Université Libre de Bruxelles

Faculté des sciences appliquées

BA2 orientation ingénieur civil

Rapport de projet BA2

Section informatique : Robot motard

Groupe 14 Tuteur : Verhaegen Gary

Delhaye Quentin

Dumont Arnaud

Giaux Camille

Lasbleis Pierre

Roig Gauthier

Bruxelles, 19 mars 2012

The Lego Motorcyclist robot

by Quentin Delhaye, Arnaud Dumont, Camille Giaux, Pierre Lasbleis and Gauthier Roig; Université Libre de Bruxelles, 2011–2012.

The main problem to be faced in the design of a motorcyclist robot able to follow a circuit with the "Lego Mindstorms" kit is about how to manage two tasks simultaneously, which are the stability-control task and the circuit-follower task. The first step was to analyse the forces applying on it, in order to find out how it must react to get back in a stable position. The next step consisted of choosing the best programming language to be used. Several were tested such as NXT-G, NBC, leJos and NXC (C-like syntax), which appeared to be the most effective language. Then, main tasks have been studied, beginning with stability control. A Proportional-Derivative-Integral (PID) controller appeared to be an accurate solution to the problem. In order to provide the information required by the PID, a gyroscopic sensor was used. The equilibrium is therefore controlled by a PID system, which can make the robot turn its front wheel in order to prevent it from falling down. A proportionnal regulation is being used to make the robot follow the line by using light sensors to give the error. Nevertheless, the problem of the mix of those two tasks has not been solved yet.

[200 words]

Keywords: motorcycle, robot, automatic, dynamic equilibrium, Lego Mindstorms , NXC, PID controller, gyroscopic sensor.

Le robot motard Lego

par Quentin Delhaye, Arnaud Dumont, Camille Giaux, Pierre Lasbleis et Gauthier Roig ; Université Libre de Bruxelles, 2011–2012.

Dans la fabrication d'un robot motard capable de suivre un circuit avec le kit "Lego Mindstorms", le problème principal à résoudre est de gérer deux tâches simultanément, à savoir : faire tenir le robot en équilibre et le faire suivre un circuit. Un bilan des forces s'exerçant sur le robot a donc été nécessaire pour comprendre comment il allait se rattraper. Il a ensuite fallu choisir un langage de programmation adapté au problème. Plusieurs de ces langages ont été envisagés comme par exemple NXT-G, LeJos, NBC et NXC (un langage similaire au C), et c'est ce dernier qui a été retenu. Par après, les tâches principales ont été étudiées de plus près, d'où il a été tiré, pour la gestion de l'équilibre, qu'un concept de gestion PID (Proportionnel - Intégral - Dérivé) pourrait convenir, l'erreur étant fournie par un capteur gyroscopique. L'équilibre est donc géré par un système PID implanté dans la brique intelligente du robot, qui rattrape l'équilibre du véhicule en le faisant tourner. Pour le suivi de circuit, une gestion similaire mais plus simple s'est avérée suffisante ; l'erreur étant cette fois fournie par des capteurs photosensibles. Enfin, une mise en commun de ces deux fonctions montre que, dans l'état actuel des choses, le robot perd trop facilement la ligne et n'est pas suffisamment réactif.

[217 mots] Mots-clés : moto, robot, automatique, équilibre dynamique, Lego Mindstorms , NXC, PID, capteur gyroscopique.

Table des matières

1. Introduction

Dans le cadre du projet d'année d'informatique de BA2 de la faculté des sciences appliquées, il a été demandé aux étudiants de construire et de programmer un robot capable de se déplacer en équilibre sur deux roues, tout en suivant un circuit au sol. Pour ce faire, un kit LEGO® MINDSTORMS® NXT[1] a été fourni au groupe.

Dans un premier temps, nous présenterons brièvement le kit LEGO qui nous a été mis à disposition. Ceci permettra d'analyser les outils dont nous disposons pour appréhender au mieux le projet.

Ensuite, nous nous intéresserons au choix du langage de programmation qui nous permettra de communiquer avec la brique NXT. Ceci fait, nous pourrons analyser la gestion des capteurs et des moteurs par le programme, avec notamment les différentes valeurs de retours possibles.

Nous décrirons aussi comment notre prototype a été élaboré, en détaillant les paramètres dont nous avons tenu compte. Nous évoquerons également l'évolution du prototype, et les modifications que nous avons dû effectuer à la suite de tests infructueux.

Enfin, nous détaillerons la méthode que nous avons utilisée pour gérer dynamiquement l'équilibre du robot, ainsi que les résultats obtenus. Parallèlement à cela, nous décrirons la méthode utilisée pour faire suivre un circuit à notre prototype et explorerons les problèmes qui pourraient survenir à la suite de la mise en commun de ces deux tâches, et les solutions auxquelles nous avons pensé.

Finalement, nous explorerons brièvement des perspectives d'avenir liées à notre prototype.

[1] LEGO et MINDSTORMS sont des marques déposées de LEGO Group

2. Kit Lego Mindstorms NXT

Ce chapitre va présenter les caractéristiques des éléments pertinents du kit Lego Mindstorms NXT [12] ainsi que quelques mesures plus précises de leurs réponses.

2.1 Brique intelligente

La brique intelligente est le cœur de tout robot construit avec ce kit ; elle sert de centre de calcul, reçoit les informations venant des quatre capteurs pouvant s'y connecter, et contrôle jusqu'à trois moteurs. Elle est alimentée par 6 piles format AA pour une tension de fonctionnement minimale de 7,2 V.

Elle est dotée de 256 kB de mémoire flash, ce qui est amplement suffisant pour stocker de nombreux programmes.

2.2 Capteur photosensible

Le capteur photosensible mesure l'intensité lumineuse de la lumière ambiante ou de la lumière réfléchie par une surface éclairée par la LED rouge si celle-ci est allumée. Ce capteur est capable de reconnaître des couleurs puisqu'elles ne réfléchissent pas toutes la même quantité de lumière. Il sera donc utilisé pour reconnaître le tracé d'un circuit et communiquer au prototype s'il le suit ou non.

L'un des points auquel il faut faire particulièrement attention est la variation d'intensité lumineuse captée en fonction de la hauteur du capteur au-dessus de la surface. La figure 2.1 reprend les mesures d'un capteur au-dessus de la couleur noire et au-dessus de la couleur blanche. Le noir retourne une intensité plus faible que le blanc, mais en augmentant la hauteur, le bruit de la mesure va augmenter et la différence entre le blanc et le noir, nette dans les premières mesures, s'amenuise. On remarquera ainsi qu'une hauteur inférieure à 3,5 mm est pathologique alors qu'au-dessus de 30 mm, les mesures sont

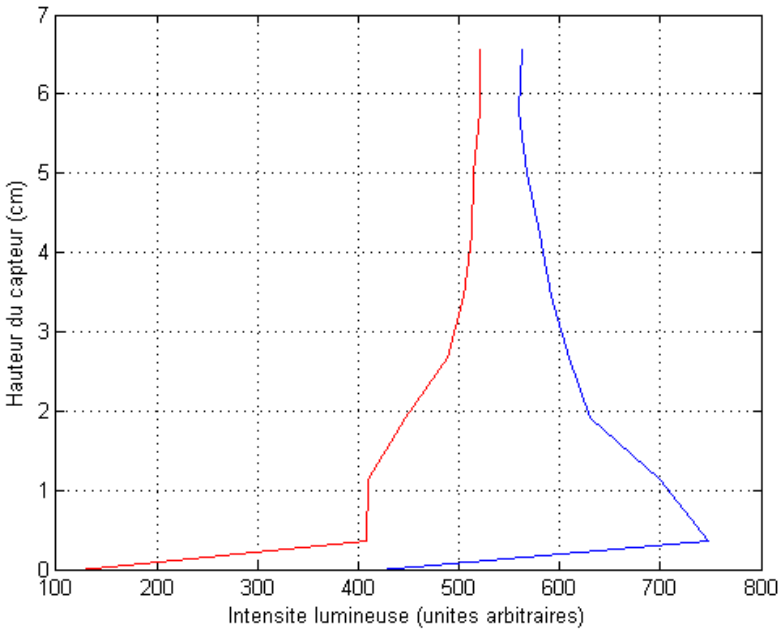

FIGURE 2.1 – Mesures de l'intensité lumineuse sur dégradé noir/blanc pour différentes hauteurs

trop proches pour pouvoir distinguer clairement la couleur. À cette hauteur, on dispose d'un intervalle de 100 unités (voir section 3.2).

2.3 Capteur gyroscopique

Le capteur gyroscopique mesure une vitesse angulaire en $deg.s^{-1}$ jusqu'à 300 fois par seconde. Il peut mesurer cette vitesse à 360° dans un plan xy [2]. Il servira à mesurer l'inclinaison du prototype afin d'en gérer l'équilibre.

La difficulté d'utilisation de ce capteur vient du fait qu'il faut le calibrer et que cette valeur de calibration « dérive » au court du temps ; elle ne reste pas constante mais est influencée par les mesures. Il est donc nécessaire de stabiliser la capteur lors de la calibration (voir 5.2.1) et de pondérer la valeur de calibration avec la mesure courante (voir 5.2.2).

2.4 Moteur

La brique NXT peut contrôler jusqu'à trois moteurs avec une précision au degré près. Ceux-ci sont en fait à la fois des moteurs, mettant en jeu huit engrenages différents (voir figure 2.2), ainsi que des capteurs puisqu'il y a un module (voir figure 2.3) permettant de compter le nombre de rotations

effectuées par le moteur via un capteur photosensible. Leurs utilisations sont bien entendu la propulsion du prototype ainsi que sa direction.

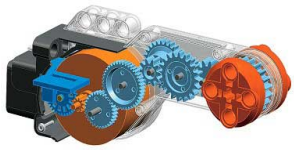

FIGURE 2.2 – Moteur NXT : vue interne [12]

FIGURE 2.3 – Moteur NXT : capteur de rotation [10]

Le graphique 2.4 reprend les mesures du nombre maximal de rotations par minute (rpm) pour un moteur en fonction du rapport entre sa puissance et sa puissance maximale. Ces mesures ont été effectuées avec une batterie chargée délivrant une tension d'environ $8V$, le moteur tournant sans contrainte extérieure.

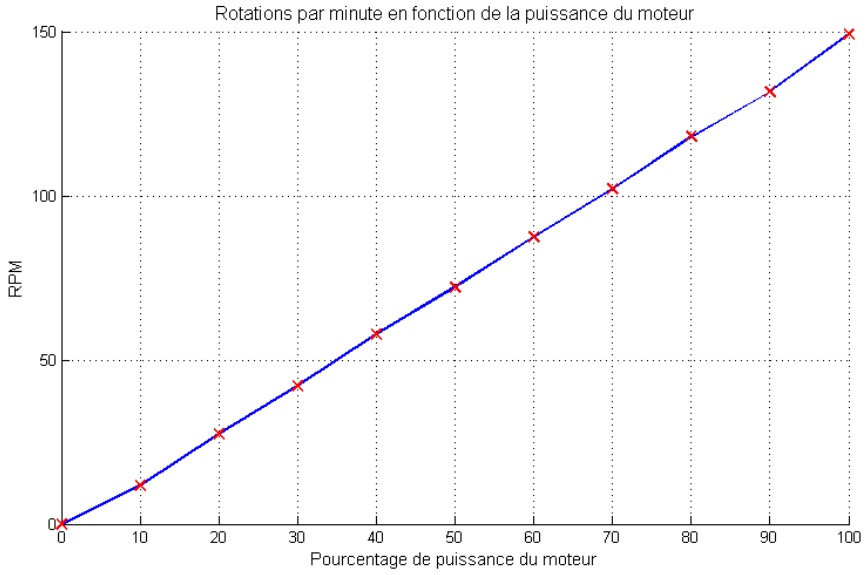

FIGURE 2.4 – Rotations par minute en fonction de la puissance

3. Programmation

3.1 Langage

Il existe de nombreuses façons de programmer la brique Lego Mindstorms NXT. Pour ne citer que les plus connues :

- NXT-G ;
- LeJos ;
- NXC ;
- RobotC ;
- NBC.

L'un des premiers choix dans ce projet a été de déterminer celui qui permettrait de répondre le plus facilement et le plus efficacement au cahier des charges. Il a donc fallu les comparer à plusieurs niveaux (ergonomie, documentation, précision, performances, etc.) en commençant par les analyser séparément.

Pour tous ces langages, le compilateur respectif se charge de compiler le code source en un bytecode (fichier *.rxe*) qui sera envoyé à la brique pour y être interprété lors de son exécution.

3.1.1 NXT-G

Ce premier langage est une méthode de programmation graphique développée par National Instrument (éditeur du logiciel professionnel LabView, très utilisé en industrie) et est fourni avec le kit Lego Mindstorms. Ce programme adopte une méthode graphique de présentation du langage en proposant différents blocs représentant des fonctions à assembler pour produire un algorithme de contrôle. Le but avoué de ce logiciel est de permettre à toute personne n'ayant jamais programmé de faire fonctionner son robot. De ce fait, l'approche en est très simple et pratique pour tester les différents capteurs et comportements de la brique, mais n'est pas suffisante pour un contrôle précis du prototype.

En effet, si le panel de fonctions de contrôle est relativement bien fourni, il manque de nombreuses opérations mathématiques plus ou moins complexes et les modes de réception des données venant des capteurs sont assez limités.

3.1.2 LeJos

LeJos [6] est une alternative open source permettant de coder en Java. Il permet d'utiliser des tableaux multi-dimensionnels, la division en tâches et la gestion des exceptions, une programmation orientée objet (POO) et la possibilité d'utiliser la récursivité. Néanmoins, ces deux derniers points ne sont pas nécessaires dans le contrôle du robot, diminuant de fait l'intérêt de LeJos. Si la communauté gravitant autour de ce langage est relativement active, la documentation disponible est pourtant relativement restreinte.

De plus, l'utilisation de ce langage nécessite l'installation d'un firmware modifié sur la brique pour lui permettre d'interpréter le bytecode produit, ce qui est rédhibitoire.

3.1.3 NXC

NXC (*Not eXactly C*) est un langage open source de haut niveau similaire au C et fonctionnant avec le compilateur de NBC (voir section 3.1.5) [7]. Ses avantages sont un environnement de programmation (Bricx Command Center) complet et performant, une documentation à la fois variée, sérieuse et à jour, sans compter que c'est un langage largement utilisé.

Bricx Command Center comporte aussi de nombreux outils tels qu'une brique virtuelle permettant d'interagir avec la brique réelle sans y toucher ainsi qu'un explorateur de fichiers, mais pèche cependant par un éditeur trop basique.

3.1.4 RobotC

Il existe un autre langage proche du C permettant d'interagir avec la brique : *RobotC*. Ce langage nécessite un firmware optimisé qui améliore les performances de la brique NXT. Mais le fait que la licence ne soit pas libre a posé problème ; la licence la moins chère s'élève à 49$, mais le budget est limité à 40€.

3.1.5 NBC

NBC (*Next Byte Codes*) est quant à lui un langage open source de bas niveau semblable à l'assembleur et est développé par John Hansen, tout comme NXC [7]. Bien qu'il produise un bytecode très léger et

un contrôle avancé du comportement de la brique (même si c'est du même ordre que NXC et LeJos), il serait inutile de se compliquer la tâche en devant apprendre l'assembleur alors que NXC offre les mêmes possibilités tout en étant plus compréhensible.

3.1.6 Choix du langage

Le tableau 3.1 est un récapitulatif des langages présentés.

Langage	Avantages	Inconvénients
NXT-G	Prise en main	Taille du bytecode compilé Précision des capteurs Opérations mathématiques pauvres
LeJos	Communauté active	Connaissance du Java nécessaire Firmware modifié nécessaire
NXC	Langage proche du C BricxCC Régulièrement mis à jour	Éditeur par défaut basique
RobotC		Licence payante
NBC	Bytecode compilé léger	Connaissances en Assembleur nécessaires

TABLE 3.1 – Comparaison des langages

Le choix de NXC pour programmer la brique NXT s'impose donc.

3.2 Gestion des capteurs

L'essentiel dans la gestion des capteurs par la brique est la valeur retournée. L'un des avantages de NXC est de proposer différents modes d'initialisation, chacun pouvant renvoyer une valeur différente pour une même mesure. Le tableau 3.2 les reprend avec l'intervalle de réponse possible.

Mode	**Valeur de retour**
RAW	Valeur comprise entre 0 et 1083
BOOL	Booléen
PERCENT	Valeur comprise entre 0 et 100
ROTATION	Mode propre au capteur de rotation
EDGE	Compte le nombre de transitions entre une valeur basse et une valeur élevée
PULSE	idem EDGE, mais uniquement les transitions de basse vers élevée

TABLE 3.2 – Modes d'initialisation des capteurs en NXC

Par défaut, le capteur photosensible est réglé en mode *PERCENT*. En le réglant en mode *RAW*, il aura droit à un plus grand intervalle de réponse (d'environ 300 à 800) et donc à une meilleure précision.

3.2.1 Moteurs

Concernant les moteurs, il existe plusieurs façons de base de les manipuler [8].

- Mode rotation : contrôle au degré près.
- Mode continu : contrôle de la puissance entre -100 % et 100 %
- Comme capteur de rotation, en comptant le nombre de rotations ou de degrés effectués par le moteurs. (Le capteur est intégré au mécanisme du moteur, voir figure 2.3)

On pourrait croire qu'un contrôle au degré près serait idéal pour contrôler la roue avant lors de ses déplacements, mais il n'en est rien. Si les méthodes internes de contrôle des moteurs ne sont pas correctement documentées, le contrôleur semble cependant marquer une pause après l'exécution de la commande de rotation d'un nombre fixé de degrés. Cet arrêt momentané du moteur induit des saccades dans le mouvement du moteur, rendant toute fluidité des mouvements impossibles avec cette méthode de contrôle.

Ceci explique pourquoi le mode continu sera privilégié tout au long du contrôle du robot, que ce soit pour la gestion de l'équilibre ou du suivi du circuit.

4. Construction du robot

La conception d'un prototype efficace et adapté à notre projet nécessite une bonne compréhension des motos en général. Cependant, une modélisation trop poussée serait superflue et nous avons choisi de gérer l'équilibre de façon empirique, à force de tests et réglages divers. Dans cette partie nous décrirons d'abord les différents paramètres que nous avons considérés pour construire une moto dont l'équilibre peut être géré efficacement.

4.1 Principaux paramètres

Le modèle que nous considérerons ne suppose aucun rôle pour le poids du conducteur dans la gestion de l'équilibre. Nous nous inspirons de la description faite par Vittore Cossalter dans *« Motorcycle Dynamics »* [1]. Celui-ci sélectionne certains paramètres comme étant ceux qui décrivent le mieux une moto (tels que l'angle de la chasse, l'empattement, l'angle de la fourche, etc.). Ces paramètres et leur influence sont également décrits sur certains sites de constructeurs, tels que celui de Yamaha [15]. Cependant, nous ne sélectionnerons, parmi ces paramètres, que ceux qui nous serviront effectivement à construire une moto la plus stable et la plus maniable possible. Les éléments permettant de décrire la moto complète sont repris dans le tableau 4.1 et représentés sur la figure 4.1.

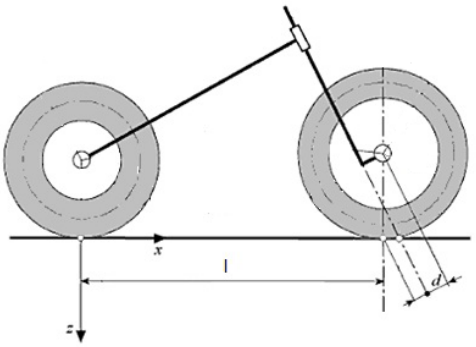

Figure 4.1 – Modèle simple d'une moto

Symbole	Nom	Explication/Complément
l	Empattement	Distance entre les roues. Influe sur la stabilité et la manœuvrabilité.
d	Chasse et fourche	Influent sur la stabilité et la maniabilité de la moto.

TABLE 4.1 – Légende de la figure 4.1

4.2 Brève étude des forces agissant sur la moto

Avant de commencer la construction de notre robot, il est utile de faire une étude des phénomènes physiques interagissant avec celle-ci. La compréhension de ceux-ci nous permettra de prendre les meilleures décisions possibles quant aux différents paramètres à fixer durant la construction. C'est pourquoi nous avons en premier lieu effectué un bilan succinct des forces liées à notre robot motard.

4.2.1 Poids

Le poids est une force qui s'exerce à distance et dont la résultante s'applique au centre de masse [3].

$$\overrightarrow{P} = mg\overrightarrow{1_z}$$

Lors d'un déséquilibre de la moto, le poids créera un moment de force avec le sol, ce qui aura pour effet de faire chuter la moto. Plus le centre de masse de celle-ci sera élevé, plus ce moment sera important. C'est un aspect que nous devrons considérer lors de la construction.

4.2.2 Force de glissement aérodynamique

Lorsque la moto se déplace, elle subit une force de frottement opposée à son mouvement [1]. Dans un milieu tel que l'air, à vitesse faible et avec la géométrie de notre moto, la force de frottement est une fonction linéaire de la vitesse :

$$\overrightarrow{F_d} = -\lambda v\overrightarrow{1_v}$$

Pour un objet important et à haute vitesse, cette force joue un rôle primordial. Cependant, pour une moto comme la nôtre, cette force peut être négligée. En effet, notre vitesse restera faible et notre modèle sera de petite taille. Nous n'aurons donc pas besoin de concevoir une moto aérodynamique.

4.2.3 Force centripète

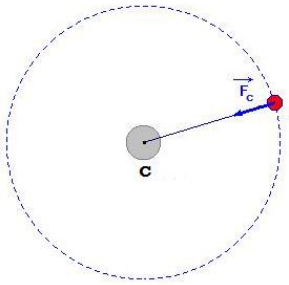

FIGURE 4.2 – Force centripète

La force centripète ($\overrightarrow{F_c}$) apparaît lorsque notre moto tourne et s'applique au centre de gravité [13]. C'est une force dirigée vers le centre de rotation (figure 4.2) qui dépend du rayon de courbure. Il faut noter qu'il existe un effet induit par cette force : l'effet centrifuge. Celle-ci est la force ressentie lors d'un virage. Elle est de même module que la force centripète mais de sens opposé.

$$\overrightarrow{F_c} = m\frac{v^2}{R}\overrightarrow{1_r}$$

C'est cette force qui nous permettra de rétablir la moto en cas de déséquilibre. En cas de chute, il nous suffira de faire tourner la moto du bon côté pour la voir se rétablir. Cependant, il convient de contrôler ce rétablissement. Ceci fera l'objet de tests et d'ajustements (cf. chapitre 5).

4.2.4 Couple gyroscopique

Lorsque que la moto est en mouvement, ses roues sont en rotation. Cette rotation provoque un effet gyroscopique [4], qui a pour effet de stabiliser la moto. Son expression est :

$$\overrightarrow{M} = \Gamma\overrightarrow{\Omega} \times \overrightarrow{\omega}$$

Où Γ est le moment cinétique des roues, $\overrightarrow{\Omega}$ leur vitesse angulaire et $\overrightarrow{\omega}$ la vitesse angulaire de leur support. Ici, on a $\Gamma = \frac{mr^2}{2}$ (moment cinétique d'un disque homogène de rayon r et de masse m) [4].

Nous pouvons dans le cas de notre moto attribuer un ordre de grandeur à ce couple gyroscopique. L'ordre de grandeur du couple gyroscopique, pour une vitesse d'avance telle que la nôtre, serait ainsi de 10^{-3} $N.m$. Si nous comparons cette valeur au couple dû au poids de la moto, nous constatons que nous pouvons négliger l'effet gyroscopique sans hésitation (cf. table 4.2).

Il ne sera donc pas nécessaire lors de la construction de prendre cet aspect en considération.

Force/couple	Ordre de grandeur total
$\Gamma : 8.10^{-6}\ kg.m^2$ $\Omega : 114\ rad/s$ $\omega : 1\ rad/s$	9.10^{-4}
$m : 650\ g$ $P : 6\ N$ $h : 1\ cm$	$6,5.10^{2}$

TABLE 4.2 – Ordre de grandeur du couple gyroscopique

4.3 Conception

Pour aborder la construction de la moto, nous avons ensuite travaillé à partir de paramètres plus spécifiques à notre prototype, en nous basant sur des travaux existants, notamment du livre de Vittore Cossalter, *« Motocycle Dynamics »* [1] (d'où est tiré le tableau 4.3).

Cependant, les paramètres s'influencent les uns les autres. À titre d'exemple, augmenter la manœuvrabilité réduira la stabilité et réciproquement. De plus, nous sommes limités par le kit LEGO et l'agencement des capteurs. Finalement, nous avons pris en compte quatre paramètres que nous jugeons incontournables : la position du centre de masse, la fourche, la chasse et l'empattement (voir tableau 4.3). Ces paramètres sont à prendre en considération de par leur forte influence sur la stabilité et la maniabilité de la moto.

Augmentation de l'empattement	
Effets positifs	Effets négatifs
Meilleure stabilité directionnelle	La moto est plus déformable donc moins manœuvrable
Meilleur transfert du poids pendant les chocs (virage subit, accélération/freinage)	Le rayon de courbure minimum lors des virages augmente, les virages sont plus difficiles à exécuter
Réduction des mouvements parasites dus à l'état de la route (montée, descente, trous ...)	-

TABLE 4.3 – Augmentation de l'empattement

4.3.1 Structure

Nous avons commencé par déterminer la position de la brique, position qui déterminerait la forme globale du robot mais surtout la position de son centre de masse. En effet, la brique représente environ 60 % du poids total de la moto. Nous déduisons d'après notre bilan de force qu'abaisser le centre de masse fait gagner en stabilité car le couple qui fait tomber la moto est ainsi plus petit, et ce sans perte sensible de maniabilité. Nous avons donc décidé de placer la brique le plus près possible du sol, sans que celle-ci ne gêne les mouvements de la moto. Ensuite, nous avons placé sur la brique les moteurs muni des roues avant et arrière, et avons fixé les capteurs.

4.3.1.1 Roue avant : direction

Afin d'équilibrer le poids du prototype, nous avons décidé de mettre le moteur avant de manière symétrique sur la moto. Nous avions donc besoin de changer l'axe de rotation du mouvement. Pour élaborer ce système, que nous appellerons « système de transmission », nous avons construit plusieurs mécanismes, jugés non fiables en raison d'un jeu trop important. Finalement, le système de transmission choisi fait appel à des axes en croix permettant la conversion d'un axe de rotation en un autre axe perpendiculaire (voir figure 4.3). Ce système, bien que comportant encore un peu de jeu dû aux composants Lego , s'est avéré satisfaisant lors des expérimentations.

Parallèlement à cela, nous avons associé des engrenages de différentes tailles pour qu'un tour de moteur corresponde à 2/3 de tour pour la roue avant. Ainsi, nous avons un meilleur contrôle de la direction de notre roue avant par rapport à un système dénué d'engrenages.

De plus, nous avons veillé à avoir une chasse positive et assez grande pour assurer une certaine stabilité à notre prototype.

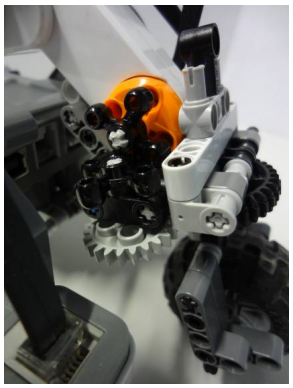

Figure 4.3 – Modèle final de la roue avant

4.3.1.2 Roue arrière : propulsion

De même que pour la roue avant, nous avons placé le moteur arrière de façon à garder le centre de masse dans l'axe des roues. Nous avons donc centré et collé le moteur à la brique. Cependant, en procédant de cette façon, il nous a fallu élaborer un système de transmission pour placer la roue arrière dans l'alignement du moteur. Ce système de transmission est une simple association d'engrenages.

En associant les engrenages, nous avons également pu augmenter la vitesse du prototype. En effet, en exploitant les différentes tailles d'engrenages, nous avons réussi à ce qu'un tour de moteur corresponde à 1,65 tour de la roue avant. Nous avons pensé à encore augmenter ce rapport, mais le couple du moteur le limite à un certain palier, proche de ce que nous avons obtenu. Pour l'augmenter encore, il aurait fallu faire appel à un deuxième moteur, ce que nous n'avons pas fait pour éviter un surpoids et des problèmes de synchronisation.

FIGURE 4.4 – Modèle final de la roue arrière

4.3.1.3 Position des capteurs

La position des capteurs est un élément clé pour le contrôle de l'équilibre et le suivi de ligne. Pour l'équilibre, il faudrait que ces capteurs soient placés de manière symétrique sur la moto. Le capteur gyroscopique doit, de plus, être le plus solidaire possible du corps de la moto (figure 4.5). En effet, s'il vibrait individuellement, les grandeurs mesurées pour l'inclinaison seraient faussées.

Pour le suivi de ligne, c'est le positionnement des capteurs photosensibles par rapport à la ligne qui est importante (voir section 6.2).

FIGURE 4.5 – Capteur gyroscopique

4.3.2 Assemblage

Nous avons ensuite procédé à l'assemblage des différentes parties (brique, système de la roue avant, système de la roue arrière et capteurs). C'est ici que les caractéristiques dues à l'empattement, l'angle d'inclinaison de la fourche et du centre de masse ont été considérées. Nous cherchions à avoir un empattement assez grand pour pouvoir maintenir facilement la moto en équilibre. Les détails du kit LEGO et la recherche d'un compromis stabilité/manœuvrabilité nous ont fait fixer l'empattement de la moto à 27 cm. Ceci fait, nous avons ajusté l'angle de la fourche : en effet, toujours d'après Vittore Cossalter, plus la fourche est inclinée, plus la moto est stable, moins elle tourne facilement (on peut d'ailleurs noter le lien direct avec les caractéristiques de l'empattement).

Il a également été nécessaire de régler l'inclinaison des bras soutenant les engrenages pour régler la hauteur de la moto par rapport à la route. Le but était d'avoir une moto la plus basse possible pour augmenter sa stabilité, tout en évitant qu'elle ne frotte le sol dans un tournant.

4.4 Évolution du prototype

Depuis le premier prototype conçu d'après les études théoriques, plusieurs problèmes ont été détectés pendant les tests et leur résolution a conduit à des modifications majeures sur la moto. La démarche pour améliorer le prototype a été d'essayer de déterminer l'influence des différents paramètres sur le mouvement du robot lors des tests. Ainsi, les premiers problèmes soulevés étaient dus à l'agencement même des pièces entre elles : il y avait un trop grand jeu dans les roues avant et arrière. Les deux systèmes ont donc été solidifiés pour obtenir une plus grande précision directionnelle. Ceci a permis à la moto de s'équilibrer sur une plus grande distance. Après cela, les choix faits sur le compromis entre stabilité et maniabilité ont dû être revus. En effet, la moto ne tournait pas assez et l'équilibre n'était

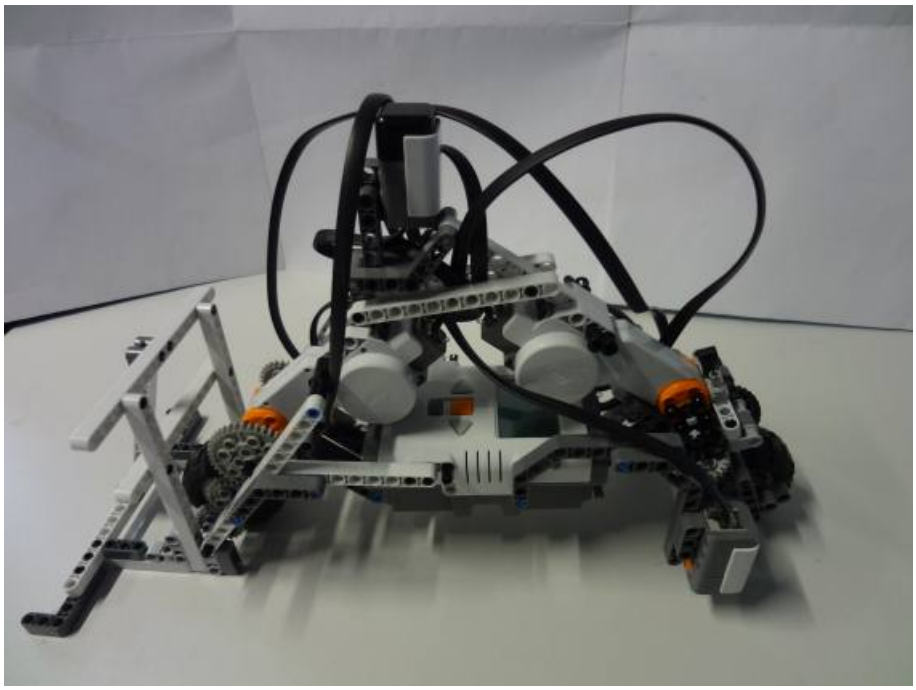

FIGURE 4.6 – Modèle final de la moto

plus récupérable à partir d'un angle de chute faible. Ainsi, l'empattement et l'angle de la fourche ont été diminués et le centre de masse a été rehaussé. Toutes ces modifications, effectuées consécutivement à des tests et des analyses, ont permis au prototype de rester de plus en plus longtemps en équilibre. Finalement, la moto est arrivée à un équilibre quasi permanent.

5. Gestion de l'équilibre

Gérer l'équilibre est sans doute la tâche la plus difficile à obtenir. Ce dernier a donc été mis en place en premier lieu indépendamment du suivi de ligne.

5.1 Système PID

5.1.1 Introduction

Les deux fonctions principales de notre projet utilisent le principe de boucle de rétroaction. En voici le principe. Une mesure donnée par un capteur induit une modification à apporter sur le comportement via un appareil de correction. Chaque modification est suivie d'une nouvelle mesure. Ce principe est montré à la figure 5.1.

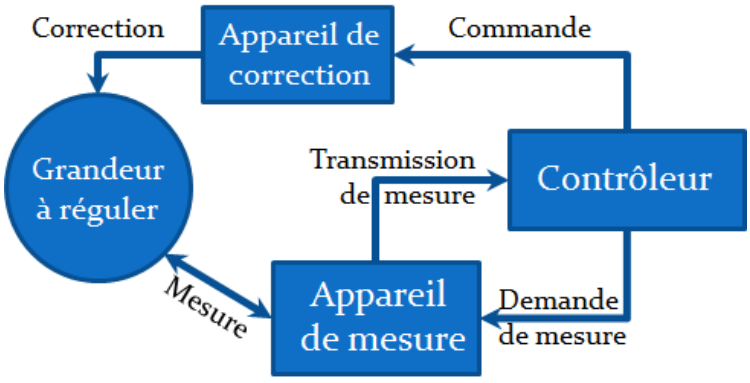

FIGURE 5.1 – Schéma d'une boucle de rétroaction

Le système PID (Proportionnel-Intégral-Dérivé) [5] est un système permettant de déterminer la commande à transmettre à l'appareil de correction.

Un système PID est basé sur une grandeur à réguler [14]. Pour la partie gestion de l'équilibre (pour la partie suivi de ligne voir le chapitre 6), la grandeur à réguler est l'angle que fait la moto par rapport à un axe vertical. Cependant le capteur gyroscopique fourni mesure une vitesse angulaire et non l'angle d'inclinaison de la moto (voir chapitre 2). Après des essais infructueux pour déduire l'angle

de la moto depuis la valeur de la vitesse angulaire (les fonctions utilisées perdaient très rapidement en précision) il a été décidé de travailler directement avec la vitesse mesurée. L'erreur à annuler est la valeur fournie par le capteur : une moto à vitesse angulaire nulle correspond à une moto stable. L'appareil de correction visible sur le schéma de la boucle de rétroaction (voir figure 5.1) est le moteur contrôlant la roue avant. Ainsi pour rattraper une chute vers la gauche, la roue avant de la moto doit tourner vers la gauche à une vitesse déterminée par le système PID.

5.1.2 Principe et application du PID

La commande du système est mis en place par l'addition de différents termes [5] (Proportionnel, Intégral et Dérivé). Nous allons détailler ces différents termes et expliquer comment ils ont été mis en place

5.1.2.1 Gain Proportionnel

Le gain, ou terme, proportionnel consiste simplement en une multiplication de l'erreur mesurée par un coefficient (nommé P). C'est ce terme qui a le plus d'influence sur l'équilibre de la moto. Mais avec un terme proportionnel seul il est seulement possible d'avoir une oscillation de plus en plus forte du modèle pour aboutir à la chute [14].

Ce principe nous a permis de déterminer le coefficient P par dichotomie en encadrant le terme optimum entre des valeurs trop importante et trop faible de P. Pour cette première estimation nous avons simplement observé la chute de la moto : une chute brutale indique un coefficient P trop fort et une chute sans réaction de la moto est due à un coefficient P trop faible.

Le résultat de cet encadrement se trouve dans le tableau 5.1. Dans ce tableau se trouve une information supplémentaire (qui n'est pas un critère en soi, mais une indication sur le comportement de la moto). En effet les moteurs du kit fourni possèdent une vitesse de rotation limite (100%). Ainsi, si le moteur de la roue avant atteint cette valeur maximale, il est possible d'en conclure que le coefficient n'est pas adapté.

Ainsi la valeur du coefficient P se trouve entre 1.5 et 2.0. Pour déterminer la valeur exacte de P il a fallu faire des tests plus précis sur des valeurs comprises entre ces deux bornes. Le caractère aléatoire du comportement de la moto doit être pris en compte : des critères objectifs pour quantifier la réussite d'un lancé ont été mis en place. Les deux critères ayant retenu notre attention sont le nombre de fois ou la moto a su rattraper son équilibre ainsi que le temps durant lequel la moto a su tenir en équilibre. Pour minimiser l'impact du caractère aléatoire de chaque lancé les test ont été effectués 10 fois pour chaque valeur de coefficient et ce sont les moyennes des différents tests qui sont comparées. La figure

Valeur de P	Vitesse maximum dépassée	Appréciation
0.3	non	Réponse bien trop faible
0.7	non	Réponse bien trop faible
1	non	Réponse bien trop faible
1.5	non	**Réponse trop faible**
2	non	**Réponse trop forte**
2.5	oui	Réponse bien trop forte
3	oui	Réponse bien trop forte

TABLE 5.1 – Encadrement Grossier de la valeur de P

5.2 regroupe nos résultats.

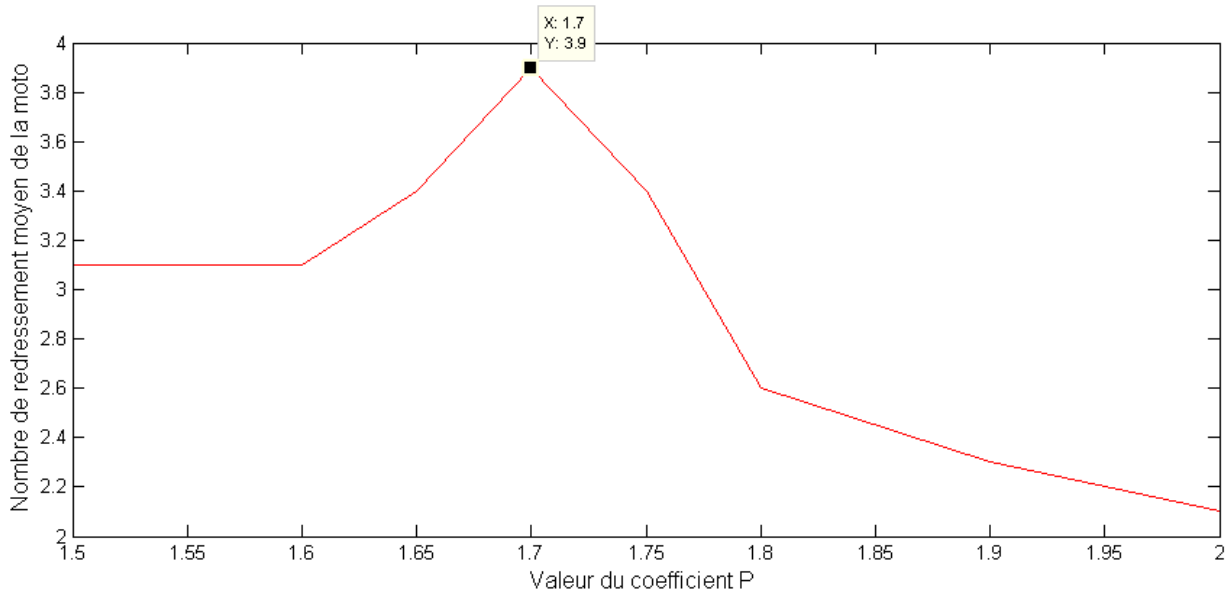

FIGURE 5.2 – Graphique représentant le nombre moyen de rattrapage en fonction de P sur un total de 10 lancés

La valeur finale de notre coefficient proportionnel est donc de 1.7. A ce stade notre commande est :

$$\text{Vitesse} = P.e(t)$$

avec e(t) la mesure de vitesse angulaire au temps t.

Durant notre projet la valeur du coefficient P a varié sans raison apparente (aucune modification de la structure de la moto et aucun changement de programme) nous laissant perplexe face à de nombreux échecs incompris. Nous avons finalement mis en évidence que les fluctuations du niveau de la batterie

au dessus de la tension minimale de 7,2 V influent énormément sur le comportement des moteurs. Nous essayons donc de toujours travailler avec des piles chargées au maximum mais il reste toujours une indétermination.

5.1.2.2 Gain Intégral

Le terme intégral permet de tenir compte de l'historique des erreurs et donc de l'inclinaison de la moto depuis son départ. Ce terme permet une réduction de l'amplification des oscillations (dues à une erreur résiduelle appelée erreur statique) présente avec un terme proportionnel seul [5]. Le gain consiste en l'intégrale (donc la somme infinitésimale) des erreurs mesurées depuis le départ multipliée par un coefficient I.

Habituellement, le terme Intégral est multiplié par la somme de toutes les erreurs précédentes [14]. Cette méthode ne semble pas fonctionner dans notre cas : l'équilibre de la moto ne s'améliore pas. Une solution est de ne prendre en considération qu'un nombre défini d'erreurs antérieures et de les stocker dans un vecteur. Ce système est concluant mais la taille du vecteur est un paramètre supplémentaire à régler. Après avoir effectué divers essais avec un coefficient intégral I arbitraire la taille du vecteur à été fixée à 300 termes. La boucle de gestion de l'équilibre tourne en approximativement 3 ms. Le vecteur est donc rempli puis complètement renouvelé toutes les 900 ms.

La taille du vecteur fixé nous avons pu déterminer notre coefficient I. L'influence du terme intégral est plus difficile à interpréter physiquement lors d'une chute de la moto et nous n'avons pu trouver aucune information/méthode du type Zygler-Nichols [11] adaptée à notre problème. Ainsi, il a été décidé de travailler de la même façon que pour le coefficient P : une valeur grossière a d'abord été encadrée puis une valeur plus exacte de I a été déterminée.

Le critère utilisé pour comparer les différentes valeurs de I est le temps pendant lequel la moto conserve son équilibre. L'inconvénient de ce critère est qu'il ne prend pas en compte les oscillations de la moto. En effet certaines valeurs de I font rouler la moto quasiment droite alors que d'autres impliquent des oscillations très importantes.

Un autre problème est la place mise à disposition pour les tests : pour avoir un réglage plus fin il serait nécessaire de faire rouler la moto sur de plus grandes distances. Le graphique 5.3 reprend le nombre moyen de secondes pour lequel la moto est restée en équilibre en fonction de I. Nous observons que nous ne pouvons fixer qu'une limite supérieure (ce qui est dû au manque de place dans nos salles de test). Le tableau 5.2 donne des éléments qualitatifs observés lors des tests. Ce tableau a été utilisé pour fixer la valeur finale du coefficient.

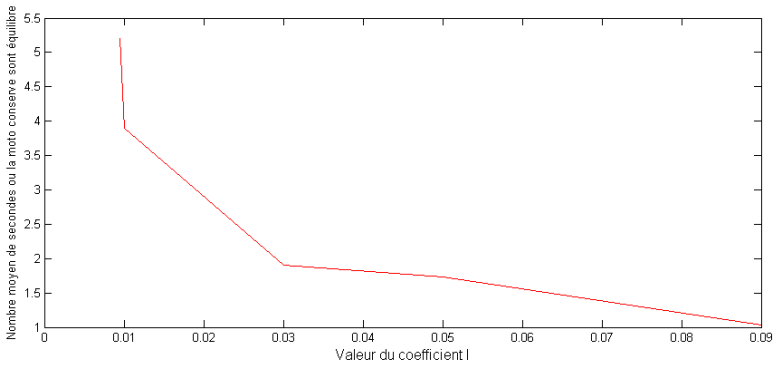

FIGURE 5.3 – Nombre moyen de secondes de maintient de l'équilibre par la moto en fonction de I

Valeur de I	Aspects Qualitatifs
0.009	Réponse nerveuse, si les piles sont très chargées alors chute
0.008	**fonctionne, trajectoire assez droite**
0.007	**fonctionne, trajectoire assez droite**
0.006	**fonctionne, trajectoire assez droite**
0.005	petits zig-zag autour d'une direction globalement droite
0.003	grands zig-zag, cause : le terme I accumule beaucoup de valeurs pour ensuite avoir un effet
0.002	grands zig-zag et environ 1 chute tous les 10 lancés (sur une distance d'environ 5 mètres)
0.0009	la trajectoire n'est plus du tout rectiligne, aucune direction privilégiée et chutes fréquentes

TABLE 5.2 – Aspects qualitatif liés aux valeurs de I

Ainsi le coefficient I est réglé sur une valeur de 0.007. La moto tient maintenant relativement bien en équilibre. Le seul élément a améliorer est de faire rouler la moto sur une ligne aussi droite que possible. Ainsi, nous avons comme commande :

$$\text{Vitesse} = P.e(t) + I.\sum_i e_i$$

où e_i sont les mesures précédentes.

5.1.2.3 Gain Dérivé

Pour parfaire le système PID il faut pouvoir commander une réponse qui tienne compte du rétablissement ou de l'accélération de chute de la moto. Le terme dérivé permet cette anticipation [14]. Ce terme consiste en la multiplication de l'accroissement de l'erreur par un coefficient D. Pour pouvoir appliquer ce gain, l'accroissement de l'erreur au temps t a été supposé égal à l'accroissement de l'erreur au temps t+dt ; l'approximation étant validée par la courte durée de la boucle (3ms).

Après avoir effectué des tests en relevant des données d'ordre qualitative (direction de la moto, réponse à un léger choc...) nous avons fixé le coefficient D à 1. Nous gardons à l'esprit que la valeur de ce coefficient pourrait légèrement varier pour favoriser une intégration du suivi de circuit.

La commande finale est donc :

$$\text{Vitesse} = P.e(t) + I\sum_i e_i + D.(e(t) - e(t - dt))$$

La moto telle qu'elle est calibrée peut être redirigée en donnant de petits coups sur le capteur gyroscopique pour forcer un rétablissement dans la direction voulue.

5.2 Utilisation du capteur gyroscopique

Dans cette section nous nous attarderons sur les détails du code (voir annexe B) indispensables au bon fonctionnement de notre moto. Ces détails peuvent être dus à des contraintes techniques mais aussi à des contraintes supplémentaires créées par l'application du PID.

5.2.1 Calibration du capteur gyroscopique

Le capteur gyroscopique tel qu'il est conçu par HiTechnics [2] (voir section 2.3) doit recevoir le zéro de ses mesures. C'est la partie calibration de notre programme qui permet d'obtenir ce zéro. Pour faire cette calibration nous avons besoin d'un court instant de stabilité avant le lâché de la moto : il a donc été nécessaire de construire une rampe de lancement (voir figure 5.4). Nous relevons ensuite 100 valeurs successives et en calculons la moyenne. L'opération est répétée tant que la moto n'est pas stable. Cette moyenne est le 0 du capteur. (Pour plus de détails voir l'annexe B comprenant notre code)

5.2.2 Prélèvement des mesures

Le 0 de notre capteur fixé il a fallu prélever la valeur de la vitesse angulaire. Deux problèmes sont apparus : il faut être sûr d'utiliser toutes les valeurs fournies par le capteur gyroscopique et ces valeurs

FIGURE 5.4 – Photo de la rampe de lancement

dérivent pour ne plus correspondre à la réalité après très peu de temps. Nous devons donc nous assurer de la prise en compte de toutes les valeurs et de la stabilité de ces valeurs.

Pour obtenir ce résultat nous calculons la moyenne de trois mesures successives (pour résoudre le premier problème). Les trois valeurs successives prélevées sont souvent les mêmes, car le capteur prélève 300 mesures par secondes (voir chapitre 2.3) et cette boucle tourne environ 1 ms. Nous déterminons ensuite un terme qui prend en compte le décalage des mesures et ce terme est soustrait de la moyenne [2].

5.3 Performances

Le robot étant très stable, nous n'avons pas pu le tester dans un environnement suffisamment large pour le pousser à sa limite. Néanmoins, nous avons pu mettre en évidence qu'il est capable de tenir en équilibre sur au moins $13m$, et qu'il parcourt cette distance en $23s$, à une vitesse moyenne de $0,56m.s^{-1}$.

6. Suivi de circuit

Dans cette section du travail seront présentés en plusieurs temps la manière par laquelle le prototype gère le parcours de son circuit, et ceci indépendamment de sa gestion d'équilibre (plein contrôle de la roue avant).

6.1 Gestion du circuit

6.1.1 Gestion statique

Une solution pour permettre au robot de suivre un circuit prédéterminé aurait été d'imposer une commande statique à la roue avant signalant tout simplement de tourner d'un angle ou d'une vitesse définis, à gauche ou à droite, à chaque rencontre de la ligne (le sens étant alors déterminé par le capteur photosensible ayant détecté une diminution d'intensité lumineuse).

Seulement, une telle gestion de l'orientation du robot ne peut convenir car, comme indiqué par la figure 6.1, il ne faut pas réagir de la même manière lorsqu'un capteur lumineux frôle la ligne et lorsqu'il se trouve totalement au-dessus de celle-ci.

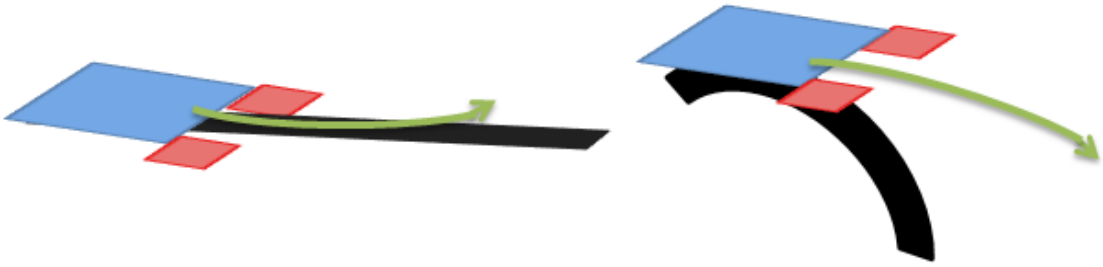

FIGURE 6.1 – Gestion statique de l'orientation de la moto, adaptée à un changement progressif (à gauche), mais pas à un changement brusque (à droite) – en bleu, robot ; en rouge, capteurs photosensibles.

6.1.2 Gestion proportionnelle

Nous avons implémenté une gestion proportionnelle à l'erreur caractérisant le degré de chevauchement de la ligne par ces capteurs.

Cette erreur est obtenue de la manière suivante :

On calcule, pour chacun des capteurs (indépendamment l'un de l'autre), la différence entre la valeur qu'il capte à l'instant donné et sa valeur de calibration (valeur captée sur du blanc en début de course par le capteur en question[1]). Après quoi on détermine l'erreur comme l'écart entre ces deux différences .

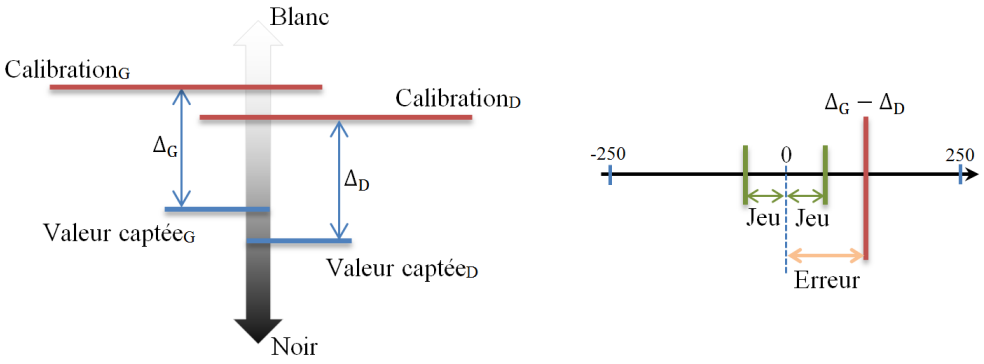

FIGURE 6.2 – Erreur caractérisant le degré de chevauchement d'un capteur sur la ligne.

Cette démarche est récapitulée à la figure 6.2 dans laquelle les indices « G » et « D » correspondent respectivement aux capteurs de gauche et de droite.

Une fois cette erreur obtenue, il convient de traiter l'information en tournant la roue à une vitesse proportionnelle à l'erreur (coefficient de proportionnalité P) si celle-ci est suffisante (plus grande en valeur absolue qu'un jeu arbitraire) pour ne pas être trop affectée par des ombres ou impuretés du circuit.

$$\text{Si } (|\Delta_{Gauche} - \Delta_{Droite}| > \text{Jeu})$$

$$\text{Commande}(t) = P.\text{e}(t)$$

Des tests ont été effectués pour déterminer une valeur du coefficient P de manière qualitative. Les résultats de ces tests sont regroupés dans le tableau 6.1 où une réponse trop faible correspond à un franchissement du tracé du circuit.

[1]Les valeurs de calibration des deux capteurs sont différentes mais leurs échelles de valeurs entre les intensités du blanc et du noir sont identiques.

Valeur de P	Appréciation
0.3	Réponse bien trop faible
0.5	Réponse trop faible
1	Réponse correcte
2	Réponse trop forte
3	Réponse bien trop forte

TABLE 6.1 – Encadrement de la valeur de P

À l'issue de ces tests, la valeur de P a été fixée à 1.

6.2 Place des capteurs photosensibles

Les places des capteurs photosensibles ont été déterminées en fonction de trois paramètres :

- leur position relative au sein de la moto dans l'axe de celle-ci ;
- la distance entre eux ;
- leur hauteur.

En effet, pour leur position relative dans l'axe de la moto, ceux-ci ont été voulus le plus près possible de la roue avant, afin de permettre la meilleure réactivité possible. Des tests ont montrés que si les capteurs sont placés trop à l'avant, la ligne est captée trop tôt, et le reste de la moto est trop en retard pour suivre ; et que si les capteurs sont placés trop à l'arrière, la ligne est captée trop tard, rendant impossible son rattrapage.

De même, la distance les séparant a été déterminée minimale par souci de réactivité, ainsi que pour limiter les problèmes dus à l'inclinaison du robot, une inclinaison pouvant faire varier l'intensité lumineuse détectée (voir section 2.2) ou encore faire détecter une mauvaise partie du circuit. Des détails pratiques ont aussi été pris en compte comme la largeur de notre circuit ou le gène que peuvent occasionner les capteurs lorsque la roue avant de la moto est en mouvement ; ce qui impose que la distance entre les capteurs soit de 11 cm (figure 6.3).

Enfin, pour leur hauteur, celle-ci a été égalée à la hauteur optimale pour la détection d'intensité lumineuse (voir section 2.2), c'est-à-dire 2 cm de haut.

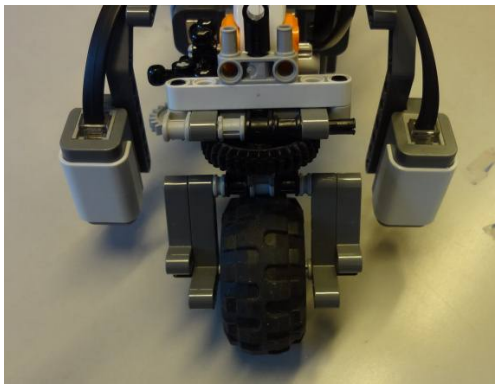

FIGURE 6.3 – Position des capteurs photosensibles

6.3 Tracé du circuit

La manière la plus simple d'élaborer un circuit est de tracer une ligne noire sur un fond blanc (voir figure 6.4). Ce système n'est pas compatible avec une gestion proportionnelle de la commande, car il cause une détection trop brusque de la ligne, et ce quelle qu'en soit sa largeur.

FIGURE 6.4 – Ligne noire sur fond blanc

Un dégradé continu (voir figure 6.5) permet de résoudre ce problème, mais s'avère tout aussi problématique car une quelconque variation de hauteur des capteurs, due à une inclinaison du robot provoquerait une commande erronée (voir la section 2.2).

FIGURE 6.5 – Dégradé continu de niveau de gris sur fond blanc.

Dès lors, il convient d'effectuer un compromis de ces deux types de tracés et de tracer un circuit en dégradé discret (voir figure 6.6).

En pratique, nous posons pour ce faire plusieurs lignes de rubans adhésifs de couleurs différentes

FIGURE 6.6 – Dégradé discret

sur du papier blanc, en l'occurrence : du noir, qui réfléchit une intensité lumineuse de 290 (unités arbitraires), et du bleu, d'intensité 370 – le blanc réfléchissant quant à lui une intensité de 530.

L'emploi de papier collant bleu plutôt que gris se justifie par une différence de prix dans le marché ; le résultat restant inchangé étant donné que les capteurs ne captent pas directement de couleurs mais une intensité lumineuse.

6.4 Performances

Le suivi de circuit n'est pas encore parfaitement abouti, mais son avancement a néanmoins permis de procéder à des mesures le long du circuit représenter à la figure 6.7.

Ce court circuit de $1,4$ m est parcouru à vitesse constante de $0,23 m.s^{-1}$, soit en $6s$.

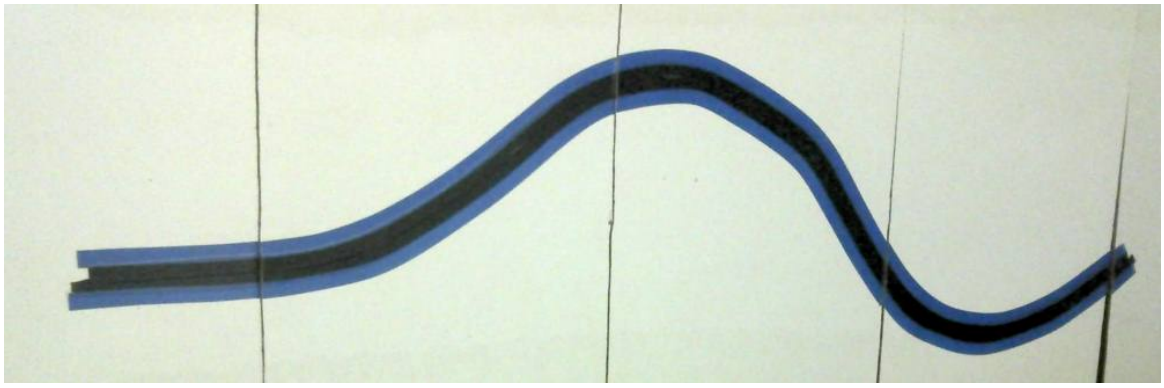

FIGURE 6.7 – Circuit permettant une mesure de performances

7. Mise en commun des deux fonctions

Pour mettre en commun l'équilibre et le suivi de ligne nous avons envisagé et testé 2 types de solutions : avec et sans multitâche.

7.1 Sans Multitâche

L'utilisation du multitâche implique un contrôle partagé de la roue avant par la gestion de l'équilibre et le suivi de ligne. Un tel système n'est pas optimal : la mise en commun des deux fonctions pourrait créer des interférences et le programme pourrait ne plus être opérationnel. Nous avons donc tenté dans un premier temps d'éviter ce multitâche. Voici les idées qui ont été développées.

7.1.1 Balancier

Une de ces idée, développée puis testée par le groupe, a été de créer un déséquilibre à l'aide du troisième moteur jusqu'alors inutilisé. Ce déséquilibre pousse la moto à se rattraper en tournant la roue avant. Si la direction de la roue avant varie, la trajectoire de la moto est modifiée. L'information qui contrôle le déséquilibre introduit vient des capteurs photosensibles et le contrôle du suivi de ligne ne s'applique plus directement sur la roue avant.

Pour appliquer une telle idée, le moteur est placé verticalement et utilise son poids pour déséquilibrer la moto (voir figure 7.1).

Des tests ont été effectués et ce système a dû être écarté : l'ensemble est bien trop sensible et le réglage du PID associé compliqué. La moto tombe trop vite, et ce, même en maintenant le moteur dans sa position d'équilibre. En effet, le jeu dans le moteur est trop grand pour commander avec précision la position de celui-ci. Enfin, le moteur semble trop lourd et son action est bien trop importante, ce qui cause une chute de la moto.

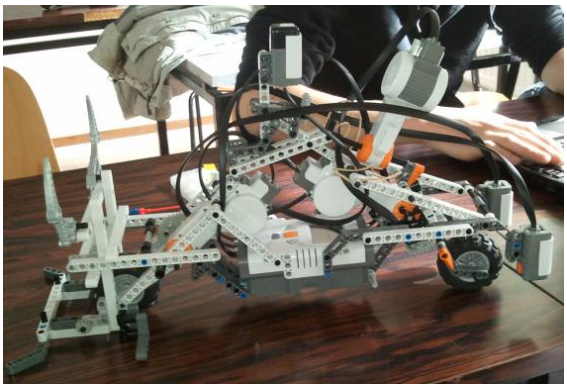

FIGURE 7.1 – Prototype de la moto modifié pour utiliser un balancier

7.1.2 Boucle simple

Une autre option qui permet d'éviter le multitâche est de concevoir l'entièreté du programme dans une seule boucle infinie. Les diverses commandes interviennent alors successivement et il ne peut y avoir d'interférence. Le seul paramètre fixé serait le temps pris par la boucle pour s'exécuter. Cette solution a été testée dans le cadre du suivi d'une ligne droite en combinant les résultats obtenus dans le chapitre sur la gestion de l'équilibre (voir chapitre 5) ainsi que ceux obtenus dans le chapitre sur le suivi de circuit (voir chapitre 6). Ces tests n'ont pas abouti à des résultats convaincants : les ordres contradictoires que reçoit la roue avant empêche toute action et la moto chute dès qu'elle rencontre la ligne.

7.2 Avec Multitâche

Le principe du multitâche est de lancer toutes les tâches utilisées dans le programme puis de limiter les phénomènes d'interférence entre ces différentes tâches. Pour exploiter ce concept, deux idées ont été évoquées : un contrôle à l'aide de l'introduction d'un déséquilibre via des coups de volant ou bien simplement en demandant à la roue avant de tourner vers la ligne.

L'outil fourni par NXC pour implémenter le multitâche est la variable "mutex" [9]. Le "mutex" est l'équivalent d'un bâton de parole. Il est déclaré en variable globale puis chaque fonction réclame son utilisation. Lorsque la fonction reçoit le mutex elle poursuit son exécution, pour le relâcher en fin de boucle.

7.2.1 Introduction d'un déséquilibre

L'idée reprise ici repose sur le même principe que le balancier (voir section 7.1.1) : gérer le suivi de ligne via la fonction de gestion de l'équilibre. Un coup de volant est introduit pour déséquilibrer la moto, qui entraîne une rotation de la roue avant grâce à la fonction de gestion de l'équilibre.

L'implémentation de ce système est problématique : les détails techniques de NXC et de la brique ne permettent pas de demander à la roue avant de tourner d'un angle précis. Il est seulement possible de tourner brutalement dans un sens pendant un court instant puis dans l'autre sens pour revenir à la position initiale. Un tel système ne peut pas être précis ; ce que des tests ont confirmer.

7.2.2 Combinaison de deux fonctions indépendantes

L'idée est ici de prendre les deux fonctions principales (équilibre et suivi de ligne telles que décrites aux chapitre 5 et 6), de les lancer simultanément et de les contrôler via un mutex. Là aussi des tests ont été réalisés sur le suivi d'une ligne droite. C'est le même problème que dans la section 7.1.2 qui a été rencontré.

8. Travaux futurs

À l'issue de ce projet, certains points peuvent encore être sensiblement améliorés.

8.1 Suivi de circuit

Outre le suivi en lui-même qui demande des optimisations tant au niveau de la vitesse de parcours qu'à la précision du suivi, la prochaine étape pourrait consister en un circuit bien plus compliqué. Il pourrait par exemple comprendre des croisements ou des codes de signalisation dictant la conduite du véhicule.

8.2 Mise en commun équilibre/suivi

La prochaine grande amélioration à apporter au prototype est la fusion de la gestion de l'équilibre avec la gestion du suivi de circuit. Les travaux présentés au chapitre 7 exposent le début de la démarche entreprise. Pour de futurs travaux, il reste des pistes à explorer concernant cette gestion combinée. On peut par exemple envisager une modification de la valeur de calibration du capteur gyroscopique par la partie suivi du circuit afin de provoquer un léger déséquilibre dans le sens désirer, ou encore une atténuation de la commande envoyée à la roue avant par la même partie suivi du circuit dans le même but de volontairement déséquilibrer le robot.

Néanmoins, de telles améliorations sont à portée puisque les problèmes ne viennent pas de la structure du prototype ; les capteurs photosensibles sont par exemple correctement placés pour pouvoir suivre une ligne sans gêner la gestion de l'équilibre.

9. Conclusion

Au terme de ce projet, nous avons atteint certains des objectifs qui nous semblaient primordiaux. En effet, nous avons étudié le kit, les langages de programmation, le fonctionnement d'une moto et l'implémentation d'un contrôle de régulation. Ceci nous a permis d'acquérir un bagage théorique suffisant pour aborder le projet d'une manière efficace et scientifique. Ensuite, nous avons testé nos acquis, revu certains de nos choix, et modifié notre prototype et nos programmes à maintes reprises. Nous avons ainsi réussi à agir méthodiquement, en suivant une démarche scientifique à chaque étape du projet ; ce qui nous a permis d'avancer et de résoudre la plupart de nos problèmes, notamment au niveau de l'équilibre où nous avons atteint notre objectif malgré les nombreux obstacles.

Cependant, après ces deux quadrimestres de travail, il convient de prendre conscience des erreurs commises et des objectifs non accomplis. Nous regrettons ainsi le fait de n'avoir su réussir à satisfaire complètement le cahier des charges, étant donné que nous n'avons pas réussir à lier l'équilibre et le suivi de ligne du robot dans les limites de temps imposées. Néanmoins, nous sommes certains que cet objectif pourrait être atteint, au vu des nombreuses pistes envisagées.

En conclusion, nous pouvons être satisfaits du travail effectué et de la méthode rigoureuse que nous avons mise en place. Même si le cahier des charges n'est pas entièrement respecté, nous avons réussi à avoir un robot qui peut d'une part suivre une ligne et d'autre part maintenir son équilibre de manière plus que satisfaisante.

A. Budget

Au sein du cahier des charges de notre projet est stipulé que nos dépenses ne peuvent excéder un total de 40 €.

Les seules dépenses effectuées se retrouvent dans l'adhésif acheté pour tester les différents types de tracés de circuit, et ce pour un montant total de 4,30 €.

Ainsi, nous pouvons constater que, au point de vue du budget, le cahier des charges est pleinement respecté.

B. Code NXC gestion de l'équilibre

Ulysse.nxc

```
1  //Programme final
   //Reglage du controle de la roue avant en fonction de la vitesse de chute
3  //Q. Delhaye, A. Dumont, C. Giaux, P. Lasbleis, G. Roig
   //ULB 2011-2012
5

7  #define OFFSET_GYRO 11
   #define GYRO IN_1
9  #define OFFSET_SAMPLE 100
   #define DECOFFSET 0.00005 //coef de decalage de l'offset
11 #define ROUE_AVANT OUT_A
   #define ROUE_ARRIERE OUT_B
13
   //variable ControleRoueAvant
15 int position=0;
   //variables GetGyroData()
17 int gyroRaw;
   int gyroSpeed;
19 float gOffset;
   //declaration des taches
21 task GetGyroData();
   task ControleRoueAvant();
23 task ControleRoueArriere();
   task SuiviLigne();
25
   void GetGyroOffset()
27 {
     TextOut(0,LCD_LINE1,"CALIBRATION");
29   float gSum;
     int gMin, gMax, g;
31   do{
       gSum = 0.0;
33     gMin = 1000;
       gMax = -1000;
35     for (i=0; i<OFFSET_SAMPLE; i++) {
         g = SensorHTGyro(GYRO);
37       if (g>gMax)
           gMax = g;
39       else if (g<gMin)
           gMin = g;
41       gSum += g;
         Wait(5);//On attend un peu, de toute facon le capteur  ne peut prendre
43               //que 300 mesures par seconde max.
       }
45   } while ((gMax-gMin)>1);
     gOffset = gSum/OFFSET_SAMPLE -1;//avec ce -1, permet de forcer l'offset a sa
```

```
47                                      //valeur une fois moteurs en route.
}
49
task GetGyroData()
51 {
   int tabRaw[] = {0,0,0};
53   while(true){
       tabRaw[0] =  SensorHTGyro(GYRO);
55     tabRaw[1] =  SensorHTGyro(GYRO);
       tabRaw[2] =  SensorHTGyro(GYRO);
57     gyroRaw = (tabRaw[0]+tabRaw[1]+tabRaw[2])/3;
       gOffset = DECOFFSET * gyroRaw + (1-DECOFFSET) * gOffset;
59     gyroSpeed = gyroRaw - floor(gOffset);
       Wait(5);
61   }
}
63
task ControleRoueAvant()
65 {
   int vitesse,n = 300,historique[300],i=0,attenteIntegrale=0,sum=0;
67   float derive;

69   for (int j=0 ; j<n; j++) // Initialisation vecteur historique
       historique[j]=0;
71   // Constantes PID
     float P = 1.7;
73   float I = 0.007;
     float D = 1;
75
     while(true)
77   {
       i++;
79     i=i%n;
       sum-=historique[i];
81     historique[i]=gyroSpeed;
       sum+=historique[i];
83     derive=historique[i]-historique[(i+n-1)%n];

85     if(gyroSpeed > 4 || gyroSpeed < -4){
         vitesse = P*gyroSpeed+I*sum+D*derive;
87       if(vitesse >= 100)
           vitesse = 99;
89       OnFwd(ROUE_AVANT, vitesse);
         Wait(2);
91     }
       else if (gyroSpeed <= 4 || gyroSpeed >= -4)
93       Off(ROUE_AVANT);
       attenteIntegrale++;
95     Wait(2);
   }
97 }

99 task ControleRoueArriere()
   {
101  while(true) {
       OnFwd(ROUE_ARRIERE, 90);
103    Wait(2);
   }
105 }
```

```
107 task Affichage()//Affichage de controle des valeurs
    {
109     while(true) {
            ClearScreen();
111         NumOut(0,LCD_LINE4, position);
            TextOut(0,LCD_LINE5,"gSpeed: ");
113         NumOut(40,LCD_LINE5, gyroSpeed);
            TextOut(0,LCD_LINE7,"Raw: ");
115         NumOut(25,LCD_LINE7, gyroRaw);
            NumOut(0,LCD_LINE8, floor(gOffset));
117         Wait(5);
        }
119 }

121 task main()
    {
123     if (BatteryLevel() >= 7200)
            PlayTone(440, 100);
125     Precedes(GetGyroData, Affichage, ControleRoueAvant, ControleRoueArriere);
        SetSensorHTGyro(GYRO);
127     GetGyroOffset();
    }
```

C. Code NXC gestion du suivi de circuit

Goldorak2.nxc

```
//Reglage du controle de la roue avant en fonction l'intensite captee par les
//capteurs photosensibles
//Q. Delhaye, A. Dumont, C. Giaux, P. Lasbleis, G. Roig
//ULB 2011-2012


#define GAUCHE IN_2
#define DROITE IN_3
#define OFFSET_SAMPLE 100
#define ROUE_AVANT OUT_A
#define ROUE_ARRIERE OUT_B

int calibG;
int calibD;
int angle;
int commande;

task SuiviLigne() // suppose la ligne strictement entre les capteurs
{
    bool tourne;
    int intensiteG,intensiteD,vitesse;
    int jeu = 20;
    float Pl=1;//coeff prop capteurs lumineux
    const int angleMax=40;
    ResetRotationCount(ROUE_AVANT);

    while(true)
    {
        angle=MotorRotationCount(ROUE_AVANT);
        tourne = false;
        intensiteG=SENSOR_2;
        intensiteD=SENSOR_3;
        commande=(calibG - intensiteG)-(calibD -  intensiteD);

        if(commande>jeu || commande<-jeu){
            vitesse=Pl*commande;
            tourne=true;
            if(commande>0 && angle>angleMax)
                tourne=false;
            else if(commande<0&&angle<-angleMax)
                tourne=false;

            if(vitesse>=100)
                vitesse=99;
            if(vitesse<=-100)
                vitesse=-99;
```

```
48            if(tourne){
                  OnFwd(ROUE_AVANT, vitesse);
50                Wait(2);
              }
52            else {
                  Off(ROUE_AVANT);
54                Wait(2);
              }
56        } //fin premier if
          else{
58            Off(ROUE_AVANT);
              Wait(2);
60        }
      }//fin while(true)
62 }

64 task Affichage()//Affichage de controle des valeurs
   {
66    while(true){
          ClearScreen();
68        NumOut(0,LCD_LINE4, angle);
          NumOut(0,LCD_LINE1, commande);
70        Wait(2);
      }
72 }

74 task ControleRoueArriere()
   {
76    OnFwd(ROUE_ARRIERE, 60);
      Wait(2);
78 }

80 void Calibrage()
   {
82    TextOut(0,LCD_LINE1,"Calibration");
      int ti, tc=0;
84    unsigned long sum1 = 0,sum2 = 0;
      int k=0;
86    ti=CurrentTick();

88    while(tc<2000){ //2 secondes de calibration
          sum1 += SENSOR_2;
90        sum2 += SENSOR_3;
          ++k;
92        tc = CurrentTick()-ti;
      }
94    ClearScreen();  //On efface tout

96    calibG = sum1/k; //Valeurs de calibration
      calibD = sum2/k;
98
      TextOut(0,LCD_LINE2,"Calibration GAUCHE:");
100   NumOut(0,LCD_LINE3, calibG);
      TextOut(0,LCD_LINE4,"Calibration DROITE:");
102   NumOut(0,LCD_LINE5, calibD);
   }
104
   task main()
106 {
```

```
    if (BatteryLevel() <= 7200){
108     PlayTone(440, 100);
    }
110 SetSensorType(GAUCHE,SENSOR_TYPE_LIGHT_ACTIVE);//led allumee
    SetSensorMode(GAUCHE,SENSOR_MODE_RAW);//En raw pour une plus grande precision
112 SetSensorType(DROITE,SENSOR_TYPE_LIGHT_ACTIVE);
    SetSensorMode(DROITE,SENSOR_MODE_RAW);
114 Calibrage();
    Precedes(ControleRoueArriere, SuiviLigne, Affichage);
116 }
```

Bibliographie

[1] Vittore Cossalter. *Motorcycle Dynamics*. LULU, s.l., 2006. 372 p.

[2] Dataport Systems, Inc. Hitechnic products. `http://www.hitechnic.com/`. Site WEB sur Internet, consulté le 18-03-2012.

[3] Alain Delchambre. *Mécanique rationnelle I*. Presses Universitaires de Bruxelles, Bruxelles, 2e édition edition, 2009.

[4] Alain Delchambre. *Mécanique rationnelle II*. Presses Universitaires de Bruxelles, Bruxelles, 5e édition edition, 2010.

[5] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feeback control of dynamics systems*. Pearson, s.l., 6th edition, 2010.

[6] Geeknet Inc. Lejos, java for lego mindstorms. `http://lejos.sourceforge.net/`. Site WEB sur Internet, consulté le 02-12-2011.

[7] Geeknet Inc. Nbc - next byte codes, not exactly c, and superpro c. `http://bricxcc.sourceforge.net/nbc/`. Site WEB sur Internet, consulté le 12-12-2011.

[8] John Hansen. Lego Mindstorms *NXT power programming : robotics in C*. variant press, s.l., 2nd edition, 2009.

[9] John Hansen. *NXC Guide*, 2011. `http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf`.

[10] Philippe Hurbain. `http://www.philohome.com/nxtmotor/nxtmotor.htm`. Site WEB sur Internet, consulté le 12-03-2012.

[11] Michel Kinnaert. *Automatique*. 2010.

[12] LEGO Group. Lego mindstorms. `http://mindstorms.lego.com`. Site WEB sur Internet, consulté le 18-03-2012.

[13] Raymond A. Serway and John W. Jewette Jr. *Physics for scientists and engineers*. Cole-Thomson learning, Belmont, USA, 6th edition, 2004.

[14] J. A. Shaw. *The PID Control Algorithm : How it Works, How to Tune it, And How to Use it*. s.l., seconde edition, 2003. 68 p.

[15] Yamaha Motor. Motorcycle workshop. part 1 : Chassis geometry. `http://www.yamaha-motor.eu/designcafe/en/about-design/technology`. Site WEB sur Internet, consulté le 13-03-2012.