
1	X1	Quartz 10Mhz
1	LED	LED rouge
1	FUSE	Fusible 5A LOW
1	FUSE2	Fusible 250mA
1	ENC	TRANS 10 DIL
1	BUS	SIL-156-06 coudé
1	PROG	SIL-156-05 coudé
4	MOT1, MOT2, MOT_0V, MOT_12V	Conn. Faston coudé

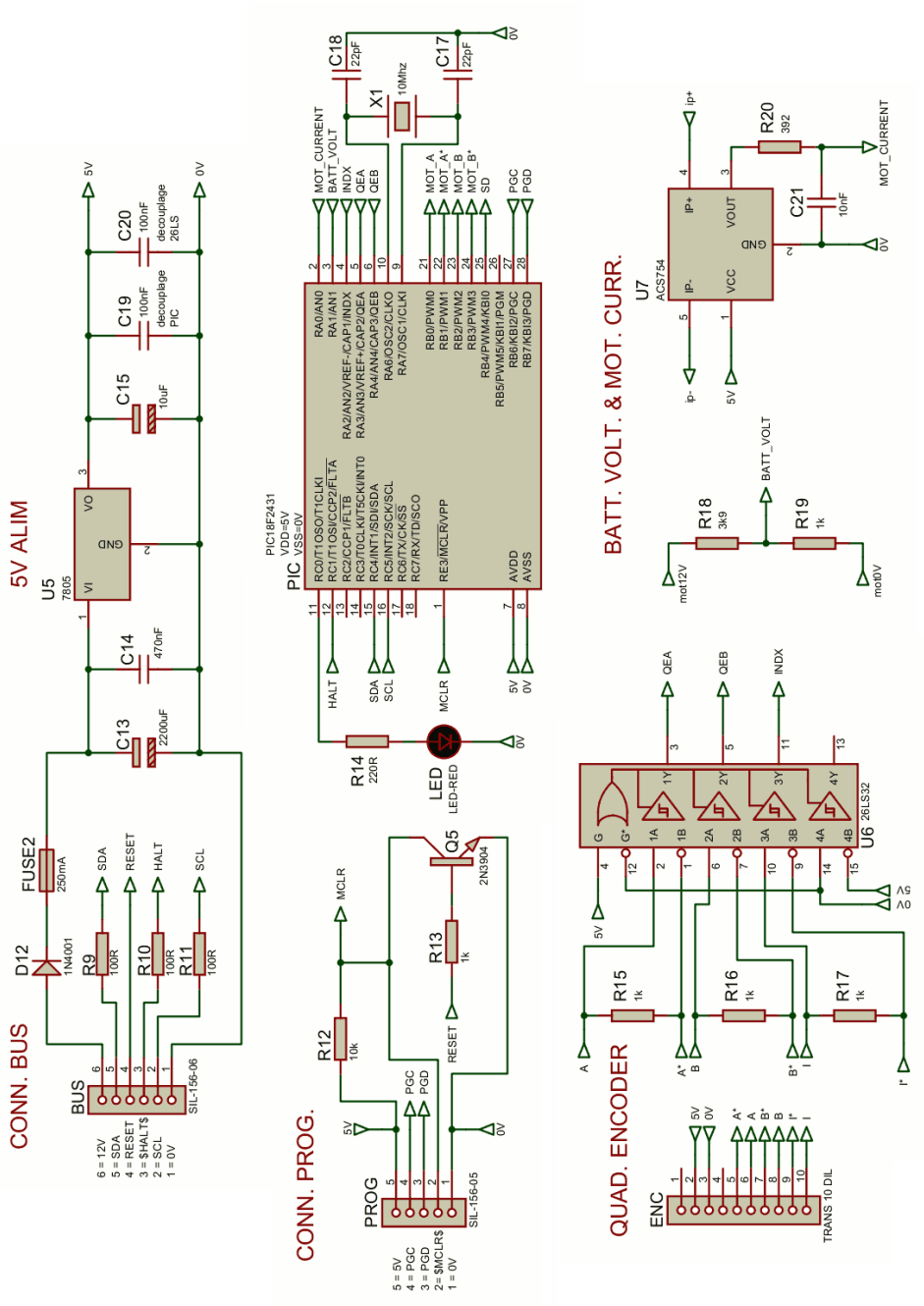

FIG. A.1 – Schéma électrique complet de la carte électronique - Partie 1

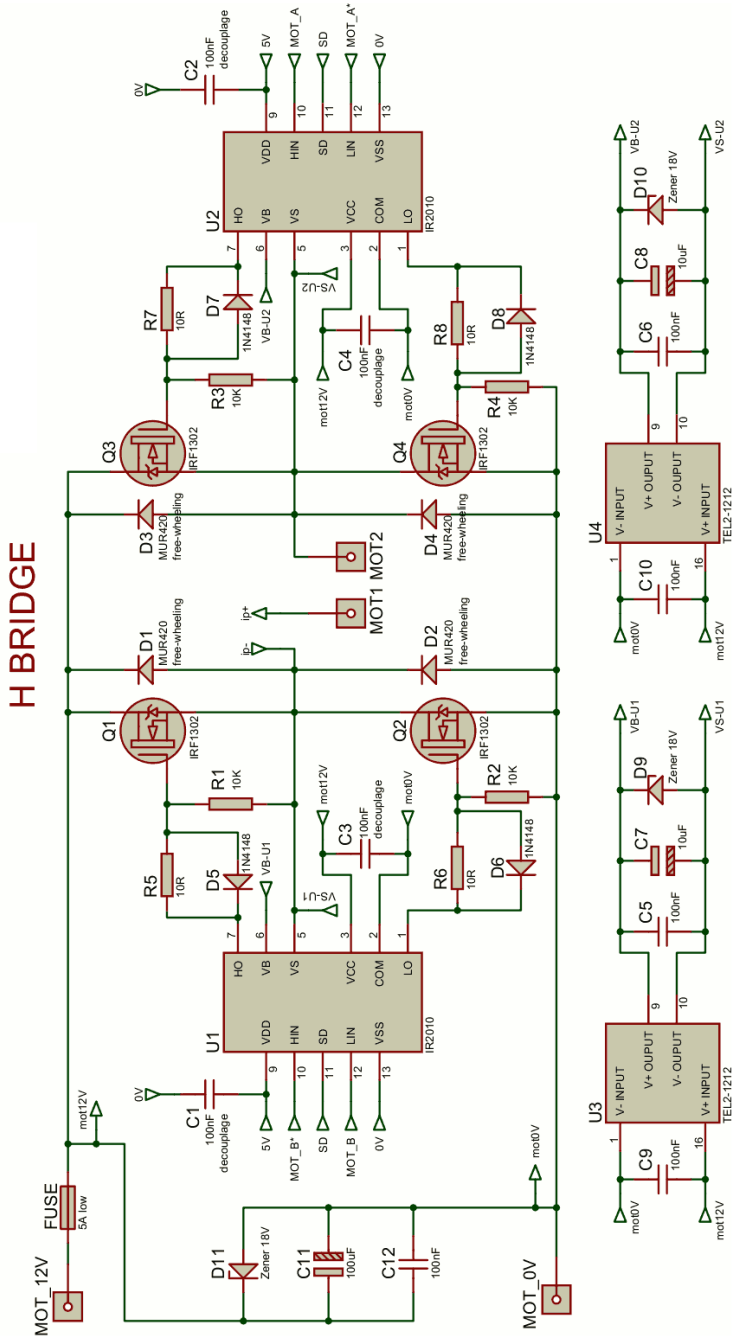

FIG. A.2 – Schéma électrique complet de la carte électronique - Partie 2

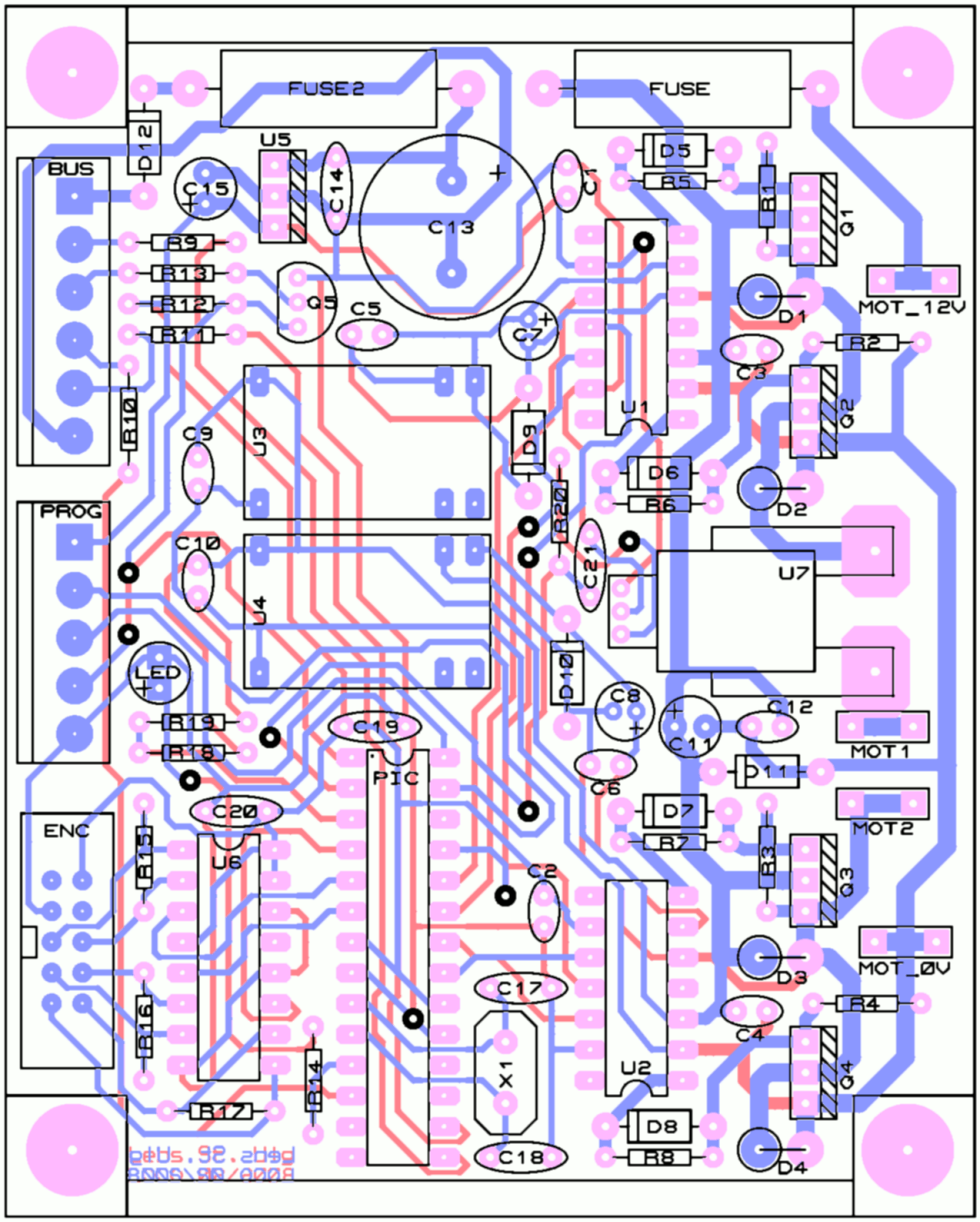

FIG. A.3 – Layout carte électronique - Placement des composants

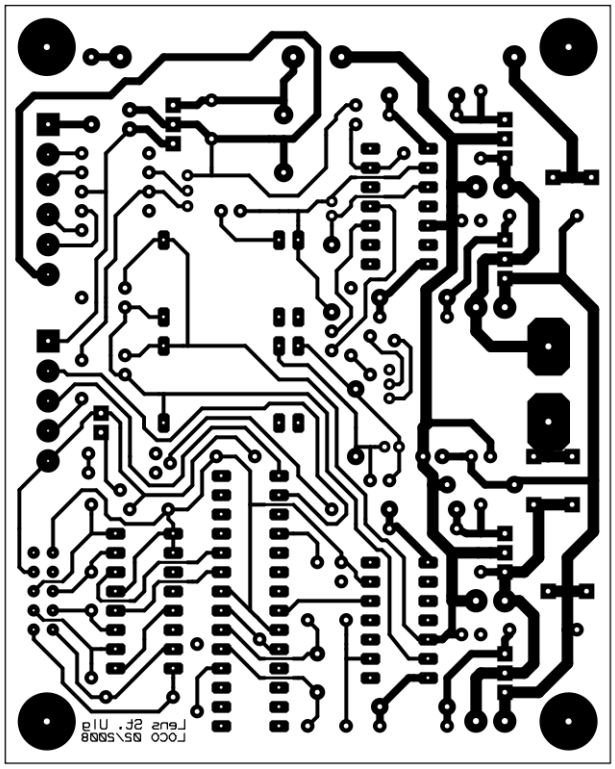

FIG. A.4 – Layout carte électronique - Face inférieure

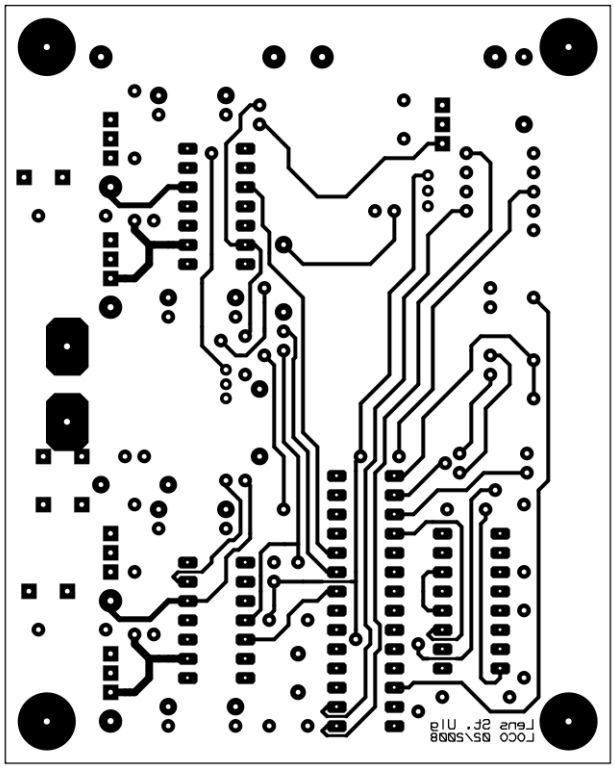

FIG. A.5 – Layout carte électronique - Face supérieure

Annexe B

Messages I2C

Messages génériques

Requête *who is*

Code de requête : 0x00

Renvoie 4 caractères indiquant le type de carte adressée suivis d'un octet spécifiant la version.

Requête *get errors*

Code de requête : 0x01

Renvoie 2 octets dont les différents bits sont associés aux différents types d'erreurs pouvant survenir sur la carte. Les bits mis à 1 indiquent qu'une erreur s'est produite.

premier octet erreurs génériques		deuxième octet erreurs spécifiques aux cartes	
bit	signification	bit	signification
0	réservé	0	sur-courant
1	réservé	1	erreur d'écriture en eeprom
2	erreur i2c : overflow	2	lag dans la boucle de contôle
3	erreur i2c : bad state error	3	//
4	erreur i2c : checksum error	4	//
5	erreur i2c : unknown command	5	//
6	erreur i2c : bad buffer size	6	//
7	erreur i2c : buffer full	7	//

Ordre *clear errors*

Code d'ordre : 0x02

Prend en argument 2 octets dont les différents bits sont associés aux différents types d'erreurs pouvant survenir sur la carte. Les bits mis à 1 indiquent les erreurs qu'il faut oublier.

Ordre *feedback parameters setup 1*

Code d'ordre : 0x03

Arguments (13 octets au total) :

- 1 octet : délai entre la réception de l'ordre et la mise à jour effective du rapport de PWM, de la vitesse de référence ou de la position de référence en 10^{-4}s.
- 2 octets : k_pv, proportional gain (speed control)
- 2 octets : k_iv, integral gain (speed control)
- 2 octets : k_dv, derivative gain (speed control)
- 2 octets : $b_{sp}v$, proportional set-point weight (speed control)
- 2 octets : $b_{sd}v$, derivative set-point weight (speed control)
- 2 octets : courant maximum. (en $10^{-2}A$)

Si délai = 0 $\Rightarrow$ aucune synchronisation.

Les différents gains et *set-point*s sont exprimés en 8.8-*fixed-point*.

k_pv : $10^{-5}\frac{V\ s}{pas}$.
k_iv : $10^{-3}\frac{V}{pas}$.
k_dv : $10^{-3}\frac{V}{pas\ s^2}$.

Ordre *feedback parameters setup 2*

Code d'ordre : 0x04

Arguments (15 octets au total) :

- 2 octets : k_pp, proportional gain (position control)
- 2 octets : k_ip, integral gain (position control)
- 2 octets : k_dp, derivative gain (position control)
- 2 octets : $b_{sd}p$, derivative set-point weight (position control)
- 3 octets : nombre de pas par tour => MAXPOS (position control)

- 2 octets : vitesse maximum (position control)
- 2 octets : accélération maximum (position control)

Les différents gains et *set-point*s sont exprimés en 8.8-*fixed-point*.

$k_pp : 10^{-2}\frac{V}{pas}$.
$k_ip : 10^{-2}\frac{V}{pas\ s}$.
$k_dp : 10^{-4}\frac{V\ s}{pas}$.

Vitesse exprimée en $10^2\frac{pas}{s}$.
Accélération exprimée en $10^2\frac{pas}{s^2}$.
Si vitesse = 0 et accélération = 0 $\Rightarrow$ aucune rampe.

Rmq : $(2vel + acc)^2 < 2^{31}$

Ordre *Stop*

Code d'ordre : 0x05

Met le moteur en roue libre.

Ordre *PWM power*

Code d'ordre : 0x06

Arguments :

- 2 octets : PWM ratio, $[0\,;\,512]$.

Moteur arrêté $\Rightarrow$ PWM ratio = 256.

Lors de la réception de cet ordre, au lieu de *reference speed*, le contrôle en feedback loop est désactivé jusqu'à la réception d'un nouvel ordre *reference speed*.

2 sec sans ordre met le moteur en roue libre.

Ordre *reference speed*

Code d'ordre : 0x07

Arguments :

- 2 octets : vitesse/16 (MSB first), [-32768 ; 32767].

La vitesse est exprimée en pas par seconde. Le seizième de la vitesse est passé en argument lors du passage de l'ordre.
Les encodeurs comportent 2000 pas par tour, le moteur possède une vitesse maximale théorique de 8000 tours/minute soit 133 tours/seconde. Il en résulte 266000 pas/seconde en vitesse de pointe, à comparer avec la gamme dynamique [-524288 ; 524272] de l'argument.

2 sec sans ordre met le moteur en roue libre.

Ordre *reference position*

Code d'ordre : 0x08

Arguments :

- 3 octets : position (MSB first), [0 ; MAXPOS-1]

La position est exprimée en pas. La position est relative au zero lorsque la carte démarre. Le mouvement maximum est d'un demi-tour.

2 sec sans ordre met le moteur en roue libre.

Requête *coder positions*

Code de requête : 0x11

Réponse :

- 2 octets : dernière valeur de l'encodeur synchronisée, [0 ; 65535].
- 2 octets : valeur de l'encodeur synchronisée 10ms avant, [0 ; 65535].

Requête *coder position and velocity*

Code de requête : 0x12

Réponse :

- 2 octets : dernière valeur de l'encodeur, [0 ; 65535].
- 2 octets : dernière vitesse calculée de l'encodeur, [0 ; 65535].

Requête *current position*

Code de requête : 0x13

Réponse :

- 3 octets : position courante, [0 ; MAXPOS-1].

La position est la position utilisée pour le contrôle en position.

Requête *PWM, voltage and current*

Code de requête : 0x14

Réponse :

- 2 octets : dernière PWM envoyée au moteur, [0 ; 512].
- 2 octets : dernière mesure de tension de l'alimentation du moteur, [0 ; 1023].
- 2 octets : dernière mesure du courant consommé par le moteur, [0 ; 1023].

Moteur arrêté $\Rightarrow$ PWM ratio = 256.

Tension = $\frac{val}{1024} * 4,9 * 5V$.
Courant = $\left(\frac{val-512}{1024}\right) * 5V * 25\frac{A}{V}$. $\quad (40\frac{mV}{A})$

Annexe C

Modélisation moteur DC

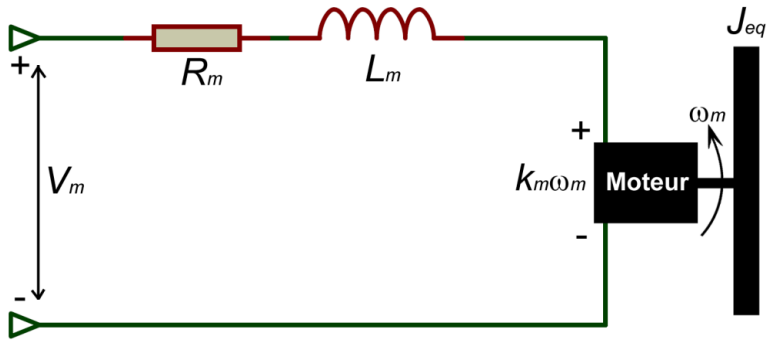

FIG. C.1 – Schéma du moteur DC

1. A partir des lois de Kirchhoff nous pouvons écrire :

$$V_m(t) = R_m I_m(t) + L_m \frac{dI_m(t)}{dt} + k_m \omega_m(t) \tag{C.1}$$

où $k_m\omega_m$ est la force contre-électromotrice du moteur.

et si $L_m \ll R_m$ nous pouvons écrire :

$$V_m(t) \simeq R_m I_m(t) + k_m \omega_m(t) \tag{C.2}$$

Ce qui donne dans le domaine de Laplace :

$$V_m(s) \simeq R_m I_m(s) + k_m \omega_m(s) \tag{C.3}$$

2. A partir des lois de Newton nous pouvons écrire :

$$J_{eq}\frac{d\omega_m(t)}{dt} = k_m I_m(t) \tag{C.4}$$

où J_{eq} est le moment total d'intertie : J_m (moteur) + J_l (charge)

Ce qui donne dans le domaine de Laplace :

$$J_{eq}\omega_m(s)s = k_m I_m(s) \tag{C.5}$$

3. En remplaçant $I_m(t)$ dans l'équation C.5 grâce à la relation C.3 on obtient :

$$J_{eq}\omega_m(s)s = k_m\frac{V_m(s) - k_m\omega_m(s)}{R_m} \tag{C.6}$$

$$J_{eq}\omega_m(s)s = \frac{k_m V_m(s)}{R_m} - \frac{k_m^2\omega_m(s)}{R_m} \tag{C.7}$$

4. Finalement à partir de C.7 nous pouvons dériver la fonction de transfert tension → vitesse :

$$G_{\omega,V}(s) = \frac{\omega_m(s)}{V_m(s)} = \left(\frac{k_m}{R_m}\right)\left(\frac{R_m}{R_m J_{eq}s + k_m^2}\right) \tag{C.8}$$

et

$$\boxed{G_{\omega,V}(s) = \frac{K}{\tau s + 1}} \tag{C.9}$$

avec

$$\boxed{K = \frac{1}{k_m}} \quad et \quad \boxed{\tau = \frac{J_{eq}R_m}{k_m^2}} \tag{C.10}$$

5. La fonction de transfert tension → position est donnée par :

$$\boxed{G_{\theta,V}(s) = \frac{1}{s}G_{\omega,V}(s)} \tag{C.11}$$

Annexe D

Levenberg-Marquardt

L'algorithme de Levenberg-Marquardt permet d'obtenir une solution au problème de minimisation des moindres carrés. Il est basé sur une interpolation de l'algorithme de Gauss-Newton et de la méthode de descente de gradient.

La fonction à minimiser sera de la forme :

$$\sum_i (y(i) - f(i|\boldsymbol{p}))^2 \tag{D.1}$$

où $\boldsymbol{p} = (p_1, p_2, ..., p_n)$ est un vecteur.

La procédure pour trouver la solution est itérative et consiste à remplacer à chaque itération $\boldsymbol{p}$ par une nouvelle approximation $\boldsymbol{p} + \boldsymbol{\Delta p}$ (nous devrons donc fournir un vecteur $\boldsymbol{p_0}$ initial à l'algorithme).

avec

$$\boldsymbol{\Delta p} = (J^T J + \lambda I)^{-1} J^T [\mathbf{y} - \mathbf{f}(\mathbf{t}|\boldsymbol{p})] \tag{D.2}$$

où J est la matrice jacobienne de f en $\boldsymbol{p}$.

Le facteur d'amortissement λ sera modifié à chaque itération selon les heuristiques suivantes :

1. Prendre $\lambda = 1$.
2. Mettre à jour $\boldsymbol{p}$ grâce à D.2.
3. Évaluer l'erreur avec le nouveau vecteur $\boldsymbol{p}$.
4. Si la mise à jour à eu pour effet d'augmenter l'erreur, alors nous revenons à la valeur précédente de $\boldsymbol{p}$ et nous multiplions λ par un facteur 10. Nous retournons à l'étape 2 et nous continuons l'algorithme.

5. Si la mise à jour à eu pour effet de diminuer l'erreur, nous conservons la nouvelle valeur de $\boldsymbol{p}$ et nous divisons λ par un facteur 10. Nous retournons à l'étape 2 et nous continuons l'algorithme.

La procédure est arrêtée, lorsque la précision désirée est atteinte ou lorsque λ dépasse une certaine valeur (i.e. il n'est plus possible d'améliorer la solution).

Nous remarquerons que lorsque λ est $\ll$ nous approchons de Gauss-Newton et que lorsque λ est $\gg$ nous approchons de la méthode de descente de gradient.

En outre, cette méthode ne garantit pas que le minimum trouvé est le minimum de la fonction mais seulement un minimum local, il serra donc parfois être nécessaire de fournir un vecteur $\boldsymbol{p_0}$ suffisamment proche de la solution recherchée.

Une description plus détaillée de l'algorithme de Levenberg-Marquardt pourra être trouvée dans [Ran04].