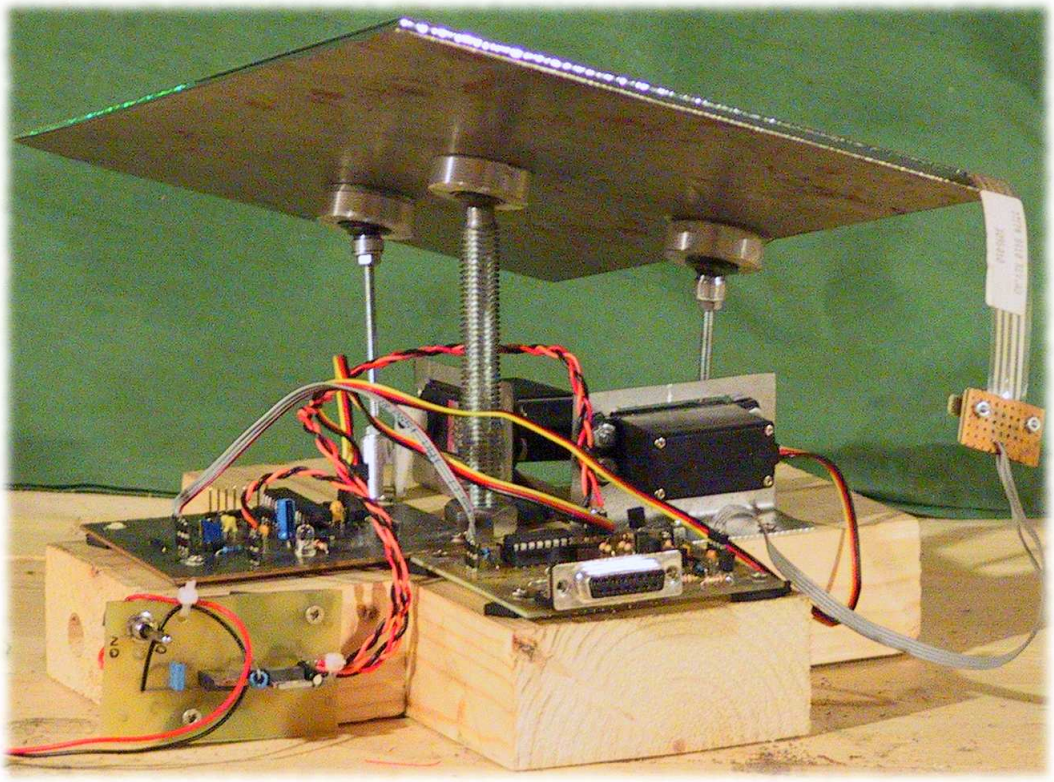
Régulation de la position d'une bille sur un plateau inclinable

Rapport de projet

L3 EEAR à distance
Université Henri Poincaré – CNED

Nicolas Bernard
n.bernard@lafraze.net

10 juin 2009



Empreinte de la version (`git`)
`6e30bccc5686c7407f0f29dc6fa5c07fda0cce70`
Document compilé (LaTeX) le 7 juin 2009.

Ce document est formaté avec LaTeX. Sauf exception, les figures ont été réalisées avec `xfig` et les graphiques à l'aide de `gnuplot` ; les schémas électroniques ont été réalisés avec `gschem` de la suite `gEDA` et les circuits imprimés avec le programme `pcb`.

Table des matières

Introduction

Le but de ce projet était d'étudier la régulation de la position d'une bille sur un plateau mobile, celui-ci devant être préalablement construit. Ce plateau est mobile en ceci qu'il peut s'incliner selon ses deux axes, l'inclinaison étant contrôlable par des moteurs. Le plateau est constitué (ou, comme ici, recouvert) d'une dalle tactile qui permet de connaître la position de la bille.

Dans la fabrication de ce projet, il y a trois parties distinctes :
- la partie mécanique ;
- la partie électronique ;
- la partie informatique, c'est-à-dire la programmation de la puce (un microcontrôleur dsPIC) qui contrôle le système.

Toutefois, avant de parler de la fabrication elle-même, nous allons commencer par étudier ce système de manière théorique. C'est l'objet du premier chapitre.

Chapitre 1

Étude du système

1.1 Modélisation

Le problème est le même selon les deux axes d'inclinaison (x et y) du plateau. Nous nous contenterons donc de le modéliser sur les x, le raisonnement est le même pour les y.

Sur un plan incliné d'un angle α par rapport à l'horizontale, une bille se met à rouler et son accélération [1] est constante et égale à

$$a_x(t) = \ddot{x}(t) = \frac{5}{7} g \sin \alpha \tag{1.1}$$

Il est facile d'intégrer cette équation et de remonter à la vitesse de la bille, puis à sa position :

$$\dot{x}(t) = \frac{5}{7} g \sin \alpha \; t \tag{1.2}$$

$$x(t) = \frac{1}{2} \frac{5}{7} g \sin \alpha \; t^2 \tag{1.3}$$

Ce qui rend les choses plus complexes ici, c'est que l'inclinaison n'est pas constante : l'angle α varie au cours du temps. L'accélération n'est donc pas constante et on a en réalité :

$$\ddot{x}(t) = \frac{5}{7} g \sin(\alpha(t)) \tag{1.4}$$

Il est beaucoup plus difficile d'intégrer cette équation étant donné que la fonction $\alpha(t)$ n'est pas connue.

Pour simplifier le problème, on va se placer dans le cas discret (c'est de toute facon celui qui va nous intéresser, puisque le microcontrôleur qui va commander notre système fonctionne avec un temps discret). On peut alors supposer que sur un intervalle de temps petit, l'angle est constant. On a alors

$$\alpha(t) = \alpha_t \tag{1.5}$$

1. Celle de son centre de gravité pour être précis

La vitesse devient alors une suite :

$$\begin{aligned} \dot{x}(0) &= \dot{x}_0 && (1.6)\\ \dot{x}(1) &= \frac{5}{7} g \sin\alpha_0 (t_1 - t_0) + \dot{x}_0 && (1.7)\\ &\vdots \\ \dot{x}(i) &= \frac{5}{7} g \sin(\alpha_{i-1}) \underbrace{(t_i - t_{i-1})}_{\Delta_i} + \dot{x}(i-1) && (1.8) \end{aligned}$$

Si l'on suppose que tous les Δ_i sont égaux à Δ, on peut écrire le terme général de la suite ainsi :

$$(\dot{x}(n))_{n>0} = \frac{5}{7} g \Delta \sum_{i=1}^{n} \sin\alpha_{n-1} + \dot{x}_0 \quad (1.9)$$

De même pour la suite $(x(n))$:

$$\begin{aligned} x(0) &= x_0 && (1.10)\\ x(1) &\approx x(0) + \frac{\dot{x}(1) + \dot{x}(0)}{2} (t_1 - t_0) && (1.11)\\ &\vdots \\ x(i) &\approx x(i-1) + \frac{\dot{x}(i) + \dot{x}(i-1)}{2} \Delta_i && (1.12) \end{aligned}$$

En supposant là aussi que tous les Δ_i valent Δ, comme

$$\frac{\dot{x}(i) + \dot{x}(i-1)}{2} = \dot{x}(i-1) + \frac{1}{2}\frac{5}{7} g \Delta \sin\alpha_{n-1} \quad (1.13)$$

le terme général peut s'écrire :

$$(x(n))_{n>0} \approx x(0) + \Delta \sum_{i=0}^{n-1} \dot{x}(i) + \frac{1}{2}\frac{5}{7} g \Delta^2 \sum_{i=0}^{n-1} \sin\alpha_i \quad (1.14)$$

Si l'on pose comme vecteur d'état $X = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$ et en posant $\forall i, U_i = [\sin\alpha_i]$, notre système se met donc sous la forme :

$$\begin{aligned} X_n &= \underbrace{\begin{bmatrix} 1 & \Delta \\ 0 & 1 \end{bmatrix}}_{A} X_{n-1} + \underbrace{\frac{5}{7} g \Delta \begin{bmatrix} \frac{\Delta}{2} \\ 1 \end{bmatrix}}_{B} U_{n-1} && (1.15)\\ Y_n &= \underbrace{[1\ 0]}_{C} X_n && (1.16) \end{aligned}$$

1.2 Étude du système

1.2.1 Système initial

Stabilité

Le déterminant de $(pI - A)$ vaut $(p-1)^2 = p^2 - 2p + 1$: la condition nécessaire du critère de Routh n'est pas vérifiée, ce qui nous confirme l'instabilité du système.

Commandabilité

Déterminons la matrice de commandabilité $\mathcal{C}$ et son rang.

$$\begin{aligned} \mathcal{C} &= \left[\begin{array}{cc} B & AB \end{array}\right] && (1.17)\\ &= \frac{5}{7}g\Delta \left[\begin{array}{cc} \frac{1}{2}\Delta & \frac{3}{2}\Delta \\ 1 & 1 \end{array}\right] && (1.18)\\ \det \mathcal{C} &\neq 0 \quad (\Delta \neq 0) && (1.19) \end{aligned}$$

Le rang de cette matrice est 2 (si Δ n'est pas nul, ce qui est nécessairement le cas), le système est donc commandable.

Observabilité

Déterminons la matrice d'observabilité $\mathcal{O}$ et son rang.

$$\begin{aligned} \mathcal{O} &= \left[\begin{array}{c} C \\ CA \end{array}\right] && (1.20)\\ &= \left[\begin{array}{cc} 1 & 0 \\ 1 & \Delta \end{array}\right] && (1.21)\\ \det \mathcal{O} &\neq 0 \quad (\Delta \neq 0) && (1.22) \end{aligned}$$

Le rang de cette matrice est 2 (si Δ n'est pas nul, ce qui est nécessairement le cas), le système est donc observable.

1.2.2 Commande par retour d'état

Si l'on utilise une commande par retour d'état, on a alors

$$U_i = -K\left(X_i - X^\star\right) \tag{1.23}$$

$X^\star$ étant la consigne et $K = \left[\begin{array}{cc} k_1 & k_2 \end{array}\right]$.

Notre système devient donc :

$$\begin{aligned} X_n &= \left[\begin{array}{cc} 1 & \Delta \\ 0 & 1 \end{array}\right] X_{n-1} - \frac{5}{7}g\Delta \left[\begin{array}{c} \frac{\Delta}{2} \\ 1 \end{array}\right] K\left(X_{n-1} - X^\star\right) && (1.24)\\ Y_n &= [1\ 0]\ X_n && (1.25) \end{aligned}$$

soit :

$$\begin{aligned} X_n &= \underbrace{(A - BK)}_{A'} X_{n-1} + \underbrace{BK}_{B'} X^\star && (1.26)\\ Y_n &= [1\ 0]\ X_n && (1.27) \end{aligned}$$

Conditions de stabilité

Étudions A' pour savoir à quelles conditions ce système sera stable :

$$\begin{aligned}\det(pI - A') &\approx p^2 + \left(g\Delta\left(\frac{5}{14}\Delta k_1 + \frac{5}{7}k_2\right) - 2\right)p \\ &\quad + g\Delta\left(\frac{5}{14}\Delta k_1 - \frac{5}{7}k_2\right) + 1 \end{aligned} \tag{1.28}$$

Comme il s'agit d'un système du second ordre, le critère de Routh nous indique qu'il faut :

$$\begin{cases} g\Delta\left(\frac{5}{14}\Delta k_1 + \frac{5}{7}k_2\right) - 2 > 0 \\ g\Delta\left(\frac{5}{14}\Delta k_1 - \frac{5}{7}k_2\right) + 1 > 0 \end{cases} \tag{1.29}$$

$$\begin{cases} g\Delta\left(\frac{5}{14}\Delta k_1 + \frac{5}{7}k_2\right) > 2 \\ g\Delta\left(\frac{5}{14}\Delta k_1 - \frac{5}{7}k_2\right) > -1 \end{cases} \tag{1.30}$$

comme $g > 0$ et $\Delta > 0$,

$$\begin{cases} \frac{5}{14}\Delta k_1 + \frac{5}{7}k_2 > 2 \\ \frac{5}{14}\Delta k_1 - \frac{5}{7}k_2 > -1 \end{cases} \tag{1.31}$$

En additionnant, on a :

$$2\frac{5}{14}\Delta k_1 > 1 \tag{1.32}$$

$$k_1 > \frac{7}{5\Delta} \tag{1.33}$$

En soustrayant cette fois, on obtient :

$$2\frac{5}{7}k_2 > 3 \tag{1.34}$$

$$k_2 > \frac{21}{10} \tag{1.35}$$

On a donc les deux conditions suivantes :

$$\begin{cases} k_1 > \frac{7}{5\Delta} \\ k_2 > \frac{21}{10} \end{cases} \tag{1.36}$$

Précision

On remarque qu'il n'est pas possible de satisfaire l'équation

$$A - BK = 0 \tag{1.37}$$

avec les conditions précédentes sur K.

Cela implique que cette régulation n'est pas précise (gain statique en boucle fermée différent de 1).

Nous allons donc « augmenter » le système pour résoudre ce problème.

1.2.3 Système augmenté

Vecteur d'état augmenté

On utilise le vecteur d'état augmenté $X = \begin{bmatrix} x \\ \dot{x} \\ r \end{bmatrix}$ où r est tel que $\dot{r} = x$.

$$X_n = \underbrace{\begin{bmatrix} 1 & \Delta & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}_{A''} X_{n-1} + \underbrace{\frac{5}{7}g\Delta \begin{bmatrix} \frac{\Delta}{2} \\ 1 \\ 0 \end{bmatrix}}_{B''} U_{n-1} \tag{1.38}$$

$$Y_n = \underbrace{[1\ 0\ 0]}_{C} X_n \tag{1.39}$$

Commandabilité et observabilité

La matrice de commandabilité est la suivante :

$$\mathcal{C}' = \begin{bmatrix} B'' & A''B'' & A''^2B'' \end{bmatrix} \tag{1.40}$$

$$= \frac{5}{7}g\Delta \begin{bmatrix} \frac{1}{2}\Delta & \frac{3}{2}\Delta & \frac{5}{2}\Delta \\ 1 & 1 & 1 \\ 0 & \frac{1}{2}\Delta & 2\Delta \end{bmatrix} \tag{1.41}$$

Son déterminant est non nul, donc le système est toujours commandable.

Il est immédiat que le système n'est pas observable à cause du nouveau paramètre x_3 dont les conditions initiales ne sont pas observables.

Commande par retour d'état

Notre commande par retour d'état est toujours de la forme :

$$U_i = -K\left(X_i - X^\star\right) \tag{1.42}$$

$X^\star$ étant la consigne, mais on a à présent $K = \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix}$.

Déterminons les conditions sur K pour que ce système soit stable.

$$\begin{aligned} \det(pI - (A'' - B''K)) = {} & p^3 + \left(\frac{5}{14}\Delta^2 g k_1 + \frac{5}{7}\Delta g k_2 - 3\right)p^2 \\ & + \left(-\frac{10}{7}\Delta g k_2 + \frac{5}{14}\Delta^2 g k_3 + 3\right)p \\ & + \frac{5}{14}\Delta^2 g(k_3 - k_1) + \frac{5}{7}\Delta g k_2 - 1 \end{aligned} \tag{1.43}$$

Selon le critère de Routh, il faut que les conditions suivantes soient vérifiées pour que le système soit stable :

$$k_1 > \frac{7}{2\Delta^2 g} \tag{1.44}$$

$$k_2 > \frac{49}{20\Delta g} \tag{1.45}$$

$$k_3 > \frac{7}{5\Delta^2 g} \tag{1.46}$$

$$\frac{5}{7}\left(\frac{\Delta^2 g}{2}k_1 + \Delta g k_2 - \frac{21}{5}\right) \qquad \left(-2\Delta g k_2 + \frac{\Delta^2 g}{2}k_3 + \frac{21}{5}\right)$$
$$> \qquad -\frac{\Delta^2 g}{2}k_1 + \Delta g k_2 + \frac{\Delta^2 g}{2}k_3 - \frac{7}{5} \qquad (1.47)$$

1.3 Limites du modèle

Les limites de ce modèle sont principalement au niveau de notre angle α : on suppose en effet que la commande est instantanée alors qu'en réalité, $\alpha(t)$ est une fonction continue qui « tente de suivre » la commande. Cette dernière peut, elle, être discontinue. Le lien entre la commande et l'angle devrait à son tour être modélisé. Il ferait intervenir des paramètres physiques comme le poids de la bille et les caractéristiques des moteurs et du montage, etc.

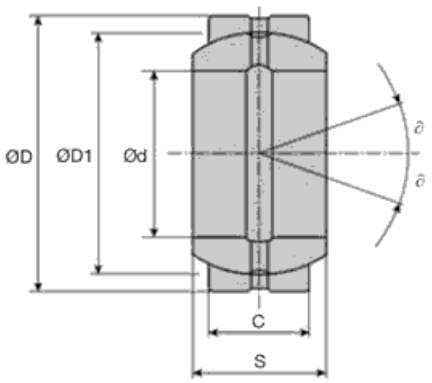
Chapitre 2

La partie mécanique

La partie mécanique de la construction consiste principalement à fabriquer et assembler différentes pièces. Nous avons en fait construit le système en nous basant sur une dalle tactile que nous avions. Le plateau a été dimensionné à partir de celle-ci et les autres pièces en conséquence.

La principale difficulté vient du fait que le plateau doit avoir deux degrés de liberté. On peut imaginer diverses méthodes pour obtenir cela ; le plus simple est sans doute d'utiliser des rotules.

La figure 2.1 présente les caractéristiques essentielles des rotules utilisées.



$\varnothing d$:	8mm $\pm 0,008$
S :	8mm $\pm 0,12$
C :	5mm $\pm 0,24$
$\varnothing D$:	16mm $\pm 0,008$
$\varnothing D1$:	$13,0$mm
∂ :	$15°$
Charge statique :	27 500N

FIGURE 2.1 – Les rotules qui permettent les mouvements du plateau (image : *HPC*).

Le plateau étant constitué d'une plaque de tôle, ces rotules ne peuvent y être fixées directement. Nous avons donc usiné dans une barre d'aluminium (cf. figure 2.2) des supports pour fixer ces rotules au plateau.

Ces supports eux-mêmes doivent êtres fixés au plateau. Nous avons envisagé différent moyens pour ce faire et avons choisi de les coller à l'époxy. Pour assurer une bonne tenue, il faut prendre soin de décaper les surfaces collées. Dans le cas de l'aluminium qui s'oxyde très vite, le décapage et l'application de la colle se font simultanément à l'aide d'un morceau de papier abrasif imperméable recouvert de colle.

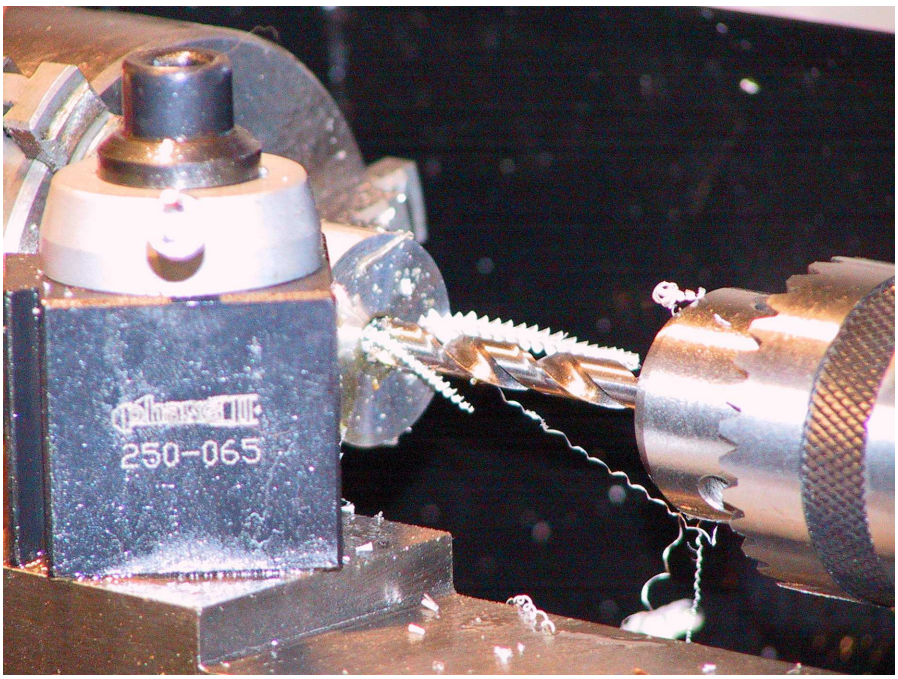


FIGURE 2.2 – Usinage des supports de rotule

Des tiges filetées M4 sont utilisées comme bielles. Là aussi, des pièces (figure 2.4) ont été usinées pour faire le lien avec les rotules d'une part, et avec le bras des servomoteurs d'autre part.

Les supports des servomoteurs sont simplement des plaques de tôle découpées et pliées. Ils sont eux-même vissés sur la base.

La base est une croix de bois faite à partir de chevrons de 5×10 centimètres. Elle assure la stabilité de l'ensemble. Une tige part de son centre et supporte le plateau, via la rotule centrale de celui-ci.

La figure 2.6 montre l'aspect global du système. Les cartes électroniques (voir chapitre suivant) ne sont pas présentes sur cette photographie, mais elles viennent se fixer sur la base.

Dans les améliorations qu'il serait possible d'apporter, on peut noter :

- le montage du plateau sur rotules fait que celui-ci peut tourner autour de l'axe central. Bien que la présence des moteurs l'en empêche normalement, les bras en plastique des servomoteurs utilisés sont relativement souples et peuvent se déformer, laissant le plateau libre de tourner sur quelques degrés. À part le stress mécanique que cela impose aux bras des moteurs, cela n'est toutefois pas gênant ;
- les supports des servomoteurs pourraient être renforcés afin d'éviter un léger jeu dû à leur déformation lorsque l'effort des moteurs change de sens.

FIGURE 2.3 – La face inférieure du plateau, avec les trois supports de rotules.

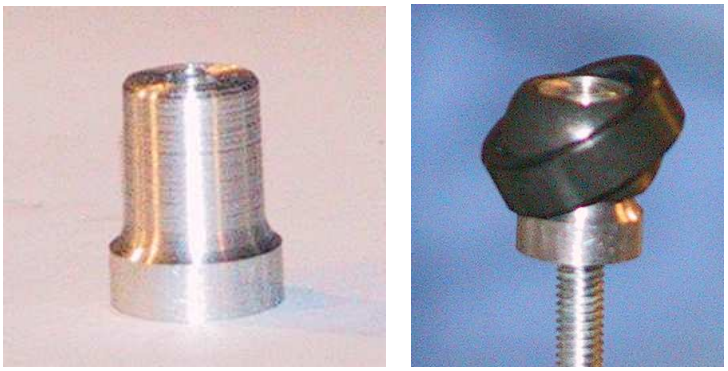


FIGURE 2.4 – Une des pièces usinées pour faire le lien entre les rotules et les bielles (seule à gauche, avec rotule et bielle à droite).

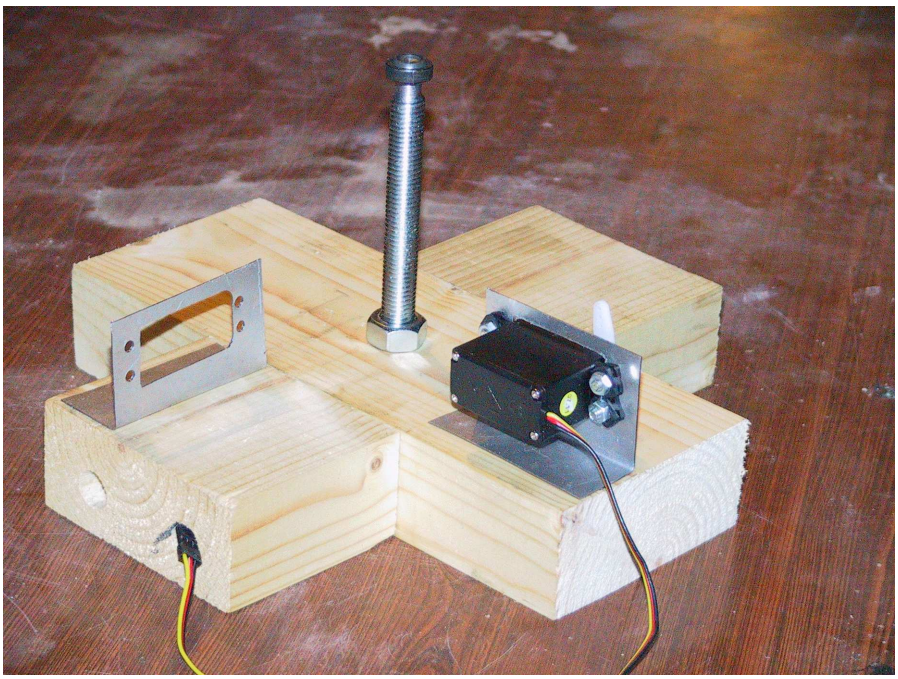


FIGURE 2.5 – La base du montage, avec la tige destinée au support du plateau, les supports des servomoteurs et un servomoteur.

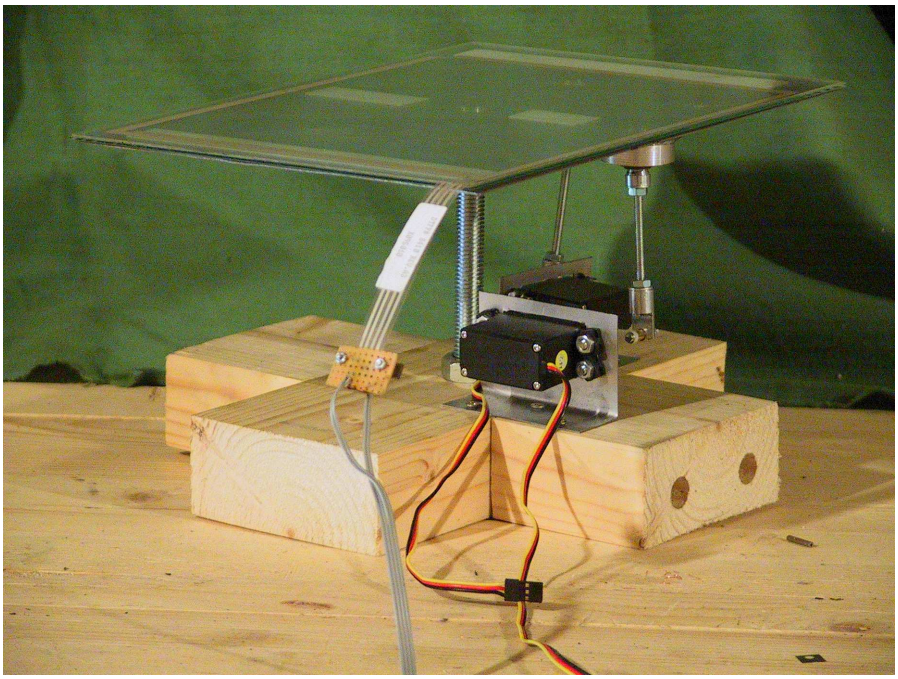


FIGURE 2.6 – Les différentes pièces de la partie mécanique assemblées.

Chapitre 3

Le circuit de contrôle

Le circuit électronique peut se diviser, sur un plan logique, en quatre parties :

1. une partie alimentation, qui fournit les tensions nécessaires ;
2. une partie acquisition qui est chargée d'interfacer les capteurs et le microcontrôleur ;
3. une partie « centrale » constituée du microcontrôleur et des composants qui lui sont liés (quartz, etc.) ;
4. une partie « action » qui fait l'interface entre le microcontrôleur et la partie mécanique.

Pour des questions de facilité de mise au point, nous avons développé des modules (circuits imprimés) distincts pour ces différentes parties (à l'exception de la dernière, très réduite il est vrai).

Revenons sur ces différentes parties un peu plus en détail.

3.1 L'alimentation

L'alimentation est particulièrement simple : elle est construite autour de deux régulateurs qui fournissent les tensions régulées dont les composants ont besoin. Notons que cette alimentation ne fournit pas le 6V utilisé par les servomoteurs, ceux-ci étant directement alimentés séparément.

Il s'agit d'une adaptation directe de celle décrite dans [1], seul le régulateur 3,3 V ayant été remplacé (pour des questions de disponibilité). Son schéma se trouve figure 3.1 ; le circuit imprimé correspondant figure 3.2.

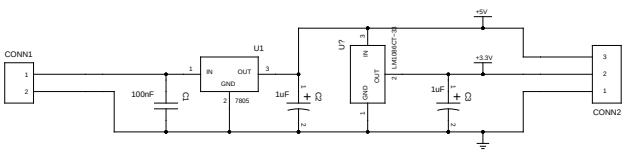


FIGURE 3.1 – Schéma de l'alimentation.

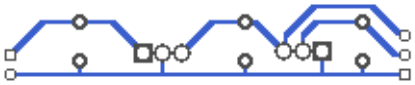


FIGURE 3.2 – Circuit imprimé de l'alimentation, correspondant au schéma de la figure 3.1.

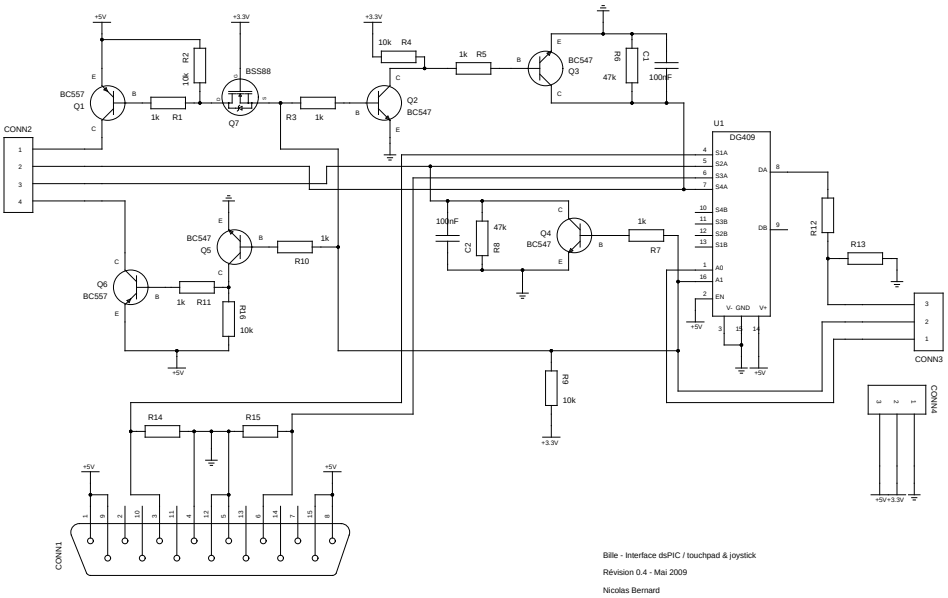


FIGURE 3.3 – Schéma de la partie acquisition.

3.2 La partie acquisition

Dans notre système, il y a deux entrées possible, selon le mode de fonctionnement choisi :

- un joystick en mode manuel ;
- la dalle tactile en mode automatique.

À chaque fois, il faut extraire les coordonnées en x et en y de ces entrées. Comme la dalle tactile impose une lecture séquentielle des coordonnées, nous avons choisi de n'utiliser qu'une seule entrée analogique sur le dsPIC.

Cette partie acquisition prend donc en entrée deux signaux numériques, l'un pour indiquer le mode (automatique ou manuel – `mode`), l'autre pour indiquer si l'on lit x ou y (`select`). En retour, ce module envoi le signal analogique correspondant. Le schéma de cette partie est donné sur la figure 3.3 et son circuit imprimé sur la figure 3.4

3.2.1 Sélection du signal analogique

Nous avons donc au total quatre signaux analogiques potentiels. La dalle tactile ne peut en fournir qu'un à la fois ; le joystick fournit les deux, mais

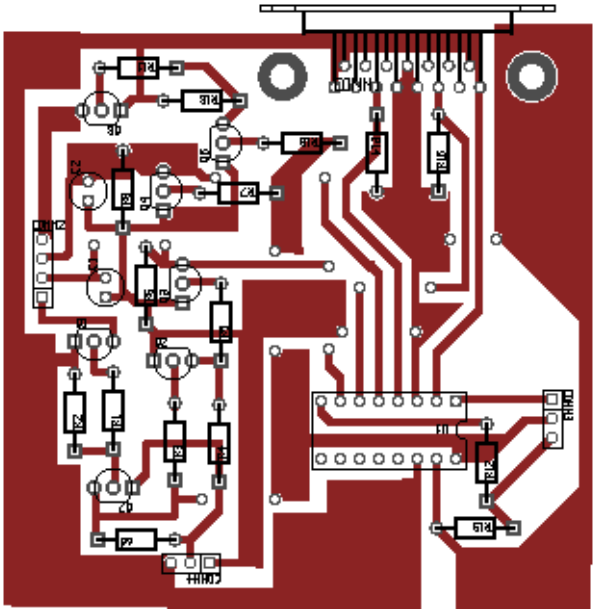


FIGURE 3.4 – Circuit imprimé de la partie acquisition.

ils seront échantillonnés successivement. Afin de n'utiliser qu'une seule entrée analogique sur le dsPIC, nous pouvons donc utiliser un multiplexeur analogique comme le DG409 [3].

Celui-ci est utilisé de manière à ce que l'inverseur de sélection du mode permette de sélectionner un groupe de deux entrées (qui seront x et y pour le périphérique d'entrée choisi), tandis qu'un signal numérique `select` en provenance du microcontrôleur indique s'il faut transmettre x ou y. Notons que la documentation indique qu'une tension supérieure à 2,7V sera interprétée comme un 1 logique ; la commande par le dsPIC peut se faire directement.

Le signal analogique sortant du DG409 est a priori entre 0 et 5V. Le dsPIC nécessite un signal compris entre 0 et 3,3V ; deux résistances forment donc un pont diviseur sur le signal de sortie.

3.2.2 Interface avec le joystick

Nous avons prévu la connexion d'un joystick PC pré-USB, c'est-à-dire muni d'une prise à 15 broches.

Ces joysticks, dans leur expression la plus simple, sont en fait deux résistances variables (0 à 100 kΩ) connectées à une alimentation 5V. Nous avons donc ajouté une résistance vers la masse pour récupérer une tension qui varie en fonction de la position du joystick.

3.2.3 Interface avec la dalle tactile

Notre dalle tactile est une dalle résistive. C'est-à-dire que si l'on applique une tension entre *haut* et *bas*, on a sur *droite* et / ou *gauche* une tension proportionnelle à la « hauteur » du point de pression (on peut faire le parallèle avec la

broche centrale d'une résistance ajustable). L'inverse se produit si l'on applique à présent la tension entre *droite* et *gauche*.

Cela veut dire qu'il faut pouvoir appliquer une tension entre une paire de bornes et avoir l'autre paire déconnectée de l'alimentation pour pouvoir « lire » ce qui s'y trouve, puis inverser les deux paires.

Pour cela, nous avons adapté le circuit proposé dans [9]. Ce circuit réalise ce que nous avons décrit ci-dessus à l'aide de transistors communs, commandés par le signal que nous appelons `select`.

Nos adaptations sont justement liées à ce signal :

- le circuit original utilisait deux signaux de commande distincts et non un seul ;
- notre signal de commande est un signal 3,3V (alors que la dalle est alimentée en +5V [1]), nous avons donc ajouté un FET pour réaliser l'adaptation comme suggéré dans [12] ;
- de plus, nous avons respectivement remplacé les transistors BC546 et BC556 par des transistors BC547 et BC557 car nous avions ces derniers à disposition.

3.2.4 Alternatives et améliorations possibles

A posteriori, il y a un certain nombres de choses que l'on pourrait faire différemment.

Tout d'abord, nous avons découvert récemment qu'il était possible de configurer certaines sorties numériques du dsPIC en collecteur ouvert. Cela signifie qu'il lui serait possible d'« envoyer » un signal de commande en +5V, et donc que le FET n'est pas nécessaire.

Ensuite, il serait possible d'envoyer les quatre signaux analogiques directement sur différentes broches du dsPIC et de choisir celle qui nous intéresse de manière logicielle. Ce choix peut se discuter et est à faire en fonction de l'usage du dsPIC. Ici il serait possible. Dans ce cas, il faudrait toutefois ajouter des résistances car chaque signal nécessitera alors son propre pont diviseur.

Remarquons d'ailleurs que même avec un montage comme le nôtre et son multiplexeur analogique, il pourrait être intéressant de placer un pont diviseur par signal avant le multiplexeur (au lieu d'un pont commun en sortie) afin d'avoir une meilleure adaptation des impédances.

3.3 Le microcontrôleur

Notre microcontrôleur est un dsPIC33FJ128MC802 [7] en boitier SDIP (28 broches). Il fait donc partie de la famille des dsPIC33 de Microchip : il s'agit de microcontrôleurs 16 bits fonctionnant en 3,3V et avec des capacités de traitement du signal.

La programmation de cette puce sera traitée au chapitre suivant et nous ne nous intéressons donc ici qu'à son environnement matériel. Celui-ci, illustré figure 3.5 (circuit imprimé figure 3.6), est très simple et se décompose principalement comme suit :

1. La dalle pourrait en fait sans doute être alimentée en 3,3V.

– un condensateur entre l'entrée V_{cap} et la masse ;
– un quartz entre OSC0 et OSC1 ;
– un circuit de reset.

Le reste du schéma est constitué d'un interrupteur pour choisir le mode de fonctionnement (contrôle automatique par le microcontrôleur ou contrôle manuel avec le joystick), d'une LEDconnectée sur la sortie RB5 du dsPIC et de connecteurs pour relier ce circuit au programmateur, à son alimentation et aux autres circuits.

Nous avons choisi de laisser la possibilité d'évolutions futures de notre montage, c'est pourquoi les broches non utilisées actuellement sont également disponibles sur des connecteurs. Les broches du PWM sont également disponibles sur un connecteur afin de laisser la possibilité d'ajouter un circuit de puissance pour s'interfacer avec d'autres actionneurs.

3.4 Les actionneurs

Il faut un moyen de convertir les ordres donnés par le microcontrôleur en des actions physiques. On peut imaginer divers moyens (pistons et vérins, moteurs, etc.). Nous avons choisi d'utiliser des servomoteurs de modélismes, qui sont à la fois faciles à obtenir et à mettre en œuvre.

3.4.1 Servomoteurs de modélisme

Les servomoteurs de modélisme ont en général trois fils : un fil pour l'alimentation de puissance (6V dans notre cas), un fil pour des impulsions de commande, et une masse commune.

D'après le constructeur [2], les impulsions de commande sont des impulsions carrées d'une durée variant entre 0,9 et 2,1 millisecondes, rafraîchies à 50 Hz et d'amplitude comprise entre 3 et 5 V.

Cela signifie entre autres que les impulsions délivrées par une unique sortie PWM du dsPIC sont d'amplitude suffisante (3,3V). Il s'agit d'un signal de commande, donc il n'est pas nécessaire d'ajouter un étage de puissance, le dsPIC peut contrôler directement les moteurs.

3.4.2 Alternatives et améliorations possibles

Une possibilité pour remplacer les servomoteurs s'ils n'étaient pas assez rapides pourrait être d'utiliser des moteurs de disques durs. Un disque dur standard comporte deux moteurs, l'un qui fait tourner les plateaux, l'autre (figure 3.7) qui positionne la tête de lecture sur le cylindre auquel on veut accéder. Les mouvements de ce dernier sont limités, mais il est à la fois puissant et très rapide.

Dans les disques durs, ces moteurs, appelés *voice-coil motors* à cause de leur mécanisme ressemblant à ceux des haut-parleurs, sont asservis par l'intermédiaire de données spécifiques inscrites sur les plateaux et lues par l'électronique du disque. Sans les plateaux, cet asservissement disparaît, mais pour notre système cela ne semble pas gênant a priori.

Le contrôle de tels moteurs nécessiterait l'usage d'une vraie partie puissance dans notre montage, par exemple basée sur un pont en H L298 : le dsPIC n'est

pas capable de fournir la puissance demandée par ces moteurs directement.

Nous n'avons pas trouvé d'exemple de montages amateurs utilisant de tels moteurs avec un contrôle fin. Le seul exemple que nous ayons trouvé les utilise en tout-ou-rien, c'est-à-dire en butée d'un côté ou de l'autre [4].

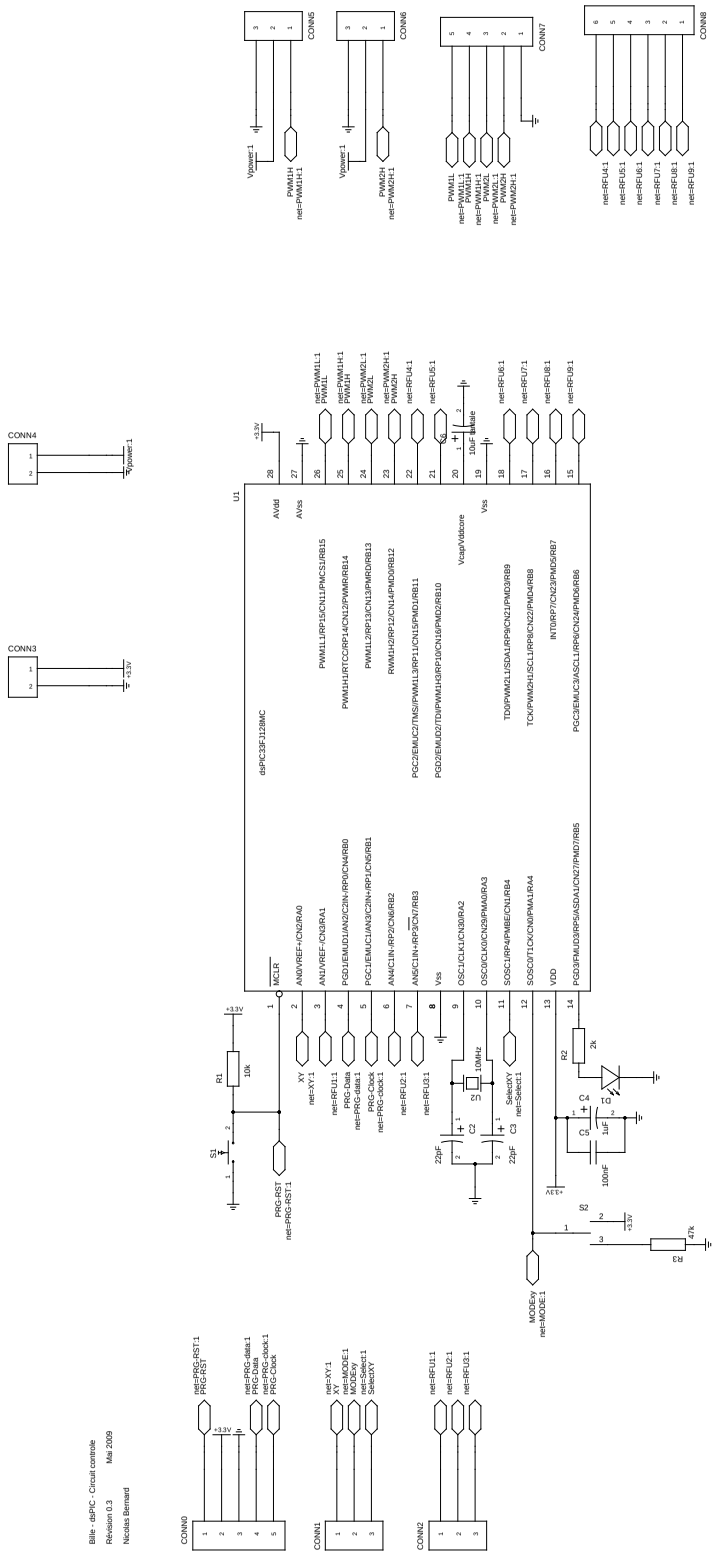


FIGURE 3.5 – Schéma de l'environnement du microcontrôleur.

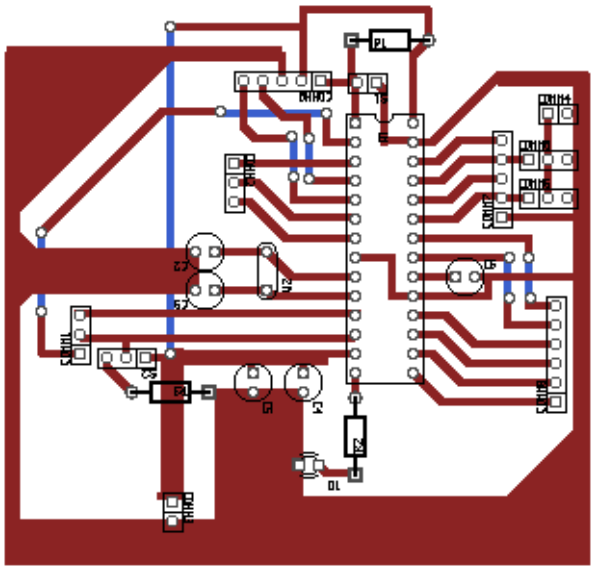


FIGURE 3.6 – Circuit imprimé correspondant au schéma de la figure 3.5. La seconde face a été réduite au minimum afin de pouvoir réaliser un circuit simple face avec des *straps*.

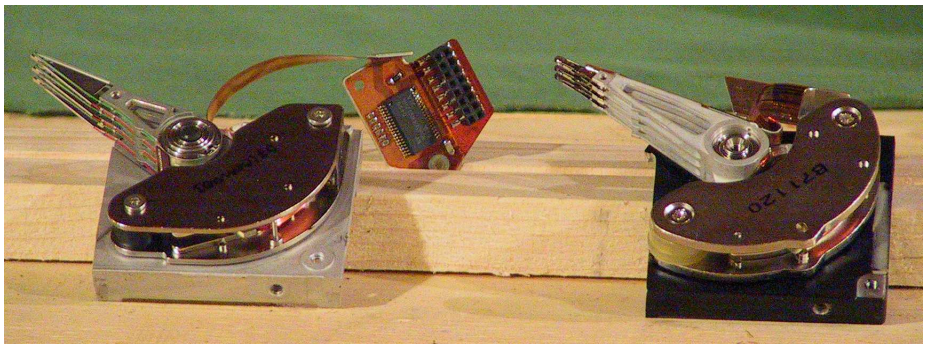


FIGURE 3.7 – Deux *voice-coil motors*. Ces moteurs positionnent les têtes dans les disques durs.

Chapitre 4

La programmation du PIC

Le microcontrôleur dsPIC est le cœur du système. C'est lui qui fait l'interface entre les différentes parties et qui assure la régulation. Sa programmation est donc particulièrement importante. C'est ce que nous explorons dans ce chapitre (*n.b.* : le programme complet se trouve dans l'annexe A).

4.1 Configuration générale

Configuration de l'horloge :

```
/* External crystal, exotic options disabled. */
_FOSC(OSCIOFNC_ON & FCKSM_CSDCMD & POSCMD_HS);
/* We use the crystal (primary) from reset. */
_FOSCSEL(FNOSC_PRI & IESO_OFF);
```

Nous utilisons un quartz de 10 MHz, donc :

$$\mathtt{F_{osc}} = 10\,000\,000 \text{ Hz} \tag{4.1}$$

$$\mathtt{F_{CY}} = \frac{\mathtt{F_{osc}}}{2} = 5\,000\,000 \text{ Hz} \tag{4.2}$$

Cela nous servira ultérieurement pour déterminer les paramètres dépendants du temps.

```
_FWDT(FWDTEN_OFF);                          //Turn off WatchDog Timer
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);  //Turn off code protect
_FPOR( FPWRT_PWR1 );                        //Turn off power up timer
```

4.2 Configuration des entrées / sorties

Pour déterminer la configuration des registres d'entrées / sorties, faisons le tour des différentes broches :

- la broche 2 est utilisée en tant qu'entrée analogique. C'est la seule entrée analogique, le registre `ADPCFG` prend donc la valeur `0xFFFE` ;
- les broches 4 et 5 sont utilisées uniquement pour la programmation du microcontrôleur ;
- la broche 11 comme sortie numérique `RB4`.

- la broche 12 est utilisée comme entrée numérique `RA4` ;
- la broche 14 est utilisée comme sortie numérique `RB5` pour une LED de contrôle ;

On a donc :

```
TRISA = 0x0011; /* RA0 is an [analog] input, RA4 is a digital input,
                   others are outputs.*/
TRISB = 0x0000;
```

4.2.1 Configuration du CAN

Le convertisseur analogique numérique est configuré pour utiliser la broche `AN0` comme canal 0. C'est le seul canal utilisé, la conversion se fait avec douze bits de précision.

La conversion commence automatiquement quand l'échantillonage est terminé, et une interruption est envoyée pour signaler la disponibilité du résultat.

L'échantillonage sera lui commandé par le `TIMER1` de manière à ce qu'il se fasse régulièrement.

```
AD1CON1 = 0x04E0; // SSRC = auto-convert
AD1CON2 = 0;
AD1CON3 = 0;
AD1CON4 = 0;
AD1CHS0 = 0; // CH0 on AN0
AD1CSSL = 0;
_AD1IF = 0;
_AD1IE = 1;
```

Les valeurs lues sont normalisées, de manière à être exploitables par le PID. Ainsi, par exemple, pour l'axe X :

```
float adcval = ADC1BUF0 - 198;
if (adcval < 0)
        adcval = 0;
adcval = adcval / 2893;
```

4.2.2 Configuration du PWM

Les servomoteurs de modélisme utilisés sont commandés par des impulsions carrées d'une durée variant entre $0,9$ et $2,1$ secondes, se répétant toute les 20 millisecondes.

$$\mathtt{P}x\mathtt{TPER} = \frac{\mathtt{F_{CY}}}{\mathtt{F_{PWM}} \times \mathtt{P}x\mathtt{TPER_{prescaler}}} - 1 \tag{4.3}$$

$$\mathtt{P1TPER} = \frac{5\,000\,000}{50 \times \mathtt{P1TPER_{prescaler}}} - 1 \tag{4.4}$$

$$\mathtt{P1TPER_{prescaler}} \in \{1,\ 4,\ 16,\ 64\} \tag{4.5}$$

La valeur 1 ne peut convenir comme $\mathtt{P}x\mathtt{TPER}$ doit tenir dans 16 bits. 4 convient, on aura donc $\mathtt{P1TCON} < 3,2 >= 01_b$

$$\mathtt{P1TPER} = \frac{5\,000\,000}{50 \times 4} - 1 \tag{4.6}$$

$$= \frac{5\,000\,000}{200} - 1 \tag{4.7}$$
$$= 25\,000 - 1 \tag{4.8}$$
$$= 24\,999 \tag{4.9}$$

$$\mathtt{PWM}_{\mathtt{Resolution}} = \log_2 \left(\frac{2\mathtt{F}_{\mathtt{CY}}}{\mathtt{F}_{\mathtt{PWM}} \times \mathtt{P}x\mathtt{TPER}_{\mathtt{prescaler}}} \right) \tag{4.10}$$
$$= \log_2 50\,000 \tag{4.11}$$
$$\approx 15,6 \tag{4.12}$$
$$\approx 16 \text{ bits} \tag{4.13}$$

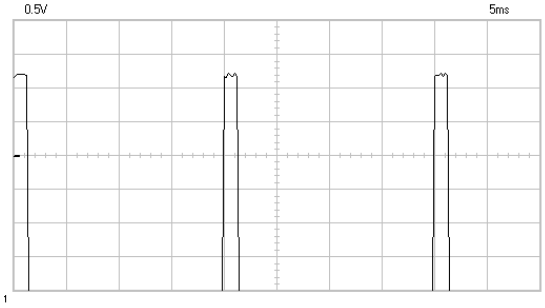


FIGURE 4.1 – Oscillogramme de la sortie PWM du dsPIC pour le contrôle d'un servomoteur de modélisme.

Pour conclure cette section, notons que si les impulsions doivent être répétées au moins toutes les 20 ms pour que le moteur ne passe pas en roues libres, rien n'empêche a priori de les rafraîchir plus fréquemment pour avoir un temps de réponse plus bas quand la commande change. Nos essais avec une fréquence quatre fois supérieure (200 Hz) n'a pas montré de problèmes autres que des légères vibrations à cette fréquence.

4.3 Régulation PID

Il serait tout à fait possible de programmer un régulateur PID (c'est d'ailleurs ce que nous avions commencé par faire), mais ce n'est pas nécessaire : la bibliothèque DSP du dsPIC en fournit déjà une implémentation [5]. Cette implémentation fait usage du module DSP et fait les calculs avec des types `fractional` : ainsi elle peut effectuer les calculs sur des valeurs comprises dans $[-1, 1]$ de manière beaucoup plus rapide qu'en les faisant avec un type `float` traditionnel.

4.3.1 Temps

Le temps est important car l'hypothèse sous-jacente est que les mesures sont faites à intervalles réguliers. C'est l'hypothèse que nous avions faite lors de la modélisation en disant que tous les Δ_i étaient égaux à Δ.

Dans notre cas, nous avons le Timer_1 qui est utilisé pour donner un « top » à intervalles réguliers. Ce signal est uniquement utilisé pour déclencher l'acquisition du CAN. L'interruption provoquée par la fin de la conversion analogique / numérique provoque la mise à jour du régulateur PID avec la nouvelle valeur et l'assignation d'une nouvelle valeur aux PWMs (en fait on ne travaille que sur un axe à la fois, en alternance).

4.3.2 Utilisation du PID DSP

Pour utiliser les fonctions fournies, il faut d'abord déclarer des variables de type `tPID` et les initialiser. Ce type `tPID` est une structure qui comporte les variables relatives à une régulation PID, définie ainsi dans la bibliothèque DSP :

```
typedef struct {
        fractional* abcCoefficients;
        fractional* controlHistory;
        fractional controlOutput;
        fractional measuredOutput;
        fractional controlReference;
} tPID;
```

Les noms des trois derniers membres sont assez explicites : ils correspondent respectivement à la sortie du PID, à la valeur de la mesure (entrée du PID) et à la consigne. Le premier membre est un pointeur sur un tableau de trois coefficients (K_p, K_i et K_d étant bien sûr les gains proportionnel, intégral et dérivé) :

$$\{\ K_p + K_i + K_d\ ,\ -(K_p + 2 \times K_d)\ ,\ K_d\ \}$$

Ce ne sont pas directement les gains qui sont utilisés pour des raisons de vitesse du calcul : il est plus efficace de remplir une fois ce tableau (une fonction `PIDCoeffCalc` est fournie pour le faire) que d'utiliser les gains à chaque fois. Le membre `controlHistory` est un tableau de trois valeurs destiné à garder un historique ; il faut toutefois faire attention en le déclarant car il doit être placé en *espace Y*.

La préparation d'une structure `tPID` se fait donc ainsi :

```
tPID X_axis;
fractional x_abccoef[3];
fractional x_ctlhist[3] __attribute((space(ymemory),far));

X_axis.abcCoefficients = x_abccoef;
X_axis.controlHistory = x_ctlhist;
X_axis.controlReference = Float2Fract(0.4); // consigne

fractional x_PIDcoef[3]; // Gains proportionnel, intégral et dérivé
x_PIDcoef[0] = Float2Fract(0.45); // Gain proportionnel K_p (= k_1)
x_PIDcoef[1] = Float2Fract(0.001); // Gain intégral K_i (= k_3)
x_PIDcoef[2] = Float2Fract(0.1);  // Gain dérivé K_d (= k_2)

PIDCoeffCalc(x_PIDcoef, &X_axis);
PIDInit(&X_axis);
```

La fonction `PID` peut ensuite être appelée régulièrement en stockant préalablement la valeur mesurée dans le membre `measuredOutput` et donne son

résultat dans le membre `controlOutput`, comme nous le faisons dans la routine d'interruption du CAN :

```
X_axis.measuredOutput = Float2Fract(adcval);
PID(&X_axis);
P1DC1 = MID_X + 1000 * Fract2Float(X_axis.controlOutput);
```

La principale difficulté est de trouver des valeurs convenables pour les gains. Rétrospectivement, il serait utile de pouvoir modifier ces gains sans avoir à reprogrammer le dsPIC, que ce soit en ayant un moyen de les régler directement sur la carte électronique (potentiomètres) ou en disposant d'une liaison série.

4.4 Fonctions diverses

Finalement, notons que nous avons défini quelques fonctions auxilliaires pour débugger le programme. Ainsi par exemple la fonction `crash` va boucler en faisant clignoter la LED répétitivement un nombre de fois passé en paramètre.

```
inline void
crash(int bip)
{
        while(1) {
                _LATB5 = 0; // led off
                unsigned long x, i;
                for (x = 0; x < 1000000; x++);
                for (i = 0; i < bip; i++) {
                        _LATB5 = 1;
                        for (x = 0; x < 100000; x++);
                        _LATB5 = 0;
                        for (x = 0; x < 100000; x++);
                }
        }
}
```

Bien que cette fonction soit très simple, elle permet de distinguer un problème d'un autre. On peut par exemple définir les interruptions suivantes qui vont appeler cette fonction et feront clignoter différement la LED selon le problème rencontré :

```
void __attribute__((__interrupt__, auto_psv))
_AddressError(void)
{
        crash(1);
}

void __attribute__((__interrupt__, auto_psv))
_StackError(void)
{
        crash(2);
}
```

Conclusion

Ce projet nous a permis d'explorer le programme de la LIE de manière transversale (et même au-delà, comme la partie mécanique).

Le système réalisé (figure 4.2) est résolument expérimental et nous avons indiqué dans ce rapport un certain nombre d'améliorations — ou du moins de modifications — qui pourraient être apportées pour une « version 2 ».

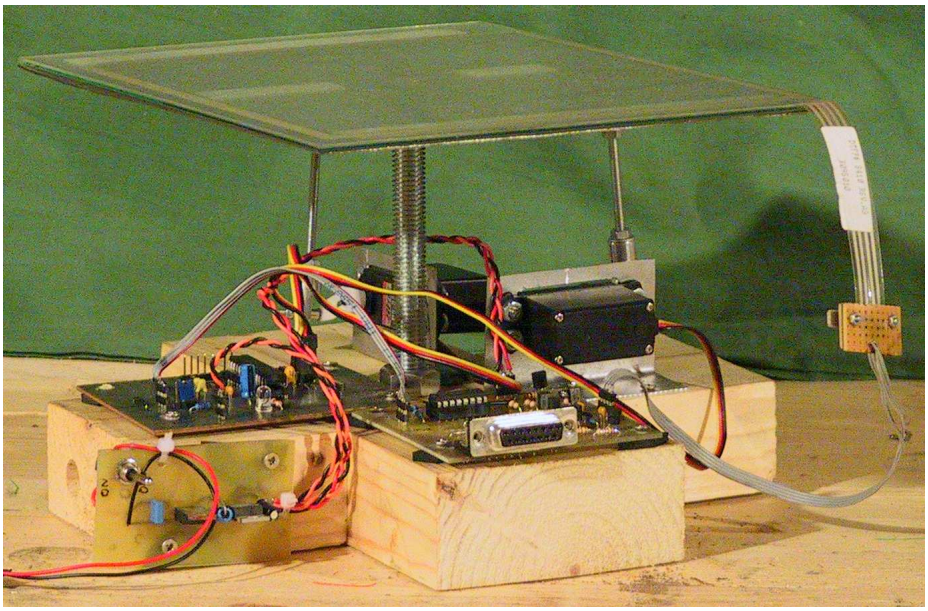


FIGURE 4.2 – L'aspect de notre système assemblé.

La principale difficulté rencontrée a été le manque de sensibilité de la dalle tactile (de récupération et sans documentation, ni fournie, ni sur l'Internet), dont nous n'avons pris réellement conscience qu'une fois le montage terminé. Nous espérions au départ utiliser comme bille des billes de flipper (diamètre usuel : 2,7 cm), mais elles sont apparues trop légères (80 grammes) pour être détectées de manière fiable par la dalle. Après plusieurs essais, nous avons finalement trouvé des billes de fer forgé d'un poids suffisant. Mais elles font 270 grammes et 4 cm de diamètre ! Un tel poids n'avait bien sûr pas été prévu lors de la conception !

FIGURE 4.3 – Billes : à gauche, une bille de flipper, dont l'usage avait été initialement prévu. À droite, une bille de fer forgé, suffisamment lourde pour être détectée de manière fiable par la dalle tactile.

Le contrôle en mode manuel atteste qu'il reste a priori possible de maîtriser la bille. En dépit de cela, nous n'avons pas pour le moment réussi à déterminer des paramètres satisfaisants pour le régulateur PID.

Annexe A

Le programme du dsPIC

Nous utilisons le `Makefile` suivant pour automatiser la compilation du programme (vers un fichier `.hex`) sous Linux, avec une installation « ordinaire » de la version Windows de `MPLAB` avec l'émulateur `wine`.

```
NAME=ball2
CC=wine ~/.wine/drive_c/Program\ Files/Microchip/MPLAB\ C30/\
    bin/bin/pic30-coff-gcc.exe
CFLAGS=-Wall -std=c99 -mcpu=33FJ128MC802 -I ~/.wine/drive_c/\
    Program\ Files/Microchip/MPLAB\ C30/support/dsPIC33F/h
LDLIBS=-Xlinker -T -Xlinker ~/.wine/drive_c/Program\ Files/\
    Microchip/MPLAB\ C30/support/dsPIC33F/gld/p33FJ128MC802.gld\
    -Xlinker -ldsp-coff
B2H=wine ~/.wine/drive_c/Program\ Files/Microchip/MPLAB\ C30/\
    bin/bin/pic30-coff-bin2hex.exe

all: $(NAME).hex

$(NAME).o: $(NAME).c Makefile
	$(CC) $(CFLAGS) -c $(NAME).c -o $(NAME).o

$(NAME): $(NAME).o Makefile
	$(CC) $(LDFLAGS) $(NAME).o -o $(NAME) $(LDLIBS)

$(NAME).hex: $(NAME) Makefile
	$(B2H) $(NAME)
```

Le programme lui-même se trouve ci-dessous. La partie principale est la routine de traitement d'interruption du CAN. La fonction `main` sert principalement pour l'initialisation.

```
#include <math.h>
#include <stddef.h>

#include "p33Fxxxx.h"
#include "dsp.h"

typedef _Bool bool;
#define false 0
```

```
#define true 1

// Oscillator Configuration
_FOSC(OSCIOFNC_ON & FCKSM_CSDCMD & POSCMD_HS);

_FOSCSEL(FNOSC_PRI & IESO_OFF);        //Oscillator Selection
_FWDT(FWDTEN_OFF);                     //Turn off WatchDog Timer
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);  //Turn off code protect
_FPOR( FPWRT_PWR1 );                   //Turn off power up timer

// P1DC1: X
// P1DC2: Y

// Motors/PWM
#define MID_X 3200
#define MID_Y 4000

// touchpad/ADC
#define A_X_MID 1750
#define A_Y_MID 1738

#define LED _LATB5
#define SELECT_XY _LATB4
#define MODE _RA4

// modes
#define MANUAL 0
#define AUTO 1

/* The "meaning" of SELECT_XY is reversed between manual and
   automatic mode! */
#define X_AXIS 0
#define Y_AXIS 1
#define Y_AXIS_AUTO 0
#define X_AXIS_AUTO 1

void
initPWM(void)
{
        P1TCON = 0x8004; // prescaler set to 4, other bits at 0
        P1TPER = 6249; // 200 Hz
        //P1TPER = 24999; // 50 Hz
        PWM1CON1 = 0x0330; // we are only using PWM1H1 and PWM1H2
        PWM1CON2 = 0;
        P1DC1 = MID_X;
        P1DC2 = MID_Y;
}

void
initADC(void)
{
        AD1CON1 = 0x04E0; // SSRC = auto-convert
        AD1CON2 = 0;
        AD1CON3 = 0;
```

```
        AD1CON4 = 0;
        AD1CHS0 = 0; // CH0 on AN0
        AD1CSSL = 0;
        _AD1IF = 0;
        _AD1IE = 1;
}

inline void
crash(int bip)
{
        while(1) {
                LED = 0; // led off
                unsigned long x, i;
                for (x = 0; x < 1000000; x++);
                for (i = 0; i < bip; i++) {
                        LED = 1;
                        for (x = 0; x < 100000; x++);
                        LED = 0;
                        for (x = 0; x < 100000; x++);
                }
        }
}

void __attribute__((__interrupt__, auto_psv))
_AddressError(void)
{
        crash(1);
}

void __attribute__((__interrupt__, auto_psv))
_StackError(void)
{
        crash(2);
}

void __attribute__((__interrupt__, auto_psv))
_MathError(void)
{
        crash(3);
}

static int init = 0;
static long man_x_init = 0;
static long man_y_init = 0;

tPID X_axis;
tPID Y_axis;

static int changedp = 1;
static bool read_y;

static int counter = 0;

void __attribute__((__interrupt__, auto_psv))
```

```
_ADC1Interrupt(void)
{
120         if (MODE == AUTO) {
                changedp = 1;
                _AD1IF = 0;
123             if (ADC1BUF0 < 150)
                        return;

126             if (read_y) {
                        if (ADC1BUF0 < 500) {
                                counter++;
129                             if (counter > 30)
                                        P1DC2 = MID_Y;
                                return;
132                     }
                        counter = 0;
                        float adcval = ADC1BUF0 - 595;
135                     if (adcval < 0)
                                adcval = 0;
                        adcval = adcval / 2310;
138                     Y_axis.measuredOutput = Float2Fract(adcval);
                        PID(&Y_axis);
                        P1DC2 = MID_Y + 15. * 57.3
141                             * asinf(Fract2Float(Y_axis.controlOutput));
                        changedp = 2;

144                     read_y = false;         // prepare next
                        SELECT_XY = X_AXIS_AUTO;
                } else {
147                     if (ADC1BUF0 < 190) {
                                counter++;
                                if (counter > 30)
150                                     P1DC1 = MID_X;
                                return;
                        }
153                     float adcval = ADC1BUF0 - 198;
                        if (adcval < 0)
                                adcval = 0;
156                     adcval = adcval / 2893;
                        adcval = 1 - adcval;
                        X_axis.measuredOutput = Float2Fract(adcval);
159                     PID(&X_axis);
                        P1DC1 = MID_X + 15. * 57.3
                                * asinf(Fract2Float(X_axis.controlOutput));
162                     changedp = 2;

                        read_y = true;
165                     SELECT_XY = Y_AXIS_AUTO;
                }
        } else { // Manual Mode
168             if (read_y) {
                        read_y = 0;
                        SELECT_XY = X_AXIS;
171                     if (man_y_init == 0) {
```

```
                                if (init < 3)
                                        init++;
                                else
                                        man_y_init = ADC1BUF0;
                        } else {
                                long tmp = ADC1BUF0 - man_y_init;
                                if (tmp > 0)
                                        tmp = tmp;
                                else
                                        tmp = 10 * tmp;
                                P1DC2 = MID_Y + tmp;
                        }
                } else {
                        read_y = 1;
                        SELECT_XY = Y_AXIS;
                        if (man_x_init == 0)
                                if (init < 3)
                                        init++;
                                else
                                        man_x_init = ADC1BUF0;
                        else {
                                long tmp = ADC1BUF0 - man_x_init;
                                if (tmp > 0)
                                        tmp = tmp;
                                else
                                        tmp = 10 * tmp;
                                P1DC1 = MID_X + tmp;
                        }
                }
                int x;
                for (x = 0; x < 200; x++);
                _SAMP = 1;
                _AD1IF = 0;
        }

}



void __attribute__((__interrupt__, auto_psv))
_T1Interrupt(void)
{
        if (!changedp)
                crash(4);
        changedp = 0;
        _SAMP = 1; // start conversion
        _T1IF = 0;
}

void
inittimer1(void)
{
        T1CON = 0;
        TMR1 = 0;
```

```
        PR1 = 25000; // 400 Hz
        _T1IF = 0;
        _T1IE = 1;
}

int
main(void)
{
        unsigned long x;
        static fractional x_abccoef[3];
        static fractional x_ctlhist[3] __attribute((space(ymemory),far));
        static fractional y_abccoef[3];
        static fractional y_ctlhist[3] __attribute((space(ymemory),far));


        fractional x_PIDcoef[3];
        x_PIDcoef[0] = Float2Fract(0.89); //Kpx
        x_PIDcoef[1] = Float2Fract(0.0); //Kix
        x_PIDcoef[2] = Float2Fract(0.0); //Kdx

        fractional y_PIDcoef[3];
        y_PIDcoef[0] = Float2Fract(0.87); //Kpy
        y_PIDcoef[1] = Float2Fract(0.0); //Kiy
        y_PIDcoef[2] = Float2Fract(0.0); //Kdy

        X_axis.abcCoefficients = x_abccoef;
        X_axis.controlHistory = x_ctlhist;
        X_axis.controlReference = Float2Fract(0.4);

        Y_axis.abcCoefficients = y_abccoef;
        Y_axis.controlHistory = y_ctlhist;
        Y_axis.controlReference = Float2Fract(0.5);

        ADPCFG = 0xFFFE; // all ADC pins digital but AN0

        TRISA = 0x0011; /* RA0 is an [analog] input,
                           RA4 is a digital input, others are outputs. */
        TRISB = 0x0000;

        LED = 1;
        if (MODE == MANUAL)
                SELECT_XY = Y_AXIS;
        else
                SELECT_XY = Y_AXIS_AUTO;
        read_y = true;

        initPWM();
        for (x = 0; x < 1000000; x++); // stage goes level

        PIDCoeffCalc(x_PIDcoef, &X_axis);
        PIDCoeffCalc(y_PIDcoef, &Y_axis);
        PIDInit(&X_axis);
        PIDInit(&Y_axis);

```

```
        initADC();
        if (MODE == AUTO)
282             inittimer1 ();

        AD1CON1bits.ADON = 1;
285     if (MODE == AUTO)
            T1CONbits.TON = 1;
        else
288             _SAMP = 1; // take the first sample

        while(1)
291     {
            for (x = 0; x < 200000; x++);
            LED = ~LED;
294     }
        return 0;
}
```

Bibliographie

[1] Lotfi Baghli. Mobrob – schéma, 2008.

[2] Hitec. *General Servo Information*, 2 edition, 2002. Disponible en ligne : `http://www.hitecrcd.com/product_file/file/44/Servomanual.pdf`.

[3] Intersil. *DG408, DG409 Data Sheet*, juin 1999. Fichier 3283.5 ; disponible en ligne.

[4] L. P. Maguire, S. Szilagyi, and R. E. Scholten. High performance laser shutter using a hard disk drive voice-coil actuator. *Review of Scientific Instruments*, 75(9) :3077–3079, septembre 2004.

[5] Microchip. *MPLAB® C30 DSP Library*. Installée dans le répertoire `docs/dsp_lib` de MPLAB C30.

[6] Microchip. *16-bit Language Tools Libraries*, 2008. Document DS51456E ; disponible en ligne sur le site de Microchip : `http://ww1.microchip.com/downloads/en/DeviceDoc/51456E.pdf`.

[7] Microchip. *dsPIC33FJ32MC302/304, dsPIC33FJ64MCX02, and dsPIC33-FJ128MCX02/X04 Data Sheet*, 2008. Document DS70291B ; disponible en ligne sur le site de Microchip : http ://ww1.microchip.com/downloads/en/DeviceDoc/70291B.pdf.

[8] Microchip. *MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs User's Guide*, 2008. Document DS51284H ; disponible en ligne sur `http://ww1.microchip.com/downloads/en/DeviceDoc/51284H.pdf`.

[9] Dusan Mihajlovic. Bon. Maintenant il vous faut un écran tactile. Article publicitaire pour *mikroElectronika*, janvier 2009. Disponible en ligne : `http://www.mikroe.com/en/article/09/01/`.

[10] National Semiconductor. *LM1086 1.5A Low Dropout Positive Regulators*, juin 2005. Disponible en ligne sur http ://www.national.com/ds/LM/LM1086.pdf.

[11] Philips Semiconductors. *BC546, BC547 NPN general purpose transistors*, avril 1999. Disponible en ligne.

[12] Herman Schutte. *Bi-directional level shifter for I^2C-bus and other systems*. Philips Semiconductors, août 1997. Application Note AN97055 ; disponible en ligne sur le site de NXP (anciennement Philips Semiconductors) : http ://www.standardics.nxp.com/support/documents/i2c/pdf/an97055.pdf.

[13] Siemens. *BSS88 SIPMOS Small-Signal Transistor*, mai 1997. Disponible en ligne.