Ecole Polytechnique de Tunisie

Application à base de microprocesseur

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Carthage

Rapport

Robot à deux roues
(Pendule inversé)

13 Juin 2014

Elaboré par :

Yassin AYADI
Amen MEMMI
Mahmoud MASMOUDI

Élèves ingénieurs en 2ème année

Année universitaire
2013-2014

REMERCIEMENT

Nous tenons à remercier toute l’équipe administrative de l'École Polytechnique de Tunisie pour avoir assuré le bon déroulement des séances du module « Application à base de microprocesseur ».

Nous remercions énormément Monsieur Adel BENZINA, Madame Jinette AOUIDETE et Madame Wided SOUIDENE pour leur assistance, leur compagnie et leur bonne volonté à guider les étudiants vers la réussite dans leurs projets.

Table des matières

Introduction ____ 5

Côté matériel ____ 6

A. Composants utilisés ____ 6

B. Représentation du circuit ____ 9

Côté logiciel ____ 10

A. Matlab ____ 10

I. Équations du système ____ 10

1. Équations ____ 10

2. Linéarisation des équations ____ 11

II. Représentation d'état du système ____ 12

III. Régulateur d'état ____ 12

1. Situation générale ____ 13

2. Présentation du régulateur LQR ____ 13

IV. Recherche des paramètres du modèle ____ 13

V. Simulation sur Matlab ____ 14

1. Principe de simulation ____ 14

2. Résultats des essais ____ 17

B. Application Android ____ 20

Problèmes rencontrés ____ 22

Perspectives d'amélioration et Conclusion ____ 24

Webographie ____ 25

Liste des figures

Figure 1 : Boitier et séparateur en polystyrène _____ 6
Figure 2 : Deux moteurs DC avec boitiers réducteurs et roues _____ 7
Figure 3 : Module Bluetooth, circuit sur la plaque à essai et la première carte Arduino (Nos condoléances !) _____ 8
Figure 4 : Représentation 2D du circuit avec les composants électriques à l'aide de **Fritzing**____9
Figure 5 : Représentation 2D du pendule inversé_____10
Figure 6 : Vue externe du modèle Simulink_____15
Figure 7 : Vue interne du modèle Simulink _____15
Figure 8a: Réponse en position _____ 18
Figure 8b: Réponse en angle _____ 18
Figure 9a : Réponse en position_____18
Figure 9b : Réponse en angle _____ 18
Figure 10 : Simulation du système avec perturbation_____19
Figure 11 : Le bloc de paramètres xddot_____19
Figure 12a: _____20
Figure 12b: _____20
Figure 13 : Module Bluetooth HC-05_____20
Figure 14 : AmenDuino_____21

Introduction

Le module des microprocesseurs étudié en 2ème trimestre est parmi les modules qui suscitent le plus d'intérêt chez les étudiants puisqu'il fait la base des technologies modernes qui garnissent notre vie d'aujourd'hui.

Pour sentir le grand intérêt du module et mettre l'accent sur l'importance du domaine, l'Ecole Polytechnique a planifié tout un module nommé « Application à base de microprocesseur » qui est une suite pratique du module du 2ème trimestre.

Notre équipe éprouvait déjà un grand intérêt pour ce domaine et nous étions déterminé de nous lancer dans un projet assez intéressant qui pourra mettre en pratique nos connaissances en électronique, en automatique et celles qu'on venait déjà d'avoir à partir du module sur les micro-presseurs.

Notre enthousiasme et notre détermination nous a poussés à construire notre propre projet en lui consacrant un petit budget. L'idée était de concevoir un tricopter où on mettra en pratique nos connaissances dans le domaine de l'automatique tout en le réalisant à base de microcontrôleur. Vu qu'on était encore novice pour une telle application et un tel domaine et comme la durée du module était assez courte, on a opté pour un projet plus simple mais qui nous permettra de bien nous lancer dans l'affaire. C'est pourquoi nous avons choisi de faire un robot à deux roues, une application similaire à l'application typique « le pendule inversé ».

Dans ce rapport, nous allons présenter les côtés hardware et software du projet ainsi que les problèmes rencontrés durant sa réalisation.

Côté matériel

A. Composants utilisés

Notre robot est constitué de différents composants :

Conteneur

- Boitier en carton 155x95 (mm)
- Velcros
- Matériaux de récupération en polystyrène

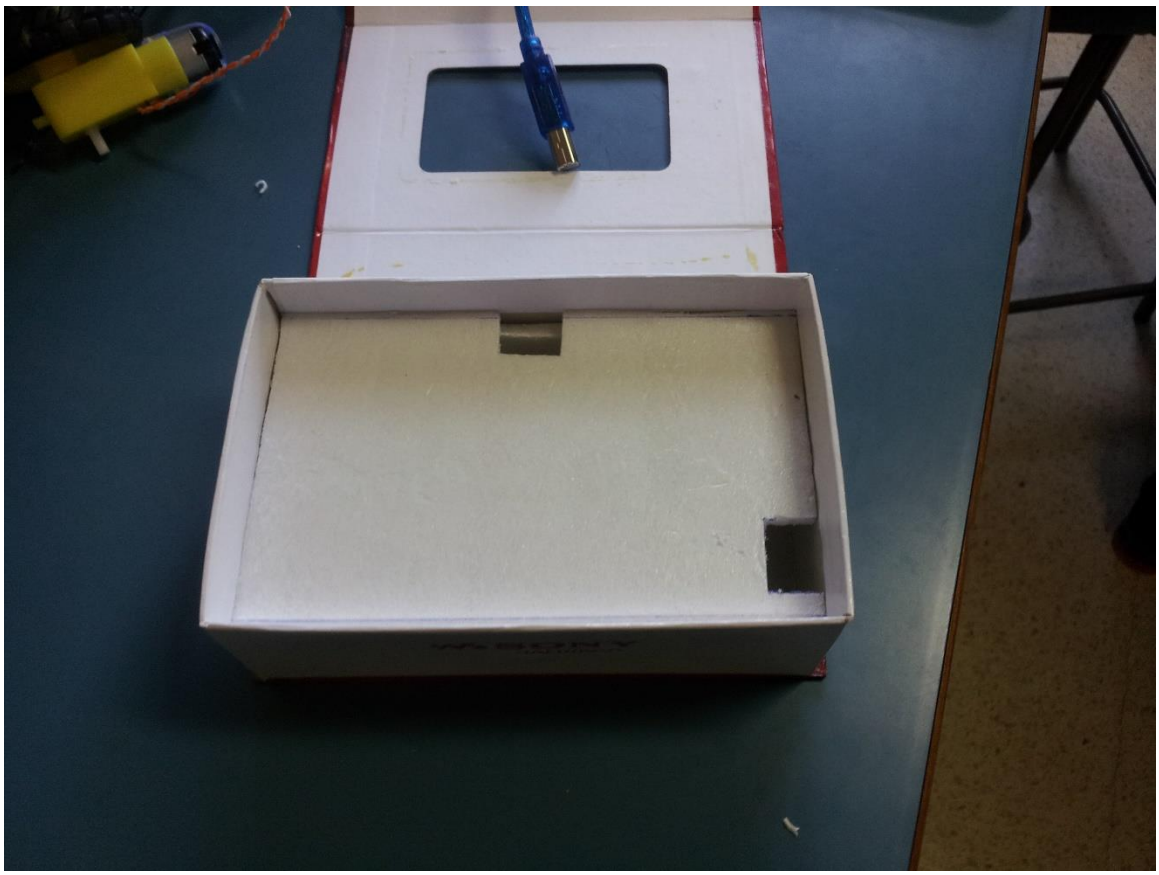


Figure 1 : Boitier et séparateur en polystyrène

Partie opérative

- Deux moteurs à courant continus avec boitiers réducteurs
- 2 roues de 60mm de diamètre

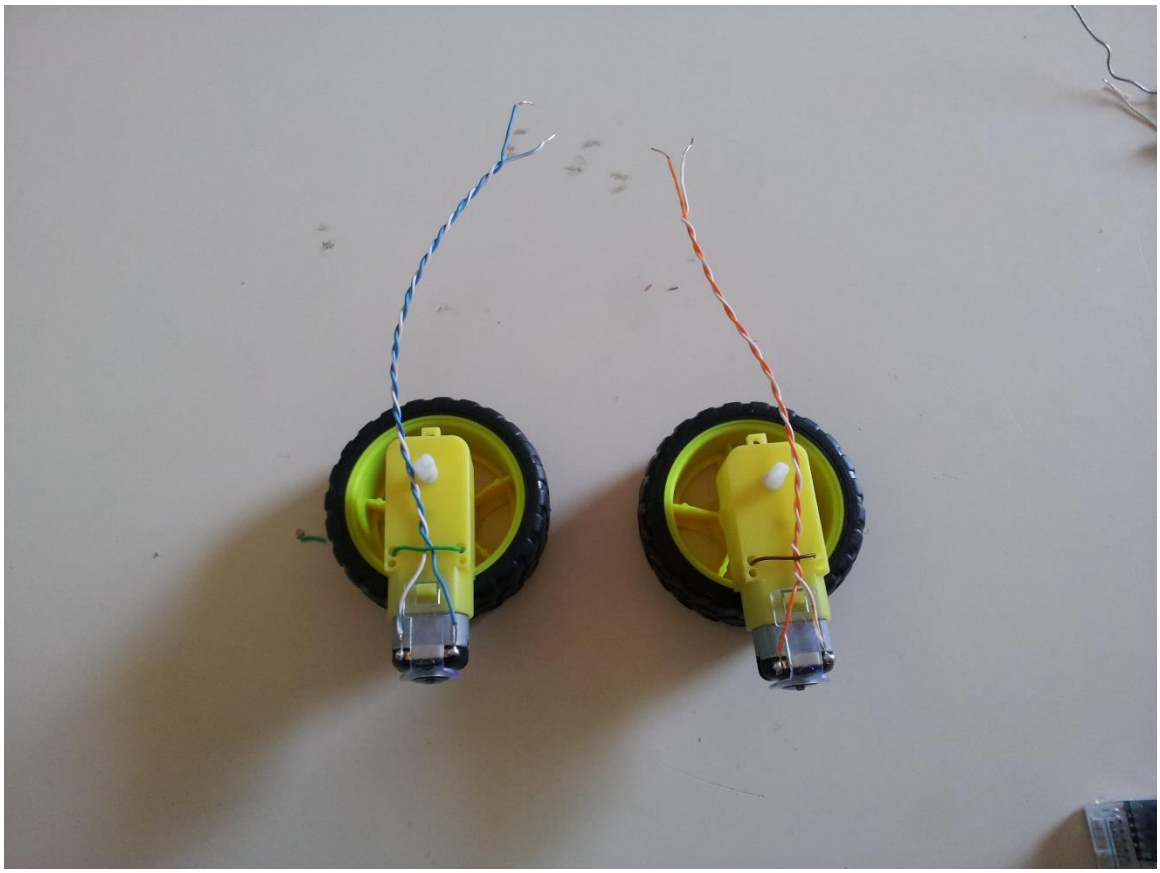


Figure 2 : Deux moteurs DC avec boitiers réducteurs et roues

Partie de contrôle et modules

- Un Samsung Galaxy S2 (Exploitation du gyroscope)
- Une Carte Arduino Leonardo
- Un module Bluetooth HC-05

Partie électrique et connexions

- Des fils électriques
- Des connecteurs mâle-mâle et femelle-femelle
- Une plaque à essai
- Une pile 9V avec clip
- Un jack d'alimentation 3.1mm
- Un régulateur 9V
- 3 condensateurs 100nF
- 3 résistances 1kOhm
- 1 L293D
- 2 interrupteurs

- Un transformateur abaisseur de tension 9V (externe)

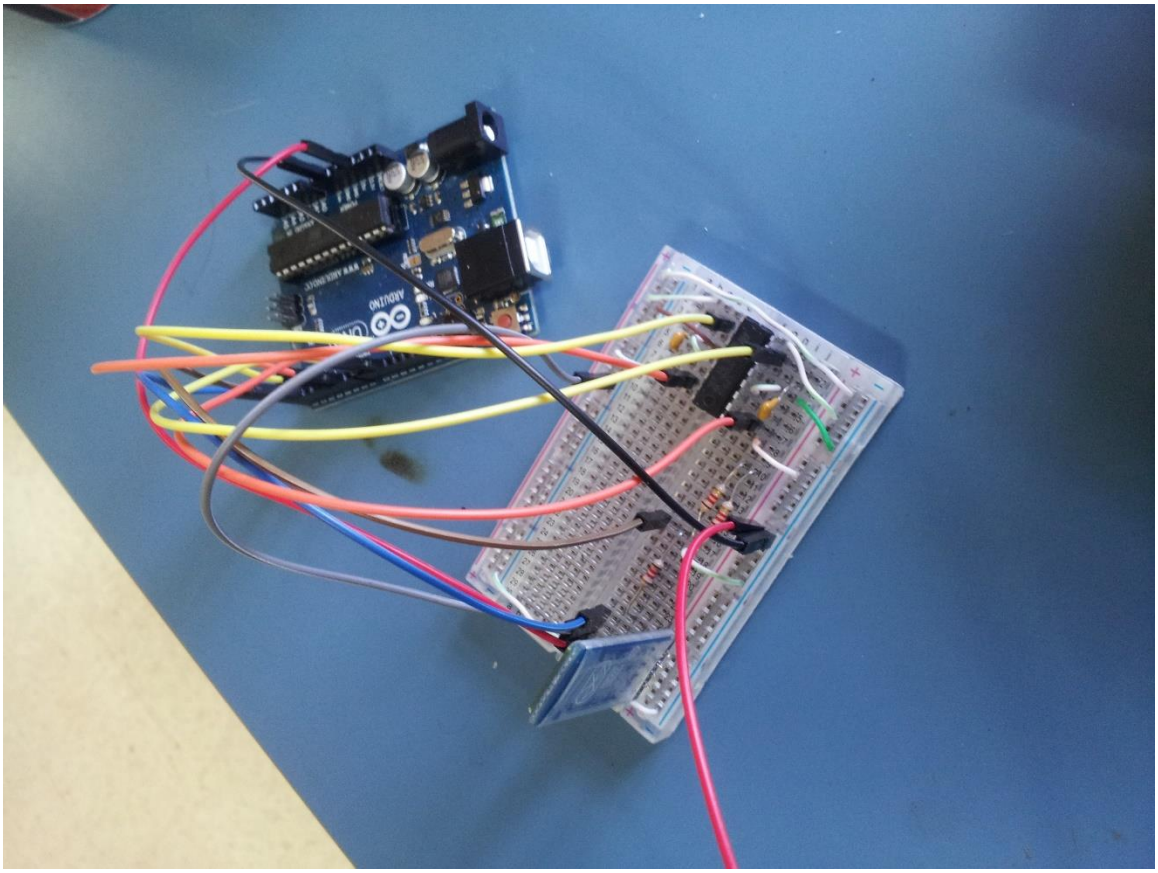


Figure 3 : Module Bluetooth, circuit sur la plaque à essai et la première carte Arduino (Nos condoléances !)

B. Représentation du circuit

Nous avons préféré d'assembler notre circuit à l'aide de fils avec connecteurs mâle-mâle et femelle-femelle pour plus de flexibilité dans le montage et démontage des composants. Aussi a-t-on opté pour un ***Breadboard*** (Plaque à essai) de type ***half+*** pour une meilleure ergonomie et pour mieux s'adapter à la taille du boitier utilisé.

Pour une meilleure représentation, on a préféré l'utilisation du logiciel ***Fritzing*** pour donner une représentation 2D beaucoup plus attractive qu'un simple schéma électrique.

D'après Wikipedia.org, ***Fritzing*** est un logiciel libre de conception de circuit imprimé permettant de concevoir de façon entièrement graphique le circuit et d'en imprimer le typon.

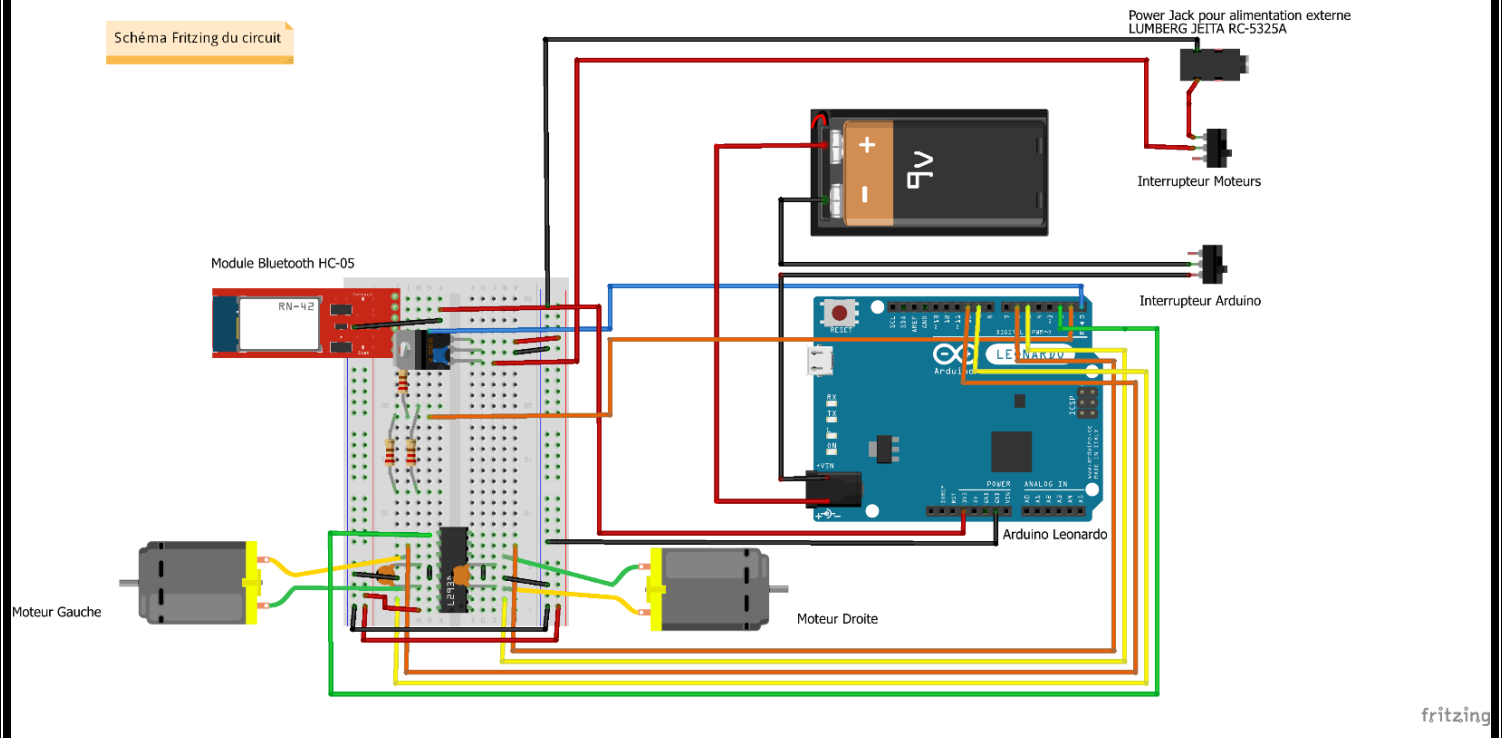


*Figure 4 : Représentation 2D du circuit avec les composants électriques à l'aide de **Fritzing***

Côté logiciel

A. Matlab

I. Équations du système

1. Équations

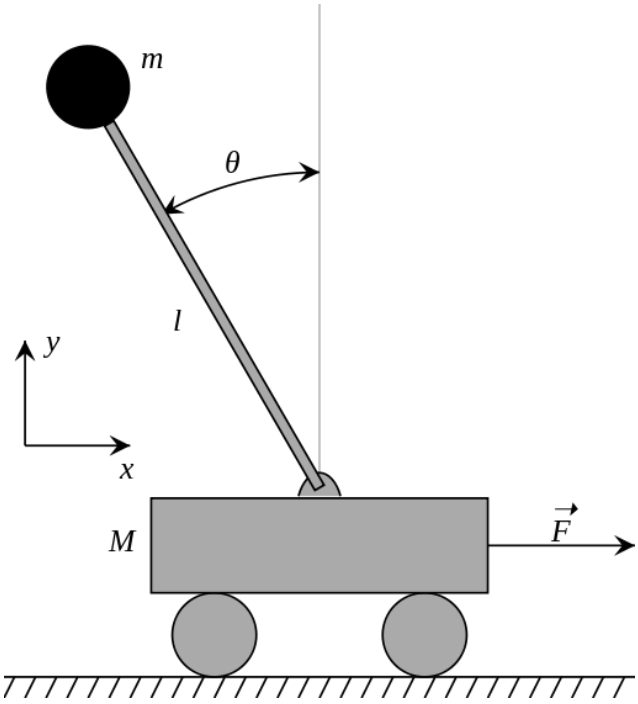


Figure 5 : Représentation 2D du pendule inversé

Les théories de la mécanique générale nous amènent aux équations suivantes:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = F$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0$$

où **L** est l'opérateur **lagrangien**.

Le développement de ces équations donne les résultats suivants:

$$(m_p + m_c)\ddot{x} + m_p\ddot{\theta}l\cos\theta - m_p\dot{\theta}^2 l\sin\theta = F \qquad (1)$$

$$m_p l\cos\theta\,\ddot{x} - m_p l\sin\theta\,\dot{\theta} + m_p\ddot{\theta}l^2 - m_p gl\sin\theta = 0 \qquad (2)$$

m_p : Masse du pendule

m_p : Masse du chariot

x : Position de la diapositive par rapport à une position de référence

θ : Angle par rapport à la verticale

g : Accélération de la pesanteur

F : Force de stabilisation

Il est clair que ces équations sont non linéaires par rapport à nos paramètres $\theta\ et\ x$.

2. Linéarisation des équations

Les équations (1) et (2) illustrent la réalité. Donc la résolution de ces équations donne la réponse réelle de la diapositive. Mais dans notre cas on va faire des approximations afin de manipuler des modèles linéaires dont l'implémentation est facile dans le microcontrôleur de la carte Arduino.

Une approximation de premier ordre est justifiée par le fait que l'angle θ reste toujours petit devant 1 radian et que la position x doit être quasiment invariante.

L'approximation du premier ordre consiste à faire les hypothèses sur les trois instructions suivantes:

- $\cos\theta \approx 1$
- $\sin\theta \approx \theta$
- $\dot{\theta}^2 \approx 0$

Après la prise en considération de ces approximations, les équations linéarisés sont:

$$\begin{cases} (m_p + m_c)\ddot{x} + m_p\ddot{\theta}l = F & (3) \\ \ddot{x} + \ddot{\theta}l - g\theta = 0 & (4) \end{cases}$$

II. Représentation d'état du système

On fait tout d'abord un réarrangement des termes du système (3) et (4).

$$\begin{cases} \ddot{x} = -\dfrac{m_p g}{m_c}\dot{x} + \dfrac{1}{m_c}F & (5) \\ \ddot{\theta} = \dfrac{(m_c + m_p)}{m_c l}g\dot{x} - \dfrac{1}{m_c l}F & (6) \end{cases}$$

Soit X le vecteur d'état du système. $X=\begin{pmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{pmatrix}$. On peut reformuler le système (3) et (4) sous une représentation matricielle.

$$\dot{X} = \begin{pmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\dfrac{m_p g}{m_c} & 0 & 0 \\ 0 & \dfrac{(m_c + m_p).g}{m_c.l} & 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \dfrac{1}{m_c} \\ -\dfrac{1}{m_c l} \end{pmatrix} F, \qquad (7)$$

On va utiliser une commande linéaire par retour d'état. C'est-à-dire on va détecter à chaque fois l'état du système X est on va donner une commande proportionnelle a X, soit –KX, en chaine de retour.

La commande se fait au niveau du force de stabilisation F=-KX avec $K=\begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{pmatrix}$

Le problème central est comment choisir les quatre constante k_1, k_2, k_3 et k_4 ?

III. Régulateur d'état

Une recherche bibliographique approfondie dans les approches pratiques de l'automatique montre que souvent on utilise le régulateur LQR (*Linear Quadratic Regulation*) dans les applications basées sur le principe du pendule inversé.

1. Situation générale

On cherche à asservir un système MIMO (*Multiple Input Multiple Output*).
On dispose pour cela :

- ✓ D'un vecteur d'état X
- ✓ D'un ensemble de variables de contrôle, u (qui est F dans notre cas)
- ✓ D'une équation qui régit le système

$$\dot{X} = AX + Bu$$

On veut :

- ✓ Amener X a un état X_{but} à moindre coût en commandant les effecteurs (u).
- ✓ Bien réagir aux perturbations.
- ✓ Contrôler u linéairement grâce à X de manière optimale.

2. Présentation du régulateur LQR

Pour contrôler F linéairement grâce à X, on espère pouvoir implémenter une formule de la forme:

$$u_k = -Kx_k$$

Ce K est appelé *Linear Quadratic Regulator*, ou LQR. Le calcul des coefficients de K est basé sur des approches de l'algèbre linéaire pour assurer la stabilité, la rapidité et la contrôlabilité du système.

IV. Recherche des paramètres du modèle

Pour trouver K en temps fini on applique la formule suivante

$$K = (R + B^T PB)^{-1} B^T PA$$

P est obtenu par la résolution de l'équation de Riccati:

$$P = Q + A^T(P - PB(R + B^T PB)^{-1}B^T P)A$$

A, B sont déjà définis dans l'équation (7). Il nous reste donc que la détermination de Q et R.

Le filtre LQR permet de calculer le contrôle linéaire optimal au sens défini par l'ingénieur grâce à une fonction de coût J, que le filtre LQR minimisera. Q et R sont alors les matrices de pondération de X et u de la fonction de coût :

$$J = \sum_{k=0}^{\infty} (x_k^T Q x_k + u_k^T R u_k)$$

Le choix de ces matrices est laissé à la discrétion des développeurs du système de contrôle. Cependant, Q doit être semi-définie positive et R définie positive .Elles expriment les préférences du concepteur en termes de contrôle sur A et B. Par exemple, si l'on veut que l'observable x_i soit minimisé en priorité, on va y affecter un poids Q_{ii} fort. Ainsi, pour la fonction de coût, ne pas minimiser x_i rapidement induira un coût assez grand. De la même façon, pour les termes non diagonaux, un poids fort en R_{ij}, i différent de j, a pour effet d´encourager l'utilisation des deux effecteurs u_j et u_i en même temps.

On ne va pas se contenter de la résolution des équations algébrique et la minimisation du critère J mais Matlab va tout faire grâce au commande lqr (A,B,Q,R). Notre tâche est de choisir les matrice Q et R. Lorsque on creuse davantage dans les théories derrière le régulateur LQR, on remarque que le choix de $Q_{ii}= \frac{1}{|x_{i\,Max}|^2}$ et Q_{ij} =0 facilite les calculs.

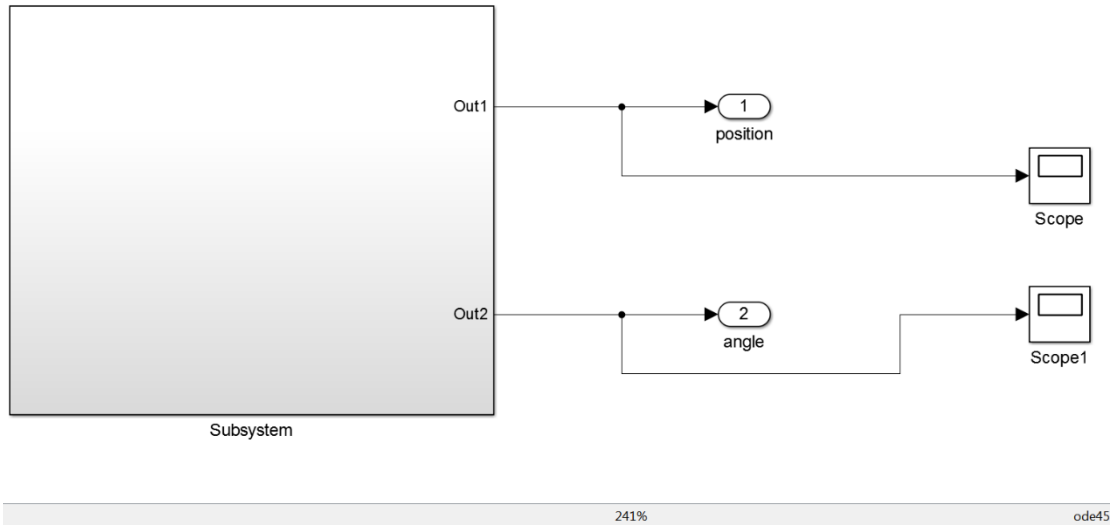
$R=\frac{1}{|V|^2}$ où v désigne l'énergie d'alimentation.

Ces choix vont donner des réponses qui convergent en un temps fini mais qui peuvent présenter des fluctuations avec des dépassements critiques. Notre tâche est donc de simuler la réponse du système puis de changer les choix de Q et R en fonction des réponses obtenues pour avoir la réponse la plus rapide et la précise avec le minimum d'énergie.

V. Simulation sur Matlab

1. Principe de simulation

On va exploiter l'outil de simulation *Simulink* de Matlab pour visualiser la réponse du système (angle et position) et définir les bons coefficients de Q et R.



Le block *Subsystem* illustre le système asservi en boucle fermée. *Scope* et *Scope1* sont les réponses du système en position x(t) et angle θ(t).

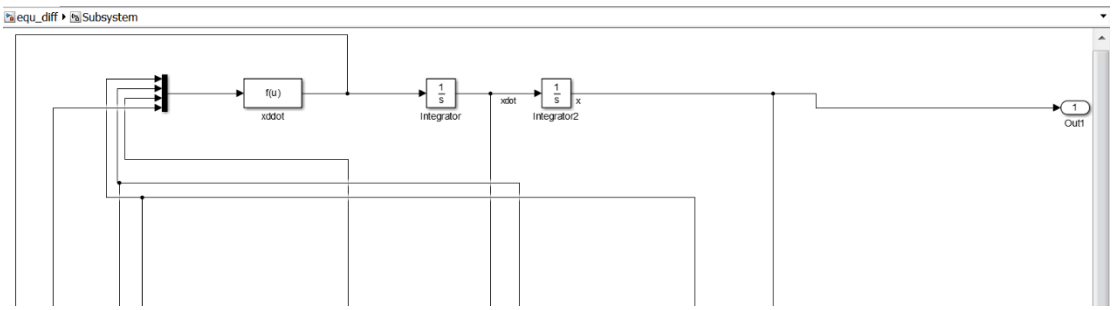


Figure 6 : Vue externe du modèle Simulink

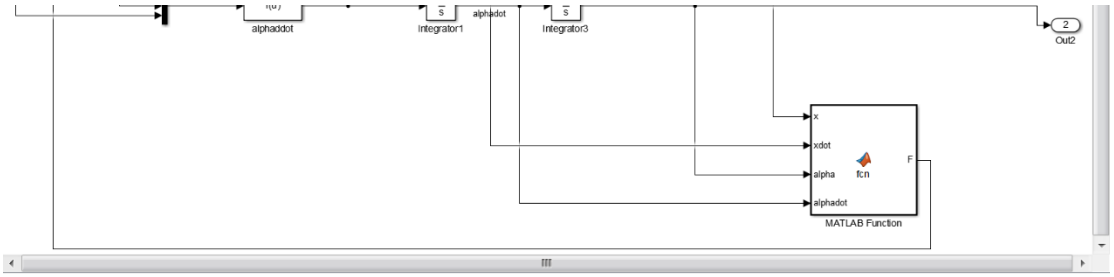


Figure 7 : Vue interne du modèle Simulink

La figure 7 illustre le mécanisme de fonctionnement du *Subsystem*.

En effet à chaque fois, on résout le système d'équations linéaires (5) et (6) pour la valeur de F actuelle. La résolution se fait par l'outil Matlab **ode45** (*ordinary differential equations*) d'une façon systématique comme il est mentionné dans l'environnement de Simulink en bas à droite. Les résultats sont envoyés vers le block *Matlab function*. Cette fonction Matlab calcule la valeur du nouvel F qui sera injectée dans les équations différentielles du système.
On peut résumer tout ça par la boucle suivant.

- ✓ Résolution des équations différentielles du système
- ✓ Calcul de la force nécessaire à la stabilisation F
- ✓ Injection de F dans les équations différentielle

La fonction Matlab va tout simplement multiplier le vecteur d'état reçu de la résolution des équations par le vecteur K calculé.

```
function F = fcn(x,xdot,alpha,alphadot)
%#codegen

F = -[ -18.7083  -93.3809  -28.8417  -
14.2414]*[x,alpha,xdot,alphadot]';
return
end
```

Les coefficients de K sont calculés dans une fonction Matlab développée à partir de la fonction Commande. Cette fonction va calculer les paramètres Ki du système. On doit donc configurer les caractéristiques dynamiques et physiques du système, les coefficients Rij et Qij, les vecteurs et la matrice de la représentation d'état. Le calcul des coefficients de K sera tout simplement le résultat du commande K=lqr(A,B,Q,r);

```
function[K]=commande()
mp=0.6; %masse du pendule
mc=0.05; %masse du chariot
lp=0.2; % longueur du pendule
g=9.81; %accélération de pesanteur

%%%%%%%%%%termes des matrice Q et r%%%%%%%
q11=7;       %x
q22=2.5;     %theta
```

```
q33=10;      %x_dot
q44=2;       %theta_dot
r=0.02;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A=[0 0 1 0 ; 0 0 0 1; 0 -mp*g/mc 0 0; 0
(mc+mp)*g/(mc*lp) 0 0];

B=[0 0 1/mc -1/(mc*lp)]' ;
q=[q11 q22 q33 q44];
Q=diag(q);

K=lqr(A,B,Q,r);
end
```

Maintenant pour déterminer les bons coefficients de Q et R, on va changer les Rij et Qij et calculer le vecteur K par la fonction *Commande()*. Puis copier les coefficients Ki dans l'outil de simulation Simulink et visualiser les réponses en angle et en position.

2. Résultats des essais

Cette tâche d'estimation des paramètres optimaux de Q et R nécessite un esprit d'analyse des résultats et une expertise dans le rôle de chaque coefficient de R et Q.
On a essayé plusieurs fois avec différentes valeurs de Q_{ii} et R et voilà les résultats de quelque essai.

On va commencer par visualiser la réponse du système pour un angle initial 0.5 radians.

Réponse du système pour K= [-1.4142 -21.4395 -2.5185 -3.7446]

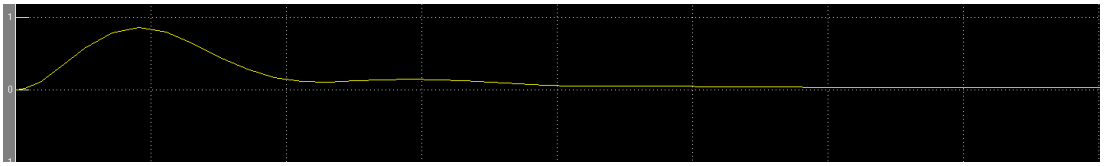


Figure 5a: Réponse en position

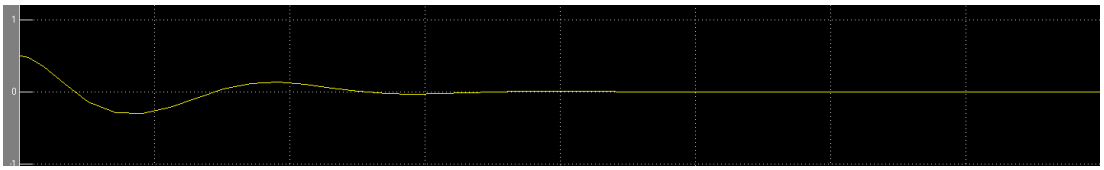


Figure 5b: Réponse en angle

Commentaire:

On remarque que ces résultats sont stables ou convergents en temps fini mais ils présentent des fluctuations. On a changé les valeurs de Q et R et on a visualisé les courbes de réponses pour aboutir finalement au résultat optimal suivant.

Réponse du système pour K= [-18.7083 -93.3809 -28.8417 -14.2414]

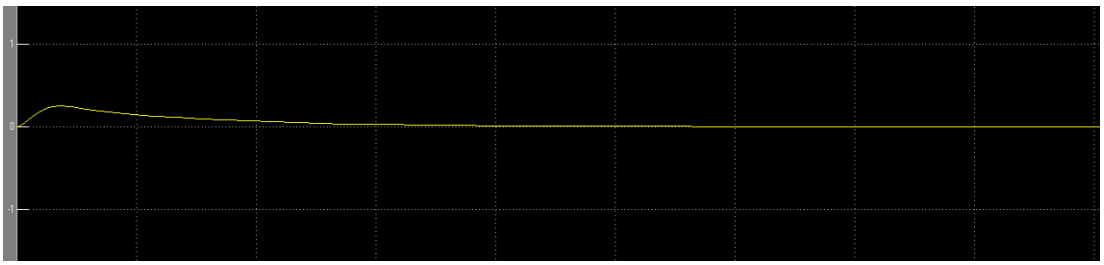


Figure 9a : Réponse en position

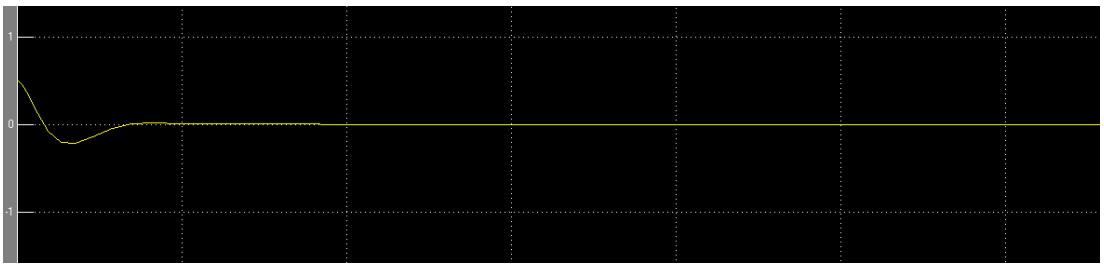


Figure 6b : Réponse en angle

Comme ça on a bien construit notre régulateur LQR,
K=[-18.7083 -93.3809 -28.8417 -14.2414]

Pour mieux s'assurer de sa stabilité vis-à-vis aux perturbations, on a simulé des perturbations de type impulsion de période T et de largeur 10%, qui s'ajoutent à chaque période à la valeur de F pour perturber le système.

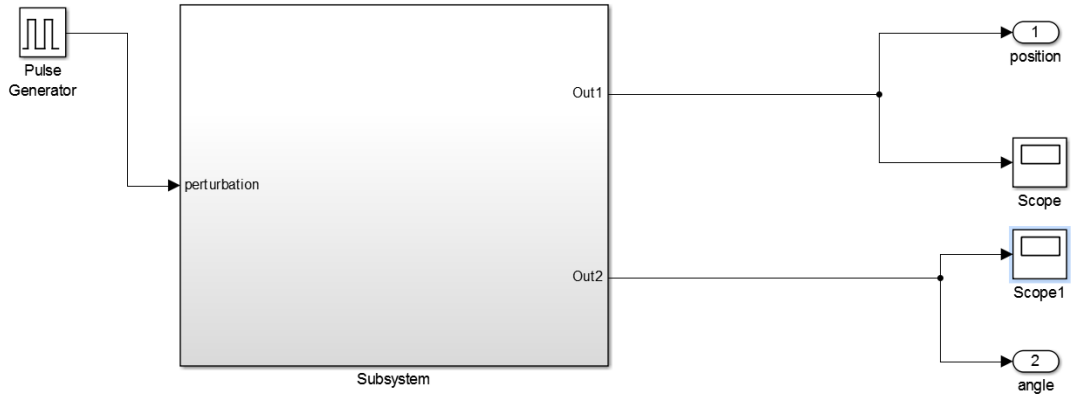


Figure 10: Simulation du système avec perturbation

On ajoute un *pulse generator* qui va générer un signal carré qu'on va ajouter comme force de perturbation. Il est mentionné dans le figure 11 ci-dessous.

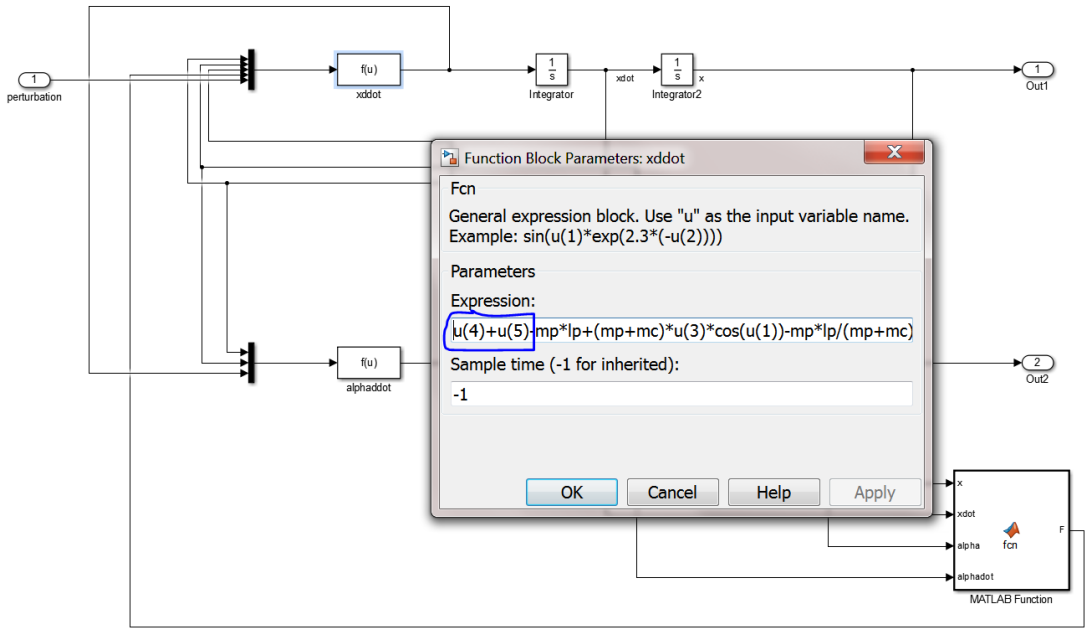


Figure 11: Le bloc de paramètres xddot

Les résultats étaient extraordinaires et comme ça on a validé la partie théorique du projet. Ces résultats sont illustrés dans les figures 12.a et 12.b

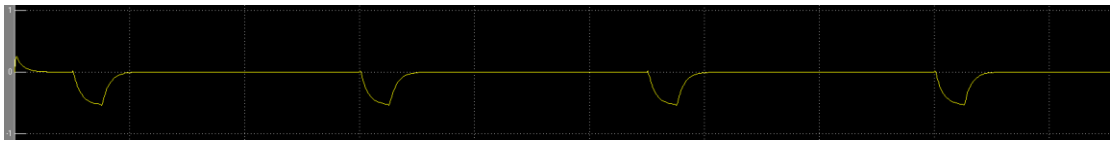


Figure 12a

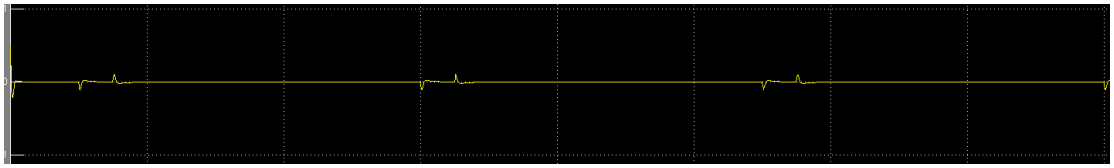


Figure 12b

Maintenant tout est parfait sur le plan théorique. On a terminé avec les équations de la physique de l'algèbre et l'automatique. On a aussi simulé le fonctionnement du robot il nous reste donc qu'implémenter tout ce travail.

B. Application Android

La commande nécessaire au maintien à l'équilibre du robot a principalement besoin de deux paramètres : la vitesse angulaire pour mesurer la vitesse de chute ainsi que l'angle par rapport à la verticale. Comment peut-on les obtenir?

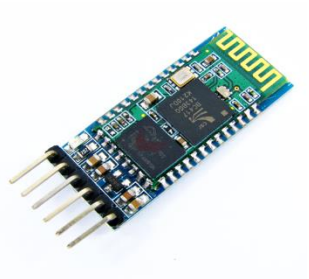


Figure 13 : Module Bluetooth HC-05

Une première idée était de se procurer d'un module gyroscope compatible avec notre carte Arduino, mais c'était relativement cher. Et c'est là qu'est née l'idée d'exploiter mon smartphone équipé, entre autres, de ce type de capteur, surtout qu'on possède déjà un module Bluetooth. Ce dernier sera parfait pour assurer le transfert des paramètres de positions du smartphone, attaché au robot, jusqu'à l'Arduino, qui sera chargé du calcul de la commande.

On a commencé alors à développer une application Android pour l'envoi des données du capteur par Bluetooth. Mais avant de se lancer à fond, on était convaincu qu'on n'est pas les premiers à y penser (exploiter les nombreux capteurs du smartphone) et on s'est mis alors à fouiller dans le Google Play à la recherche d'une application qui pourra faire l'affaire. Là, je suis tombé sur une bonne candidate: SensoDuino. Après les tests, on s'est rendu compte qu'elle a en fait un handicap: la vitesse de transmission de données est limitée à une valeur de 100 ms. On a jugé que c'était insuffisant, surtout que notre gyroscope pourrait aller jusqu'à 110 Hz, soit dix fois plus de valeurs transmises pour une même durée de temps. Là, on s'est trouvé obligé de faire de l'ingénierie inverse : on a dû extraire de l'apk de l'application, décompiler des différents fichiers, se passer de la signature et puis faire certaines manipulations délicates pour enfin donner naissance à la première AmenDuino, capable d'assurer l'envoi d'une valeur chaque 10 ms (100Hz).

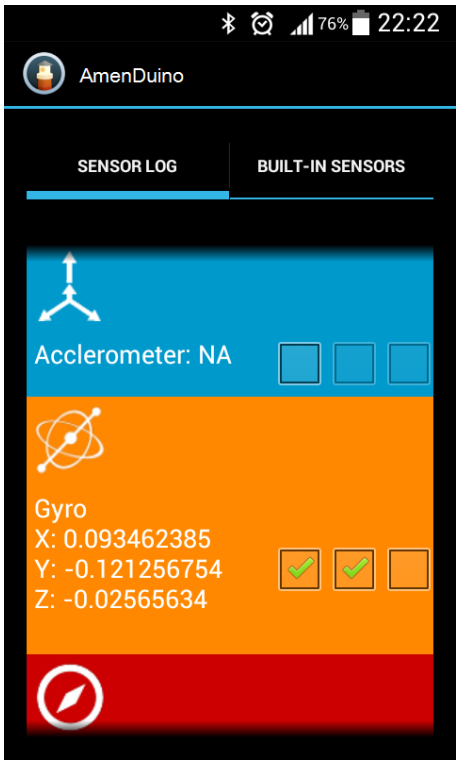


Figure 14 : AmenDuino

Ainsi, les paramètres de positionnement sont bien transmis à l'Arduino dans les bons délais.

Place maintenant à la conception et l'assemblage ainsi que la préparation de la commande adéquate.

Problèmes rencontrés

Durant notre travail sur ce robot, on a rencontré plusieurs obstacles qui nous ont fait perdre beaucoup de temps pour les contourner, mais qui nous ont appris certaines subtilités qu'on évitera systématiquement dans l'avenir.

- **Le problème du voltage maximum supporté par la L293D**

Nous avons testé les moteurs en les commandant, via le driver L293D, avec une tension de 12 volts afin d'avoir la meilleure vitesse de rotation angulaire. Les moteurs répondaient pour pas mal de tentatives d'essais, mais tout à coup, leurs réponses sont devenues bizarres ! Le pire, c'est qu'on a détecté un court-circuit d'origine inconnue dans notre circuit ! Après plusieurs tests pour la détection de l'origine du court-circuit, il s'est avéré que le driver L293D était alimenté par sa tension de travail limite, les 12V, ce qui a causé sa détérioration.

- **Le problème de la tension seuil des moteurs**

La commande des moteurs doit être optimale pour que ces derniers réagissent rapidement aux changements des valeurs mesurées par le gyroscope. Les premières commandes établies ont fait une transformation linéaire de la commande en voltage pour l'appliquer aux moteurs. Ces actionneurs produisaient une sorte de gazouillement sans tourner, qu'à partir un angle assez important.
Le problème était le fait de ne pas tenir compte de la tension seuil des moteurs (aux alentours de 1.6V), la tension à partir de laquelle les moteurs commencent à tourner. On a dû donc prendre en compte ce fait dans le *mapping* de la commande vers le voltage applicable aux moteurs.

- **Le problème de l'Arduino UNO (La première carte Arduino utilisée)**

Au début, on a utilisé une carte Arduino UNO pour la commande du robot. Quelques fois, des jours n'étaient pas nos jours de chance ! Pas mal de fois par exemple, on a fait accidentellement tomber le boitier du robot avec les moteurs bien fixés et il a fallu tout refaire.
Un jour, on a fait des tests pour alimenter la carte avec un transformateur externe. On sentait parfois quelques parties de la carte s'échauffer, mais on n'a constaté pas de problème puisqu'elle marchait très bien. A la fin de la séance, quand tout était prêt pour un vrai test de la commande, l'Arduino n'est plus détectée sur le PC. Même sur les autres ordinateurs c'était le même problème. On a donc documenté dessus pour savoir la cause du problème. Il s'est avéré que le convertisseur série-USB de la carte a été passé de vie à

trépas à cause de la mauvaise qualité du transformateur qui donnait une tension bien plus supérieur que celle indiquée dessus.

- ## Le problème du « Serial » de la Arduino Leonardo

Après avoir changé vers une Arduino Leonardo, on a constaté un problème dans la réponse sérielle de la carte. En effet, on devrait utiliser sa connexion série (les pins TX-RX) pour transmettre et recevoir les données de mesure des capteurs et ça n'a, malheureusement, pas marché ! Même avec un simple programme qui lit et écrit sur le moniteur série, cela a encore échoué.

Après une dizaine de jours sans solutions, on a réalisé que la carte Leonardo était en fait un peu particulière : elle possédait deux objets pour la communication série : ***Serial*** et ***Serial1***, l'un matériel et l'autre virtuel. Il faudrait savoir lequel des deux utiliser pour notre cas et qu'il faudrait aussi faire attendre le ***Serial1*** à l'initialisation de la carte jusqu'à l'établissement d'une nouvelle connexion. Cette dernière condition consistait à ajouter ces lignes de codes de le bloc *setup()* du programme

```
while(!Serial1){ //required for the Leonardo
    ;
}
```

Perspectives d'amélioration et Conclusion

Ce robot est en fait une application basée sur la théorie de la commande. Une application similaire de l'application typique « le pendule inversé » (utilisé dans des laboratoires d'automatique pour les travaux pratiques). Cependant, notre robot pourra être amélioré pour faire des tâches plus intelligentes telles que la simulation d'un serveur dans un restaurant. Il pourrait être aussi un porteur d'objets tout terrain ou servir comme chaise roulante pour monter ou descendre des pentes en sécurité etc…

Pour conclure, le module « Applications à base de microprocesseur » est l'une des matières les plus appréciées de cette année. En effet, elle nous a permis de sortir du quotidien des études pour appliquer les connaissances sur des projets réels. La théorie seule ne peut pas être comprise et bien assimilée sans l'appliques dans des cas réels et sans sentir les difficultés de manipulation et de mise en œuvre des microcontrôleurs.

Le travail sur ce robot est le fruit d'un travail collectif persévérant et sérieux. L'entente, la flexibilité et l'esprit d'initiative partagé étaient en fait les atouts de notre équipe.

Nous avons acquis de nouvelles connaissances à partir de ce projet et nous avons appris davantage sur l'électronique et la programmation grâce aux problèmes rencontrés qu'on a finis par résoudre.

Finalement, notre équipe est déterminée pour continuer dans des projets plus développés et plus intéressants qui nous permettrons d'approfondir notre savoir-faire et acquérir de nouvelles expériences, tout comme a dit Emil Michel Cioran : "Notre caractère est déterminé par l'absence de certaines expériences plus encore que par celles que l'on fait."

Webographie

- http://www.instructables.com/id/A-Simple-and-Very-Easy-Inverted-Pendulum-Balancing/
- http://www.youtube.com/watch?v=Rm-2lXlCWZk (Arduino Tutorial #9: Leonardo vs. Uno)
- http://www.youtube.com/watch?v=ApcEqZ7Twys (Two Wheel Self Balancing Robot)
- http://www.youtube.com/watch?v=nXymP6ttxD4 (Arduino - Control DC Motor via Bluetooth)
- http://arduino.cc/
- http://vimeo.com/2952236 (Inverted Pendulum)
- http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SimulinkModeling