GEII
Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

# Motorisation et pilotage d'un fauteuil pour personne à mobilité réduite

## RAPPORT DE PROJET

*Année 2017-2018*

***Pierre JULITA***

**IUT de Belfort**

**Licence Pro VEGA**

Tuteur : Mr Patrick HIEBEL

# Table des matières

## Remerciements

Ce projet de motorisation d'un fauteuil pour personne à mobilité réduite s'est déroulé dans le cadre des projets tutorés de Licence Pro VEGA. Au sein de l'IUT de Belfort Montbéliard.

Tout d'abord je tiens à remercier Mr Patrick HIEBEL, tuteur du projet, qui m'a suivi tout au long du projet.

Je tiens également à remercier toute l'équipe pédagogique pour l'aide qu'elle nous a apportée tout le long de l'année.

## Introduction

Le projet tutoré a pour but de mettre en application nos connaissances acquises tout au long de l'année, ainsi que notre esprit d'initiative pour répondre à un cahier des charges.

Le but de ce projet est de créer de nouvelles lois de commandes qui vont permettre de commander un fauteuil électrique à l'aide d'un joystick. Le projet de motorisation ce fauteuil existe déjà depuis maintenant un certain temps à l'IUT, il a fallu reprendre un projet déjà existant et de le comprendre avant de pouvoir travailler dessus correctement.

## Matériel

Lorsque nous avons repris le projet en début d'année il était constitué des matériels suivants :

- Le fauteuil complet
- 2 batteries 12V (24V total)
- 2 commandes de puissance pour moteurs DC (Modules MD03, hacheurs 4 quadrants)
- 2 motoréducteurs à renvoi d'angle associés à chaque moteur
- 1 joystick inductif
- Carte de commande avec PIC 16F877

## Schéma synoptique

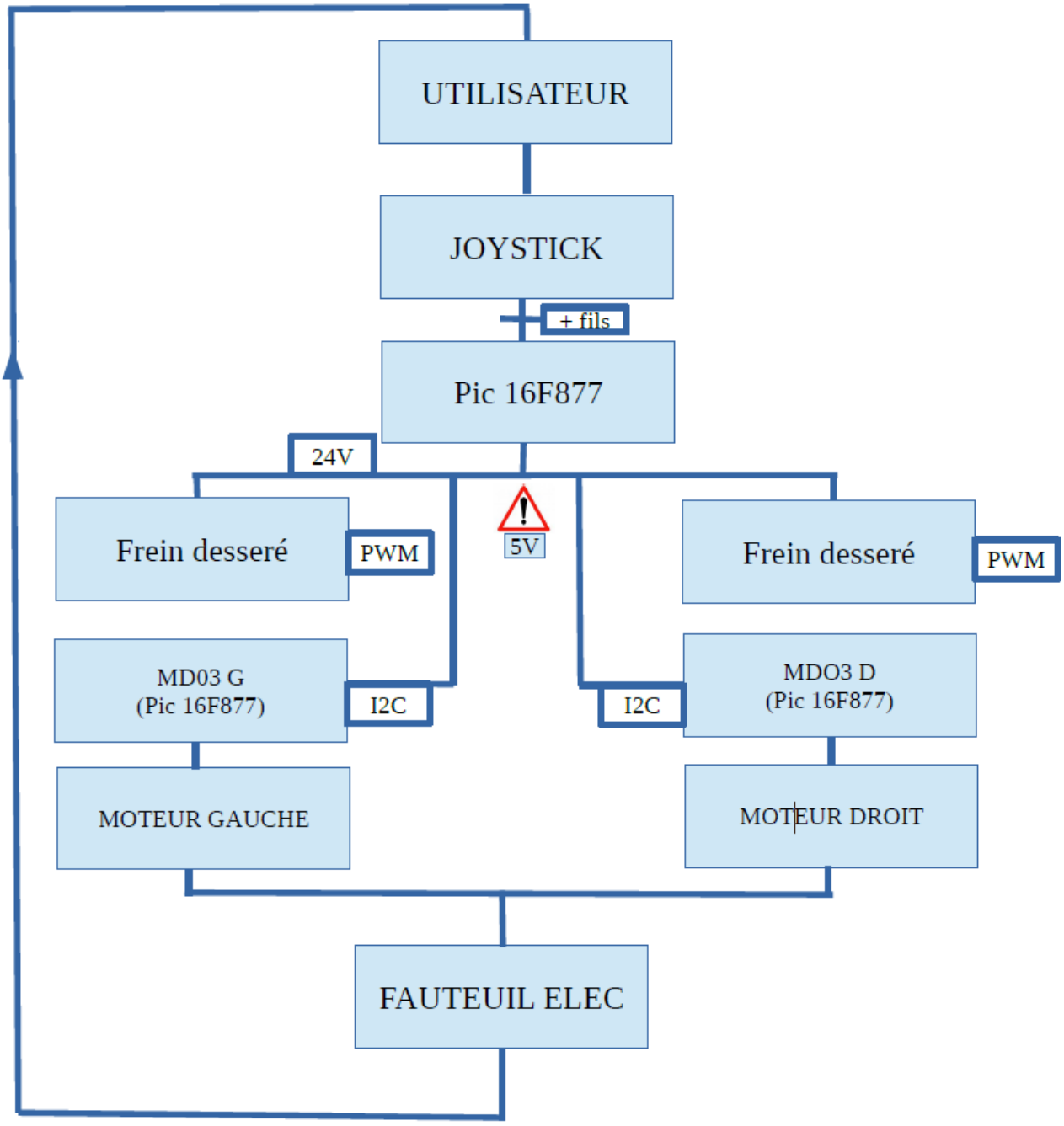



*Figure 1 Schéma synoptique*

Le fonctionnement du système est le suivant.

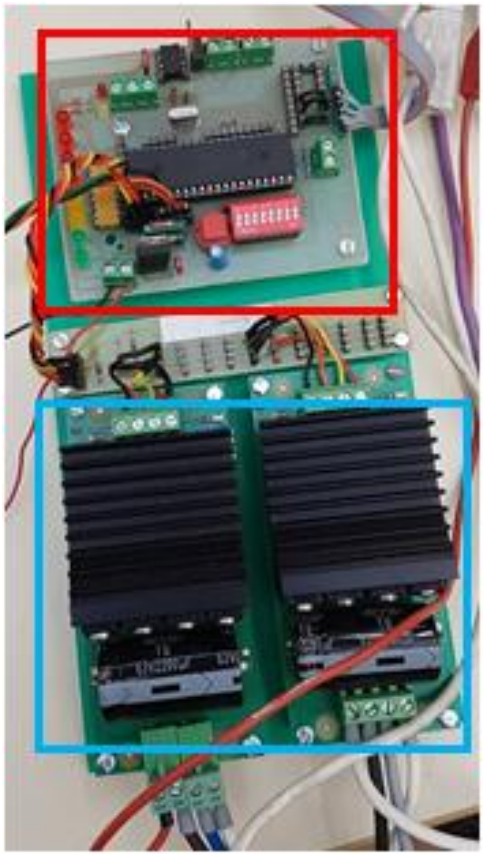
L'utilisateur agira directement sur son joystick. Celui-ci enverra ses informations sur la carte de commande du fauteuil où elle sera reçue par le PIC. A la mise sous tension du fauteuil, les freins des moteurs se débloqueront automatiquement au-dessus d'un seuil de 18V.

Les freins fonctionnent en logique négative, ils sont actifs lorsqu'il n'y a pas de tension. Cela est dû à des raisons de sécurité, si l'utilisateur est en pente et qu'il y a un défaut électrique au niveau de l'alimentation, le fauteuil ne dévalera pas la pente, les freins se bloqueront automatiquement dès qu'ils verront à leurs bornes une tension inférieure à 5V.

Les tensions analogiques du joystick sont ensuite traitées par le PIC qui enverra des signaux des commandes aux deux modules de puissances MD03 via une trame I²C. Les 2 hacheurs 4 quadrants piloteront ainsi chacun des 2 moteurs gauches et droites du fauteuil.

Au début de notre projet, la première chose qui nous a été demandé fus de reprendre dans un premier temps simplement le pilotage du fauteuil via joystick, puis ensuite d'ajouter de nouvelles fonctions comme un mode de fonctionnement petite vitesse et grande vitesse qui auront des marges de manœuvre différentes en virage. En petite vitesse nous aurions des mouvements plus amples pour faire des virages serrés et à grande vitesse des mouvements plus restreints pour éviter une éventuelle sortie de route.

## Le microcontrôleur



Carte de commande avec PIC

Carte de puissance hacheur 4 quadrants (modules MD03)

*Figure 2 Modules de commande et puissance*

Nous disposons d'un PIC 16F877A (Figure 3). Nous utilisons les entrées analogiques pour l'acquisition des tensions du joystick (en rouge), la sortie CCP1 pour le signal MLI (violet), les deux broches RC3 et RC4 pour la communication I²C (orange), les broches RC6 et RC7 pour la programmation du PIC en RS232 (vert) et le PORTB pour différents modes d'opérations (bleu). Le PORTB nous permet une visualisation des données sur des LEDs pour un débogage rapide. Ça nous permets de visualiser rapidement l'activité du microcontrôleur.

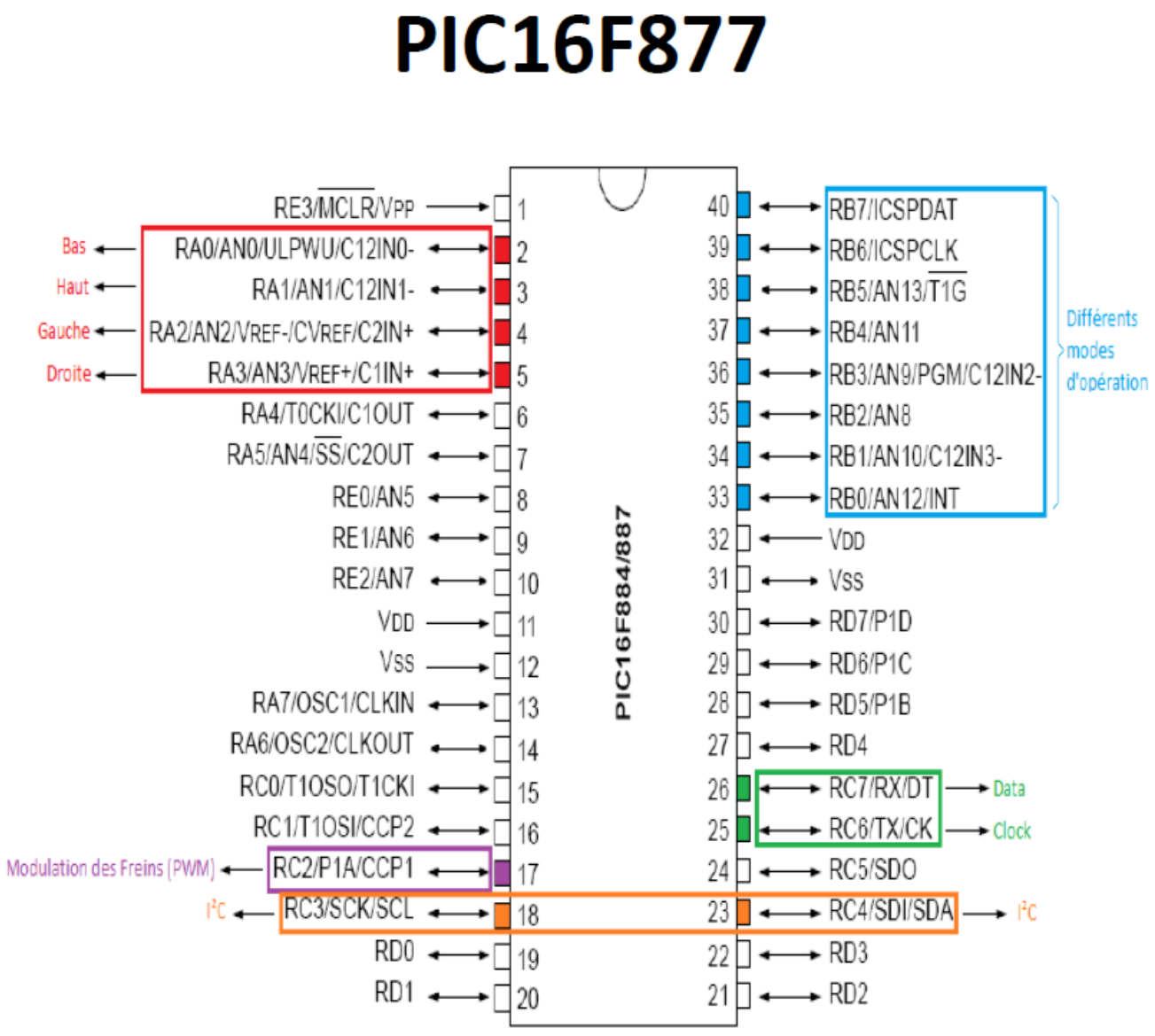



*Figure 3 PIC 16F877*

## Le joystick

Le principe de fonctionnement du joystick est de transformer une position mécanique dans un espace défini en un signal électrique. Notre joystick nous permettra de diriger le fauteuil dans le sens où l'utilisateur l'incline. Il est composé de 5 bobines qui lui fournissent une tension induite.

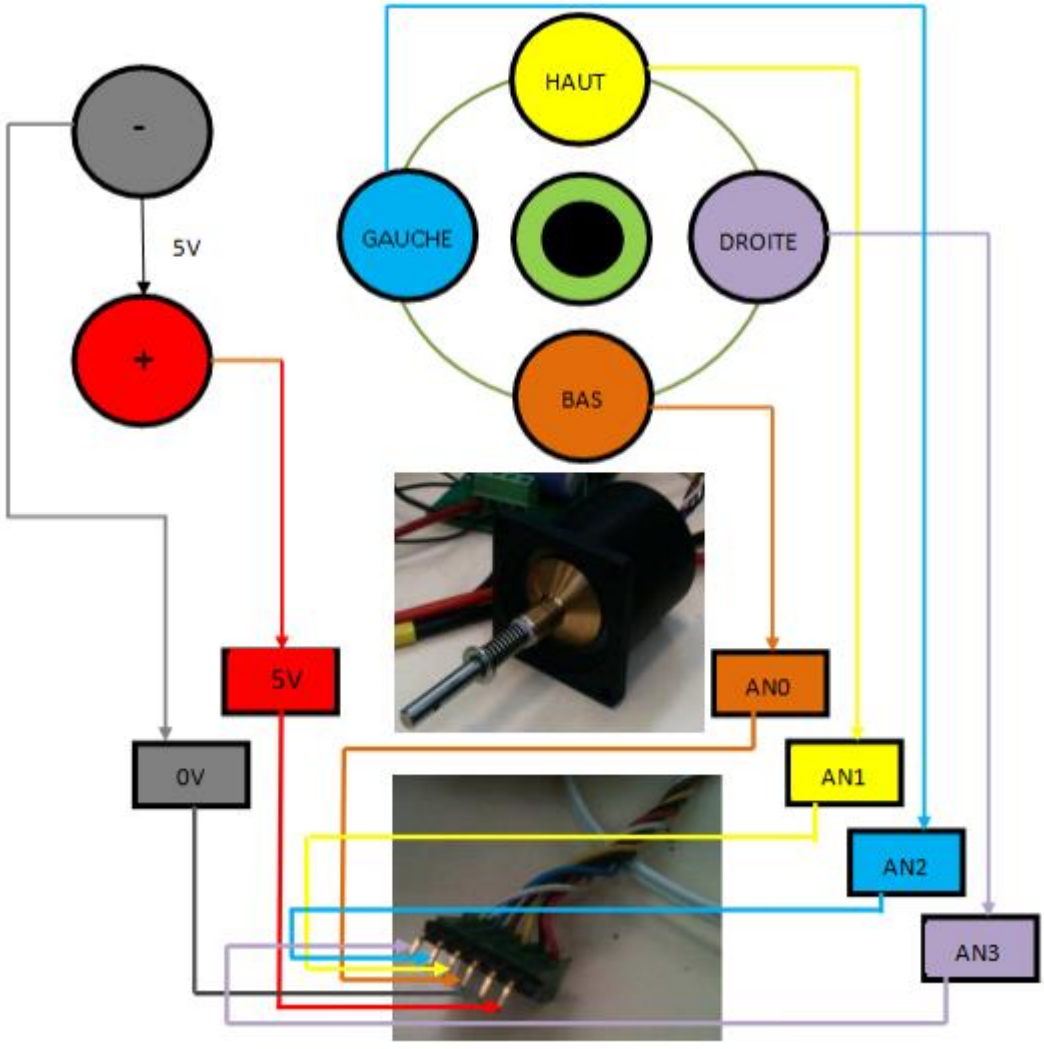



*Figure 4 Schéma du joystick*

Lorsque l'on incline le joystick dans une certaine position, la tension induite qui est donnée par chaque bobine change en fonction de cette inclinaison. Cela est dû à la présence d'une cinquième bobine placée au centre du joystick. Nous avons observé une variation de tension allant de 0.4V à 4.7V. Une fois la conversion Analogique numérique faite, nous obtenons une valeur sur 8 bits variant entre 22 et 239.

## Loi de commande des moteurs

Il y a différentes façons de faire des virages avec le moteur. Plus la vitesse est basse, plus le virage sera serré. La méthode retenue pour tourner et de jouer sur un différentiel de vitesse entre les deux roues. A partir du moment où le fauteuil avance, si l'utilisateur veut tourner à droite ou à gauche, la vitesse de la roue intérieure du virage diminuera tandis que la vitesse de la roue extérieure au virage augmentera. La différence de vitesse de rotation entre les deux roues sera d'autant plus importante que le virage sera grand. En revanche, une exception sera faite à vitesse nulle, la loi de commande ne sera pas la même qu'à vitesse > 1. Sur place, nous imaginons que la personne veut faire un virage serré sur lui-même. Dans ce cas, si vitesse = 0, les deux roues tourneront dans le sens opposé.

Après une première mise en forme de cette loi de commande, nous avons observé des mouvements parasites du fauteuil dû à la sensibilité du joystick trop fine. En effet, lorsque nous voulions avancer en ligne droite, nous ne laissions pas réellement le joystick pile poil sur l'axe Y (x=0), nous avions des petites erreurs de position ce qui ne faisait pas avancer le fauteuil bien droit mais dérivait d'un côté ou de l'autre. Nous n'avions qu'une tolérance de 1 bit d'erreur pour un déplacement sur un des 4 axes.

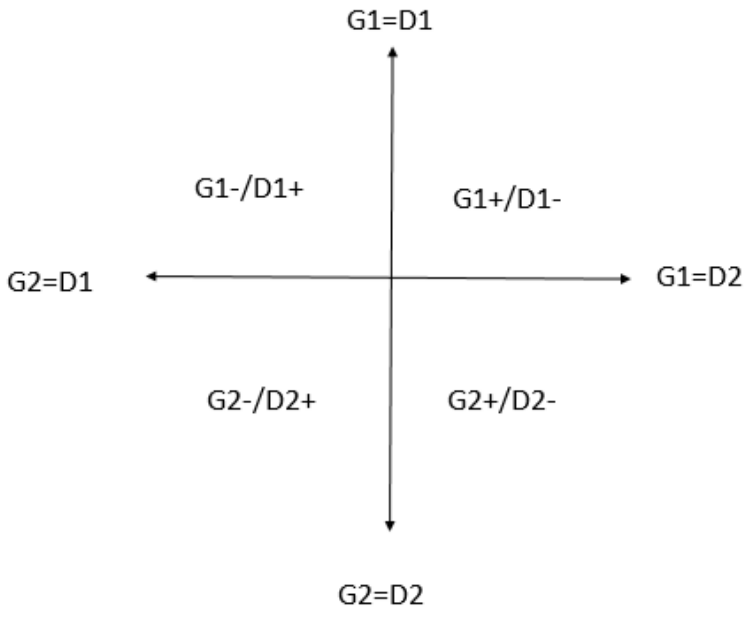



*Figure 5 Position des axes du joystick*

Sur cette figure ; G : gauche et D : Droite

1 : marche avant et 2 : marche arrière. Les indications « + » et « - » indique quelle roue tourne plus vite que l'autre.

Nous avons ainsi mis en place un système admettant une tolérance sur nos 4 axes. Ce système est représenté de la façon suivante, nous créons dans le programme une bande de 10 bits de large sur les 4 axes. Lorsque le joystick est dans cette bande « morte », il ne peut lire que ses valeurs dans un sens. Si nous sommes sur l'axe Y par exemple et que nous nous situons dans cette zone de tolérance, peu importe si nous dérivons légèrement à gauche ou à droite de l'axe Y, sa valeur en abscisse ne sera pas lue. Il faut en revanche sortir de cette bande si nous voulons amorcer un virage dans un sens ou dans l'autre.

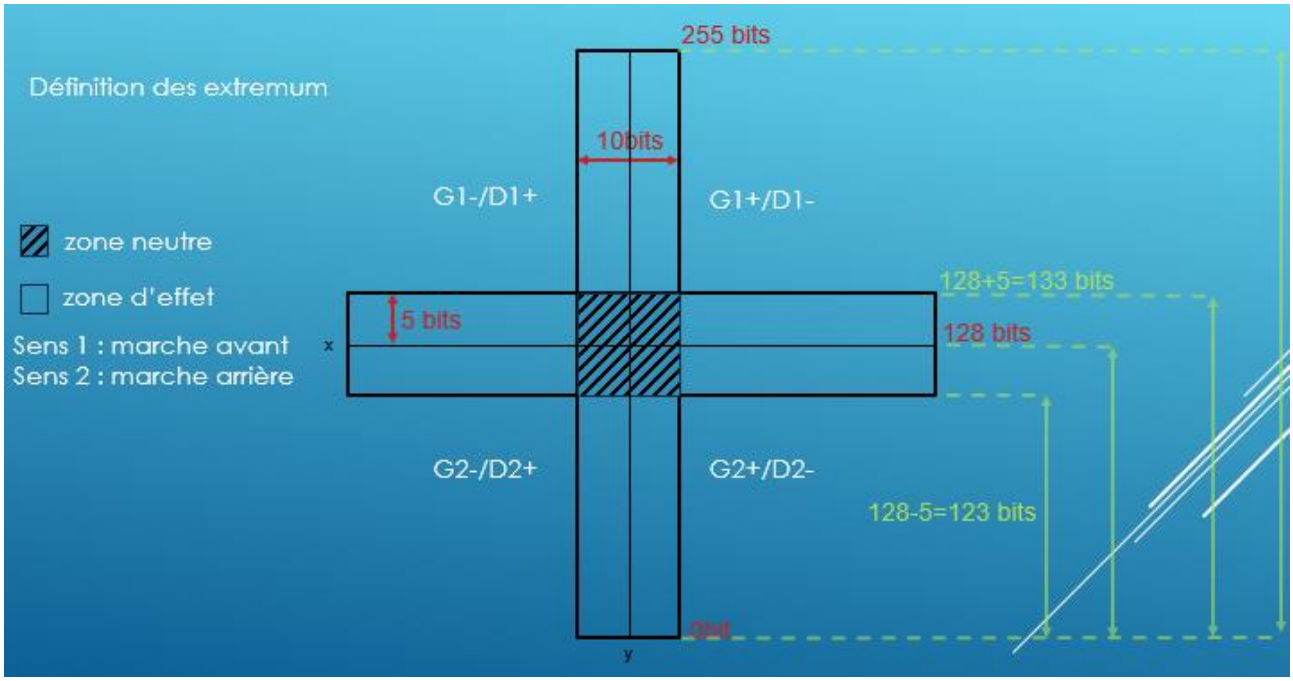



*Figure 6 Définition des extremum*

Cette représentation admet une zone morte en son centre de 10 bits². Cela nous convient également puisque au repos, il arrivait au fauteuil d'avoir des mouvements parasites puisqu'il n'existant qu'un seul point neutre à l'intersection des deux axes, donc infime. Une simple petite perturbation due à un choque mécanique et il bougeait dans un sens.

Voici une représentation de la commande des roues.

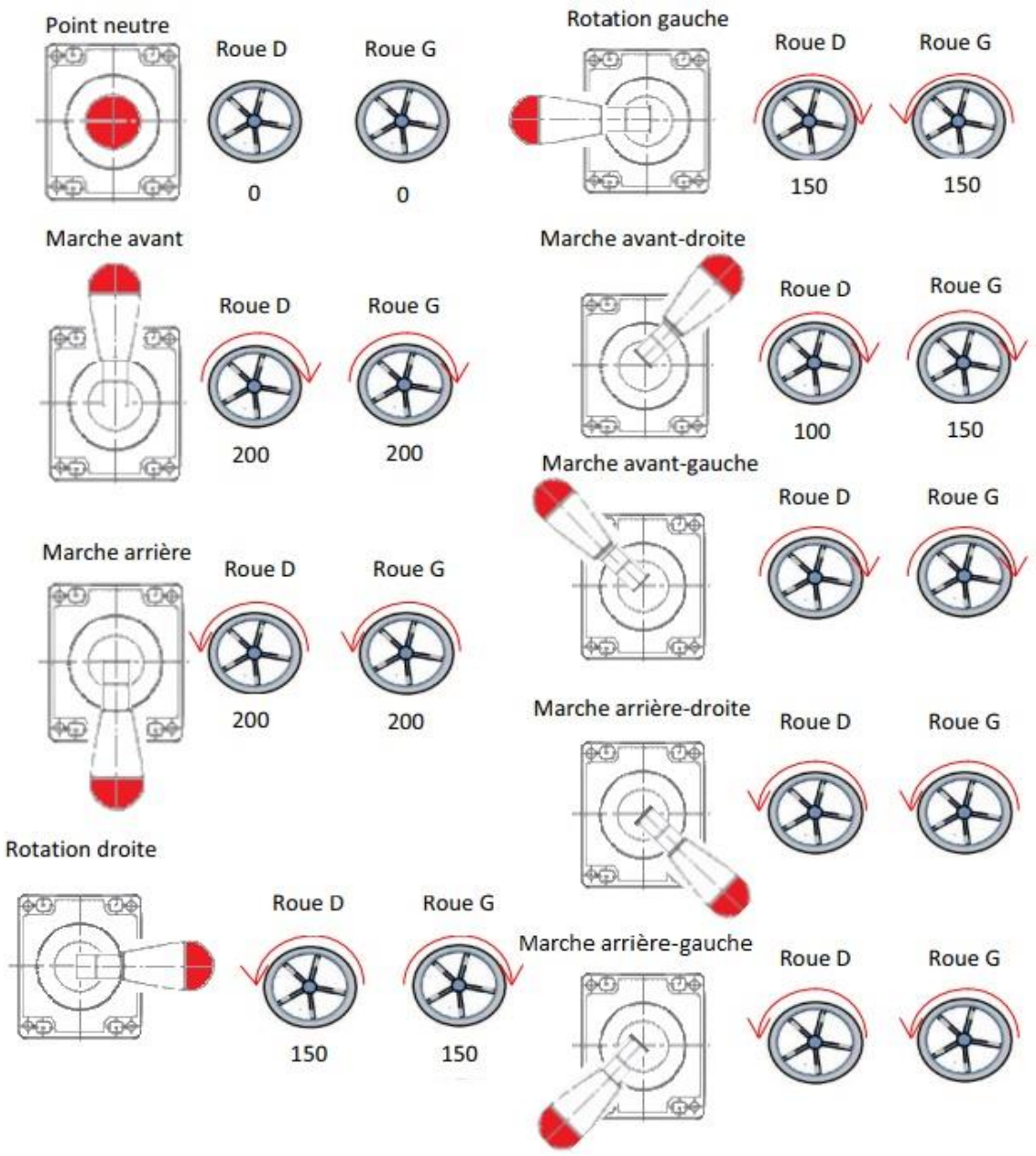



*Figure 7 Loi de commande des roues*

Comme expliqué précédemment, l'idée lorsque nous tournons dans un sens ou dans l'autre est de diminuer la vitesse de rotation de la roue intérieure au virage et d'accélérer celle extérieure au virage. Ce différentiel de vitesse entre les deux roues dépend en revanche de la façon dont l'utilisateur veut tourner. Plus le joystick sera incliné et éloigné de l'axe Y, plus le différentiel sera important, et inversement. Pour ce qui est de la vitesse de rotation des roues pour un déplacement linéaire, nous regarderons sa distance vis-à-vis de l'axe X. Plus le joystick sera éloigné de l'axe X sur l'intervalle Y [0 ; $Y_{max}$], plus le fauteuil ira vite en avant et plus le joystick sera éloigné de l'axe X sur l'intervalle Y [0 ; $-Y_{max}$], plus le fauteuil ira vite en arrière.

## Modules de puissance MD03

Le module MD03 est une commande de puissance pour moteurs DC, ce n'est qu'un hacheur 4 quadrants déjà équipé pouvant recevoir des trames I²C. Ce module nous a été imposé au début du projet. La tension moyenne aux bornes du moteur est régulée par un signal MLI qui viendra bloquer ou saturer des transistors admettant une tension tantôt positive tantôt négative en fonction de l'envie de pilotage.

Nous disposons de 2 modules identiques, un pour chaque moteur. Voici leurs caractéristiques :

- Tension d'entrée 50V DC max
- Tension de sortie 24V DC max
- Courant de sortie 20A max

Les modes de commandes des modules MD03 :

- I²C
- 0V-2.5V-5V entrée analogique
- 0V-5V entrée analogique
- Mode radio contrôle
- MLI

Seul la commande par envoie de trame sous protocole I²C nous intéresse. Ce protocole de commande est le plus intéressant puisqu'il permet de commander jusqu'à 8 modules avec seulement 4 fils (SDA, SCL, alim et GND). Il est simple à mettre en place et nous pouvons facilement changer les adresses des modules grâce à des interrupteurs déjà implémentés sur la carte. L'attribution de ces adresses et le protocole de communication se feront ensuite de manière logicielle avec le programme en C qui sera directement téléchargé dans le PIC.

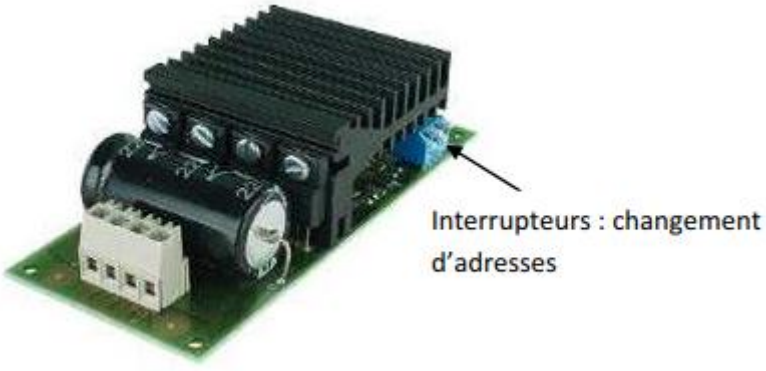



*Figure 8 Module MD03*

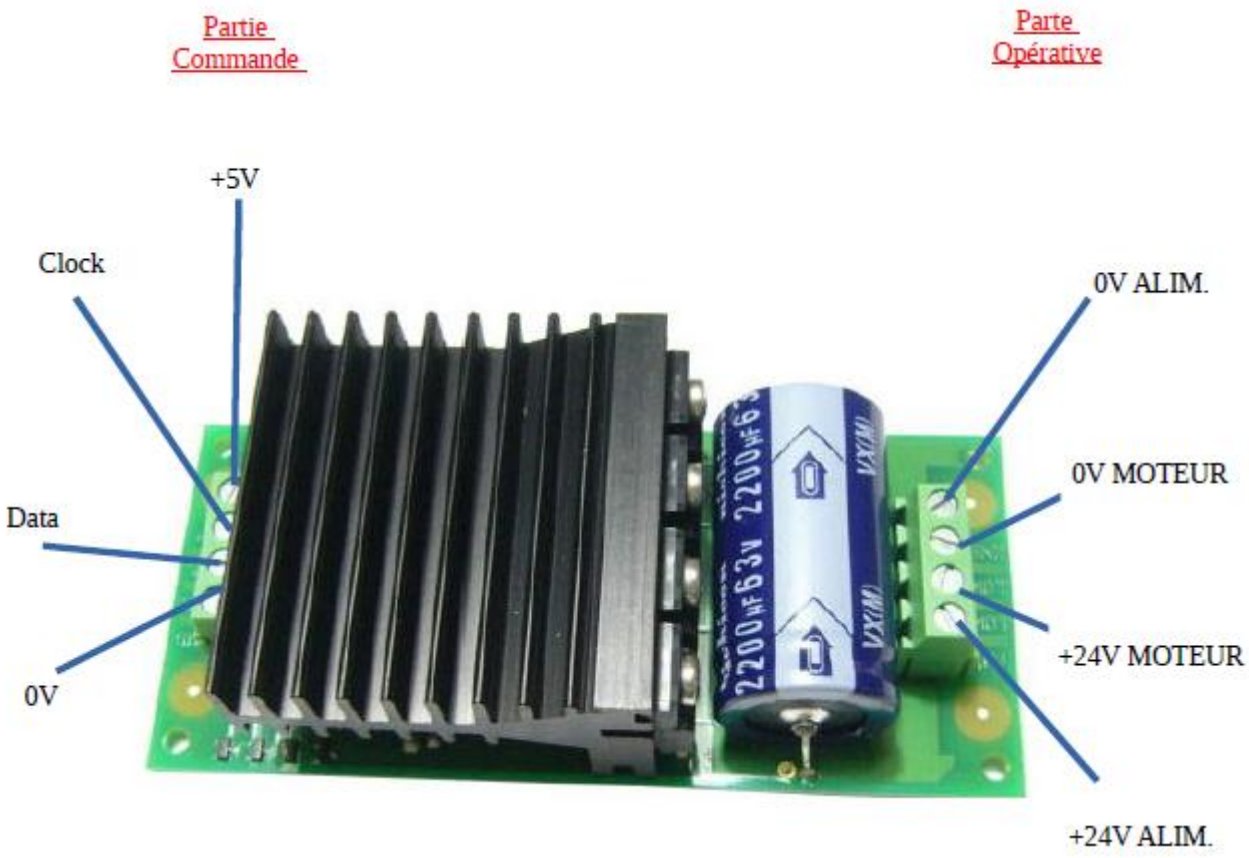



*Figure 9 Module MD03 description*

Les modules doivent être configuré avec deux adresses différentes pour éviter les conflits. La configuration des adresses se fait via les interrupteurs situés derrière les modules (figure 8).

| Mode | Switch 1 | Switch 2 | Switch 3 | Switch 4 |
|---|---|---|---|---|
| I2C Bus - address 0xB0 | On | On | On | On |
| I2C Bus - address 0xB2 | Off | On | On | On |
| I2C Bus - address 0xB4 | On | Off | On | On |
| I2C Bus - address 0xB6 | Off | Off | On | On |
| I2C Bus - address 0xB8 | On | On | Off | On |
| I2C Bus - address 0xBA | Off | On | Off | On |
| I2C Bus - address 0xBC | On | Off | Off | On |
| I2C Bus - address 0xBE | Off | Off | Off | On |

*Figure 10 Tableau d'attribution des adresses modules MD03*

La modification du sens, de la vitesse et de l'accélération se fait dans des registres. Chaque registre a des adresses et contient des variables qui vont commander ces paramètres. Seuls 3 registres nous intéressent dans notre cas :

- Le registre « Command » à l'adresse 0x00 qui commande le sens de rotation des moteurs (avant/arrière).
- Le registre « Speed » à l'adresse 0x02 qui commande la vitesse de rotation des moteurs.
- Le registre « Acceleration » à l'adresse 0x03 qui commande l'accélération des moteurs.

| Register Address | Name | Read/Write | Description |
|---|---|---|---|
| 0 | Command | R/W | Write 01 Forwards - 02 Reverse (00 for instant stop - Rev9 firmware only) |
| 1 | Status | Read only | Acceleration, Temperature and Current Status |
| 2 | Speed | R/W | Motor Speed 0-255 (0x00 - 0xFF) |
| 3 | Acceleration | R/W | Motor Acceleration 0-255 (0x00 - 0xFF) |
| 4 | Temperature | Read only | Module temperature |
| 5 | Motor Current | Read only | Motor Current |
| 6 | Unused | Read only | Read as zero |
| 7 | Software Revision | Read only | Software Revision Number- Currently 9 |

*Figure 11 Registre et adresse des modules MD03*

## Liaison RS232

Cette liaison RS232 permet de programmer le microcontrôleur PIC. La carte développée n'a pas été prévu pour recevoir un programmateur Pickit 3 et nous ne programmons pas sous MPLab. Cette liaison est réalisée grâce au MAX232. Nous utilisons 2 fils pour les données, le RX pour la réception des données et le TX pour la transmission des données. La connexion au PC est faite via un connecteur DB9.

*Figure 12 Liaison RS232 via MX232*

## Motorisation du fauteuil

La motorisation du fauteuil est constituée de 2 moteurs à courants continu à excitation série avec réducteur à renvoi d'angle. Des freins à manque de courant sont directement montés sur les moteurs. Comme expliqué précédemment, cela rajoute une dimension sécuritaire au fauteuil en cas de défaut d'alimentation. Pour la commande des freins, nous avons choisi de moduler la tension d'alimentation de ces freins par commande MLI afin de permettre une économie d'énergie de la batterie.

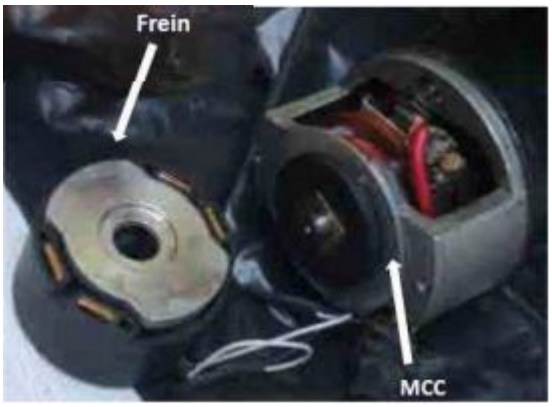



*Figure 13 Frein de la MC*

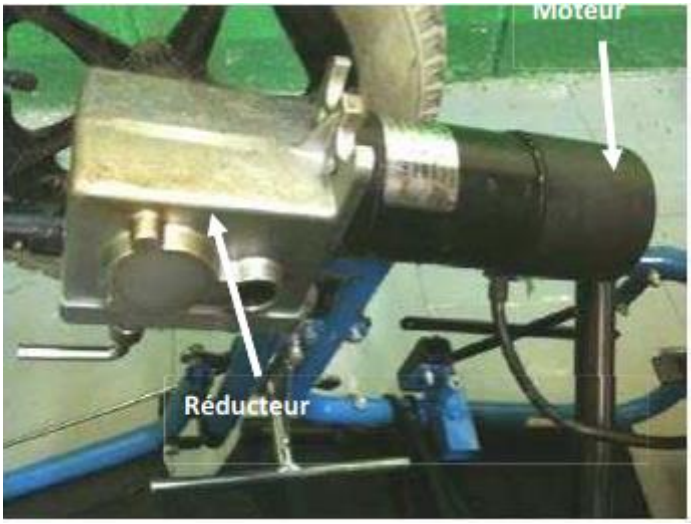



*Figure 14 Moteur et réducteur=*

Voici les caractéristiques des moteurs DC :

- Alimentation des moteurs en 24V DC
- Tension de libération des freins U>17V
- Tension de serrage des freins U<5V
- $N_n$ = 110tr/min
- $I_n$ = 3.3A

## Utilisation de deux gammes de vitesse

Nous avons imaginé de mettre en place un système de 2 gammes de vitesse sur le fauteuil de manière à avoir une correction du différentiel de vitesse entre les deux roues lorsque nous sommes à petite vitesse en train de faire des manœuvres serrées ou à grande vitesse en train de rouler à vitesse de nominale. La première idée qui nous est venue était à l'aide d'un interrupteur quelconque (Switch, BP, etc.) de pouvoir sélectionner notre mode de fonctionnement.

Dans le mode de fonctionnement basse vitesse, même en poussant le stick du joystick à ce maximum, nous sommes bridés à 50% de la vitesse max mais pas de l'accélération. Nous encourageons les mouvements de rotation serrés. Il est donc possible de faire une rotation à 360° sur soi-même en poussant le joystick à son maximum à gauche ou à droite ou de faire des virages très serrés en avançant ou en reculant.

Dans le mode de fonctionnement grande vitesse, il ne sera plus possible de faire ce genre de manœuvre. En effet, puisque nous roulons plus vite, pour sécuriser le fauteuil, nous nous interdisons une manœuvre subite à 360° sur soi-même alors que nous sommes potentiellement à notre vitesse de déplacement nominal. Cela aurait pour conséquence un déséquilibre instantané du fauteuil qui pourrait s'ensuivre par un basculement de celui-ci.

De manière logicielle, il suffirait de mettre 2 conditions dans le programme principal. Si le mode grande vitesse est enclenché, le programme exécute la loi de commande déjà existante, sinon la vitesse max est diminuée de 50% mais le différentiel de vitesse entre les deux roues sera plus important.

Nous n'avons pas réussi à mettre ce système en place par manque de temps.

## Autonomie et puissance

Pour l'instant, durant tous les tests réalisés, la source d'énergie du fauteuil était une alimentation de puissance de laboratoire. Il existe avec ce projet deux batteries de 12V chacune qui pourraient être réhabilités pour le fauteuil. Il faudrait les recharger et effectuer des tests de débit de courant de puissance. Nous avons déjà observé que lorsque les deux roues du fauteuil tournent les 2 dans le même sens à vitesse maximum, les moteurs consomment environ 3A.

## Odométrie

En parallèle du travail réalisé sur le pilotage du fauteuil, une autre tâche nous a été confiée quant à la mesure de la vitesse de celui-ci. Le travail demandé était via un petit moteur pas-à-pas, de retrouver l'information de vitesse du fauteuil. Dans l'idée, il y aurait 2 moteurs pas-à-pas mécaniquement liés à chaque roue du fauteuil. L'avantage serait d'avoir une estimation de la distance parcourue par le fauteuil. Avec nos batteries nous pourrions estimer la distance moyenne que peut parcourir le fauteuil, l'utilisateur ayant en direct l'information de la distance parcourue depuis son départ saurait s'il se rapproche du seuil critique des batteries. Dans notre cas l'information de la vitesse serait plutôt accessoire.

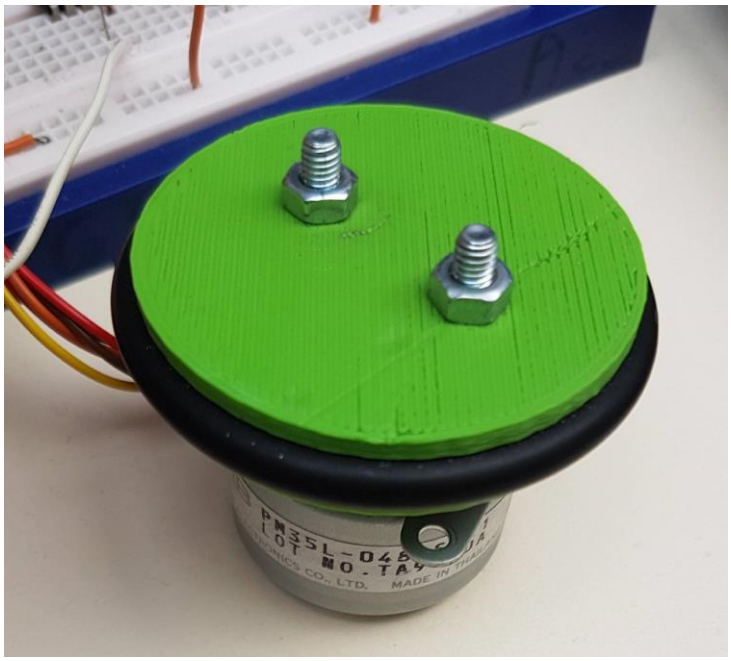


*Figure 15 Moteur pas-à-pas*

En récupérant la tension générée à chaque fois que le rotor passe un pas, nous pouvons la mettre en forme avec un montage électronique et la traiter via un pic. En connaissant le diamètre de la roue du moteur, nous pouvons calculer son périmètre. C'est un moteur 48 pas, à chaque fois que nous comptons 48 interruptions de fronts montant nous savons que nous avons fait un tour. Connaissant le périmètre nous multiplions le nombre de tours par celui-ci et avons la distance parcourue.

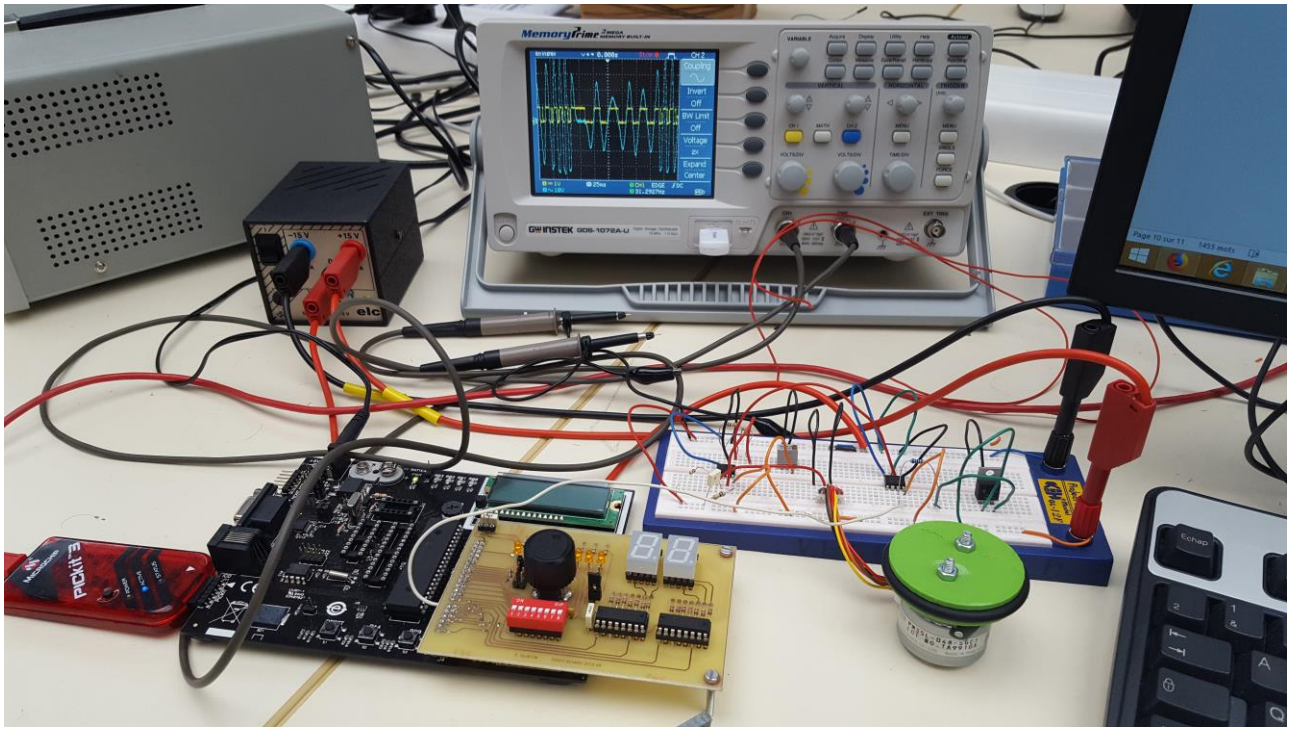


*Figure 16 montage moteur pas-à-pas*

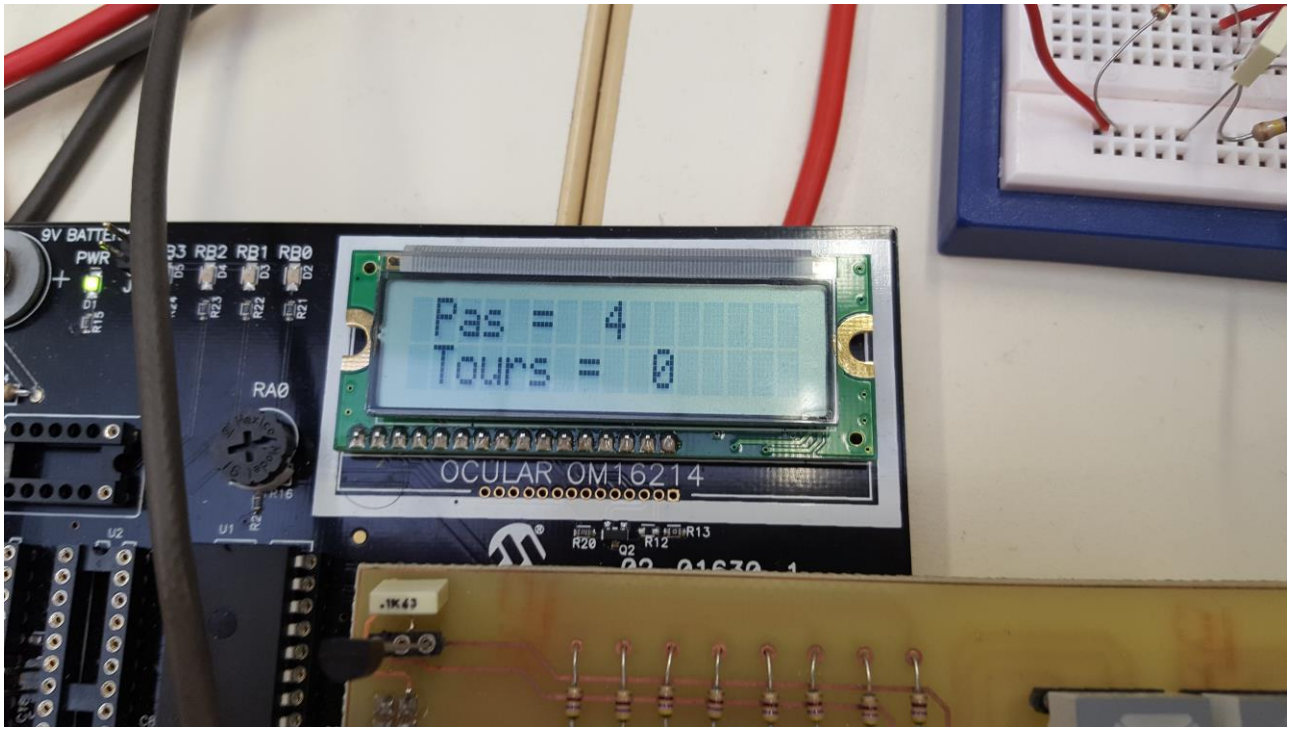



*Figure 17 Comptage des pas et des tours*

## Conclusion

Ce projet s'étant déroulé sur une année entière nous a permis de mettre en pratique nos connaissances théoriques acquises durant notre formation en Licence VEGA et des années précédentes.

Nous avons dû faire un gros travail de réflexion pour le pilotage du fauteuil avec les différentes lois de commandes envisageables.

Ayant déjà tous les deux mené des projets au cours de nos formations précédentes, nous avons pu nous organiser plus facilement dans notre façon de travailler et de nous répartir les tâches.

# Annexes

```
#include <16F877.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#use i2c(MASTER, sda=PIN_C4, scl=PIN_C3)  // utilisation bus I2C
#include <MD03_2014.c>

////////////////////////////////////////////////////
// Acquisition PORT analogique et acquisition joystick//
////////////////////////////////////////////////////

//valeur milieu=128
//valeurs max haut=217     //valeur max gauche=40
//valeurs max bas=43       //valeur max droit=215


int Y=0, haut=0, X=0, droite=0, fenetre=5, tolerance=10, sensG=0, sensD=0, vitG=0, vitD=0;
int accG=0, accD=0;

void init_joystic()
   {
       //init du PORTA analogique
       setup_adc(ADC_CLOCK_INTERNAL);   //utilisation de l'horloge interne
       setup_port_a( ALL_ANALOG );      //tout le PORTA en entrée analogique

       set_adc_channel( 0 );            //Choix "voie ANALOGIQUE"
       delay_us(10);                    //Délai stabilisation avant acquisition
       Y=read_adc();                    //Acquisition capteur
       set_adc_channel( 1 );            //Choix "voie ANALOGIQUE"
       delay_us(10);                    //Délai stabilisation avant acquisition
       haut=read_adc();
       set_adc_channel( 2 );            //Choix "voie ANALOGIQUE"
       delay_us(10);                    //Délai stabilisation avant acquisition
       X=read_adc();
       set_adc_channel( 3 );            //Choix "voie ANALOGIQUE"
       delay_us(10);                    //Délai stabilisation avant acquisition
       droite=read_adc();

       //écriture bus i2c
       Ecriture_acc_MD03(0,accG);       //registre MD03 motG 0xB0
       Ecriture_acc_MD03(2,accD);       //registre MD03 motD 0xB2
   }
```

```
void fonctionnement()
   {
      //Acquisition du quadrant 1 (+/+)
      if((Y>(128+fenetre=)&&(X>(128+fenetre)))    //condition pour mesure du quadrant 1 -> virage avant à droite progressif
      {vitG=Y-128; vitD=Y-vitG; sensG=1; sensD=1} //on tourne à droite -> on diminue la vitesse de rotation droite

      //Acquisition du quadrant 2 (-/+)
      if((Y>(128+fenetre))&&(X<(128-fenetre)))    //condition pour mesure du quadrant 2 -> virage avant à gauche progressif
      {vitG=vitD-Y; vitD=Y-128; sansG=1; sensD=1;}//on tourne à gauche -> on diminue la vitesse de rotation gauche

      //Acquisition du quadrant 3(-/-)
      if((Y<(128+fenetre))&&(X<(128-fenetre)))    //condition pour mesure quadrant 3 -> virage arrière gauche progressif
      {vitG=vitD-Y; vitD=Y-128; sensG=2; sensD=2;} //on tourne à gauche -> on diminue la vitesse de rotation à gauche

      //Acquisition du quadrant 4 (+/-)
      if((Y<(128+fenetre))&&(X>(128+fenetre)))   //condition pour mesure quadrant 4 -> virage arrière droit progressif
      {vitG=Y-128; vitD=vitG-Y; sensG=2; sensD=2;} // on tourne à droite -> on diminue la vitesse de rotation droite

      //Tolérance couloir Y+
      if((Y>(128+tolerance))&&(X<(128+tolerance))&&(X>(128-tolerance))) //Plage de valeur réfercée comme purement axeY+
      {vitG=Y-128; vitD=Y-128; sensG=1; sensD=1;}  //marche avant

      //Tolérance couloir Y-
      if((Y<(128+tolerance))&&(X<(128+tolerance))&&(X>(128-tolerance))) //Plage de valeur réfercée comme purement axeY-
      {vitG=Y-128; vitD=Y-128; sensG=2; sensD=2;}  //marche arrière

      //Tolérance couloir X+
      if((X>(128+tolerance))&&(Y<(128+tolerance))&&(Y>(128-tolerance))) //Plage de valeur réfercée comme purement axeX+
      {vitD=X-128; vitG=X-128; sensG=1; sensD=2;}  //virage droite serré

      //tolérance couloir X-
      if((X>(128+tolerance))&&(Y<(128+tolerance))&&(Y>(128-tolerance))) //Plage de valeur réfercée comme purement axeY+
      {vitD=X-128; vitG=X-128; sensG=2; sensD=1;}  //virage gaucge serré

      //POINT NEUTRE
      if ((Y>(128-fenetre))&&(Y<(128+fenetre))&&(X>(128-fenetre))&&(X<(128+fenetre)) //on reste dans la fenetre morte (+/- 5)
      {vitG=0; vitD=0; sensG=0; sensD=0;} //on ne bouge pas, point neutre

      //ECRITURE SUR MD03
      Ecriture_speed_MD03(0,sensG,vitG);   //registre MD03 motG 0xB0
      Ecriture_speed_MD03(2,sensD,vitG);   //registre MD03 motD 0xB2

      //AFFICHAGE SUR TERMINAL
      printf("\n\r axeY axeX VITG VITD SENSG SENSD %2U %2U %2U %2U %2U %2U",bas, gauche, vitG, vitD, sensG, sensD);
      printf("\n\r ACCD ACCG %2U %2U", accG, accG);
      delay_ms(10);
   }
```

```
//////////////////////////
// PROGRAMME PRINCIPAL//
//////////////////////////

void main()
   {
      while(TRUE) //appel des fonctions
      {
         init_joystic();
         fonctionnement();
      }
   }
```

# Tables des illustrations