ROBOT GUIDE

PILOTAGE DU ROBOT PAR BLUETOOTH

Rapport de projet

GEII2 – Etude & Réalisation – Groupe D1

Etudiant : Lory-Stan CHAMSOUDINE

Professeurs encadrants : Éric LABOURE | Claire LALLEMAND

Remerciements

Tout d’abord, je remercie toutes les personnes qui ont contribué à l’avancement et au succès du projet.

Ensuite, je tiens à vivement remercier Monsieur Éric LABOURE, enseignant et chercheur à l’IUT de Cachan, grâce à qui j’ai pu effectuer mon projet d’E&R en prenant le temps de m’aider, de m’encadrer et de me guider tout au long du projet.

J’adresse aussi mes remerciements à Monsieur Maxime HOUBION pour m’avoir aidé lors des premières séances d’E&R et je lui souhaite de réussir dans sa nouvelle vie professionnelle.

Je remercie également ma professeure de Culture et Communication, Madame Claire LALLEMAND pour m’avoir conseillé durant l’intégralité du projet.

Enfin, je tiens à remercier les techniciens du bâtiment G, pour m’avoir fourni le matériel indispensable à la réalisation du projet et pour avoir imprimé mon circuit électrique.

Résumé

Au cours du troisième semestre à l'IUT de Cachan, j'ai choisi de travailler sur le robot guide qui interviendra durant les journées portes ouvertes.

Ce rapport vous présente les méthodes et les solutions, qui m'ont permis de mettre en œuvre le pilotage, par communication Bluetooth, du robot guide.

Pour réaliser ce projet, j'ai élaboré un circuit imprimé, qui permet de contrôler les moteurs et la direction du robot. De plus, ce circuit communique par Bluetooth avec mon smartphone, ce qui me permet de piloter le robot à distance. Afin de traiter les informations reçues par Bluetooth, j'ai également programmé un microcontrôleur Mbed en langage C++.

Mots-clés : Robot, Bluetooth, Programmation, Circuit imprimé, Microcontrôleur, Moteur « brushless », Vérin électrique, Informatique embarquée, Contrôle de trajectoire

Sommaire

Introduction

Au cours du troisième semestre à l'IUT de Cachan en filière Génie Électrique et Informatique Industrielle, j'ai eu l'opportunité de réaliser la partie déplacement du robot guide, au sein du cours d'Étude & Réalisation.

Ce projet m'a permis d'acquérir de nouvelles connaissances qui m'apporteront beaucoup pour la vie professionnelle.

Ce robot guide aura pour but de faire visiter les locaux de l'IUT de Cachan lors des journées portes ouvertes, de façon totalement autonome. En effet, il doit pouvoir identifier les obstacles face à lui et réagir en conséquence.

Nous nous intéresserons tout d'abord à la présentation du projet avant de se focaliser sur sa réalisation.

1 Présentation du projet

1.1 Les objectifs du projet

L'objectif principal de ce projet est de pouvoir piloter le robot guide depuis notre smartphone, par communication Bluetooth.

Il faudra donc concevoir une carte électronique, permettant de régler la vitesse et la direction de déplacement.

De plus, il sera nécessaire de programmer le microcontrôleur assurant la communication Bluetooth et le réglage des moteurs du robot.

1.2 Présentation des composants

Afin de piloter le robot, j'ai eu besoin de plusieurs composants.

Le microcontrôleur Mbed NXP LPC1768 comporte 40 broches dont certaines peuvent envoyer et recevoir des signaux tels que des signaux PWM ou encore des signaux par liaison série.

Il fera la liaison entre tous les composants du robot et va donc permettre de commander celui-ci.

Figure 1 - Mbed NXP LPC1768

Le robot est piloté à l'aide d'une communication Bluetooth. La mise en place d'une connexion Bluetooth s'effectue avec un module.

Le module HC-05 est un dispositif permettant de communiquer par Bluetooth avec son smartphone. Il peut également communiquer avec le microcontrôleur, à l'aide d'une liaison série.

Figure 2 - Module Bluetooth HC-05

Figure 3 - Trottinette électrique

Le robot est constitué de deux trottinettes électriques Airwheel Z3. Ceux-ci ont chacune une batterie de 41 V et un moteur brushless triphasé pour les faire avancer.

Ces moteurs sont, entre autres, utilisés dans la plupart des motos, scooters et voitures électriques, mais aussi dans les roues de certains modèles de trottinettes électriques (telles qu'étudiées dans mon projet).

Un moteur brushless est un type de moteur synchrone dont le rotor est constitué d'un ou de plusieurs aimants permanents et dont le stator est constitué de plusieurs bobinages. Ces bobines sont alimentées de façon séquentielle, créant un champ magnétique et donc faisant tourner les moteurs.

Afin de réaliser cette séquence permettant de faire tourner les roues, un onduleur a été requis.

Le BOOSTXL-DRV8305EVM a été celui qui correspondait au mieux au résultat attendu. Constitué de 40 broches, cet onduleur permet d'alimenter très facilement des moteurs triphasés.

Figure 4 - Onduleur BOOSTXL-DRV8305EVM

Figure 5 - Vérin électrique

Afin de diriger le robot, un vérin électrique a été placé sur le robot pour le faire tourner.

Le vérin électrique peut s'allonger ou se rétrécir en fonction de la tension envoyée (positive ou négative).

Afin de contrôler le vérin électrique, il est nécessaire d'avoir un hacheur qui permette d'envoyer une tension positive ou négative en fonction de ce que l'on souhaite.

Le hacheur VNH5019 est constitué de 10 broches de commande destinée au microcontrôleur et de 4 broches de puissance destinée au vérin.

Figure 6 - Hacheur VNH5019

1.3 Schéma fonctionnel

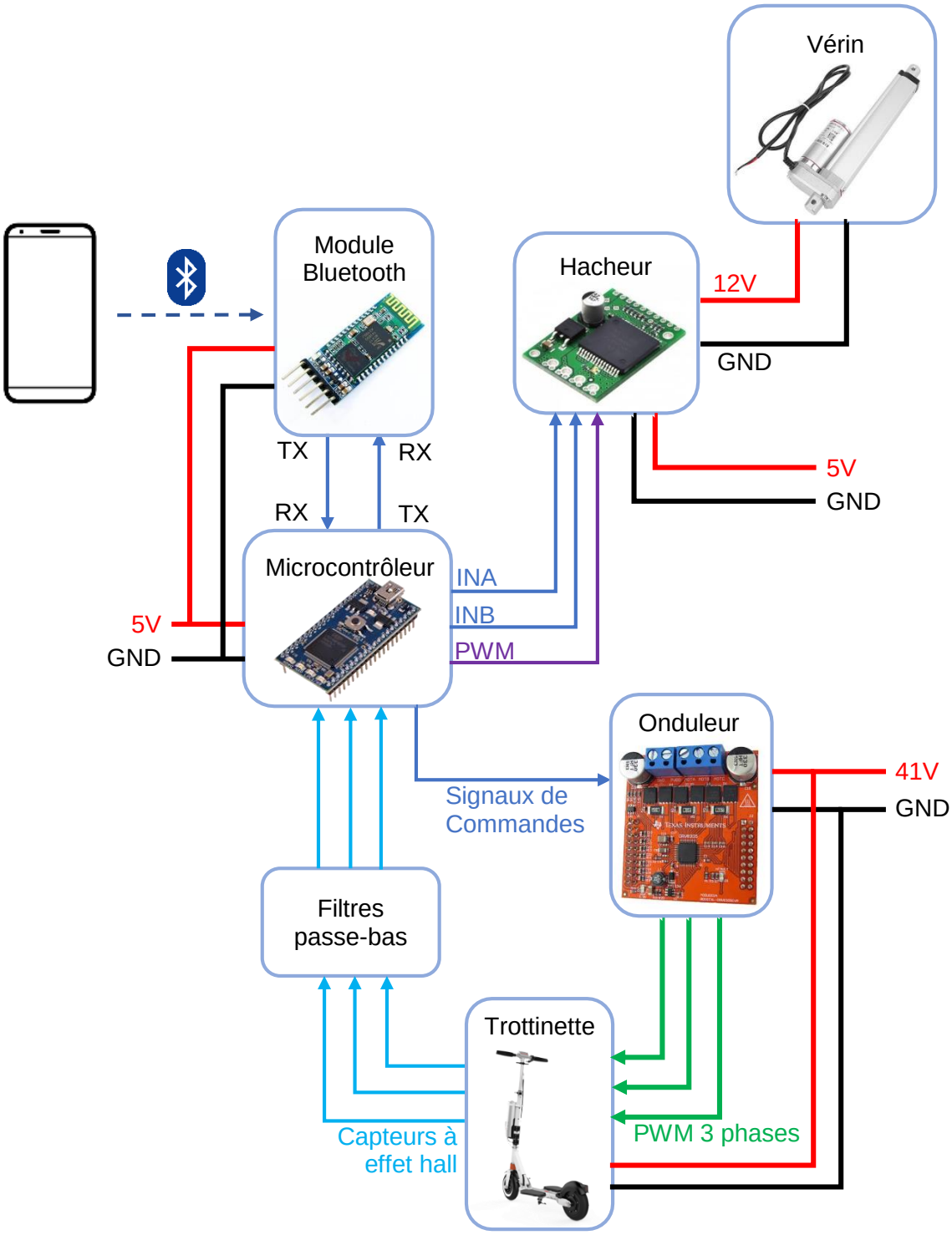

Figure 7 - Schéma fonctionnel du robot guide

1.4 Présentation des logiciels

Afin de réaliser ce projet, plusieurs logiciels m'ont été nécessaires :

Tout d'abord, et comme à chaque projet d'E&R, je me suis servi du logiciel Altium Designer. Celui-ci permet de concevoir des cartes électroniques puis les imprimer.

Ensuite, j'ai utilisé le site internet Mbed. Celui-ci permet d'écrire un programme en langage C++ et à l'aide d'un simple câble USB, de l'envoyer sur le microcontrôleur.

Enfin, afin de communiquer avec le robot par communication Bluetooth, j'ai également eu recours à l'application « Bluetooth Electronics », disponible sur le Google Play Store. Cette application permet d'envoyer des signaux par Bluetooth de façon très intuitive.

1.5 Planning prévisionnel

Afin de planifier mon projet, j'ai décidé de réaliser un diagramme de Gantt pour toute la durée du projet.

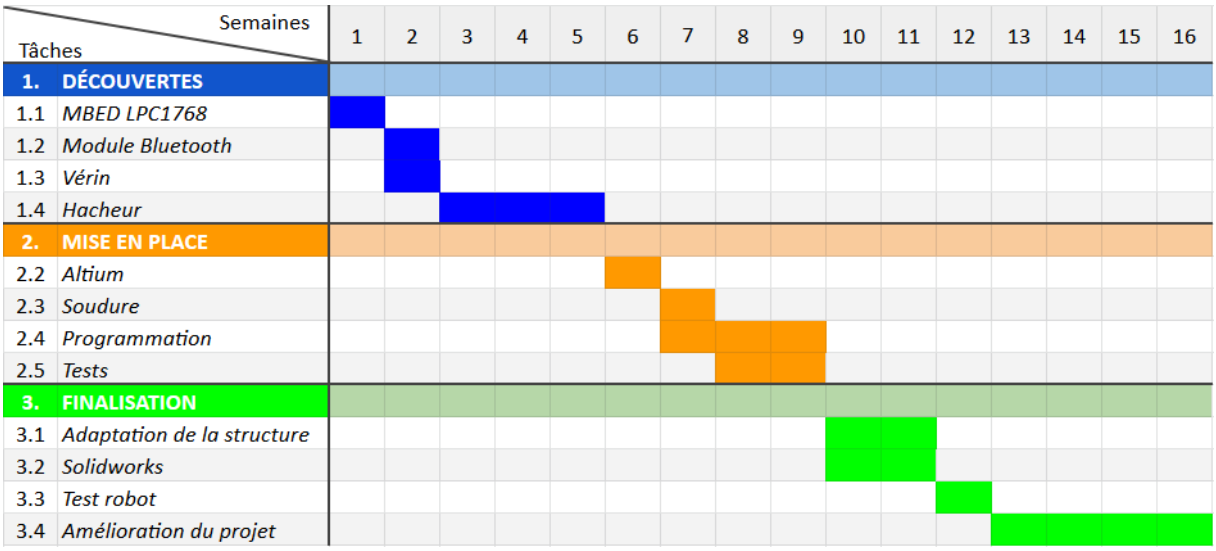

Figure 8 - Diagramme de Gantt

2 Réalisation du projet

2.1 Mise en place

2.1.1 Communication Bluetooth

Afin de piloter le robot à distance, je mets en lien un module Bluetooth situé sur la carte électronique du robot et mon smartphone.

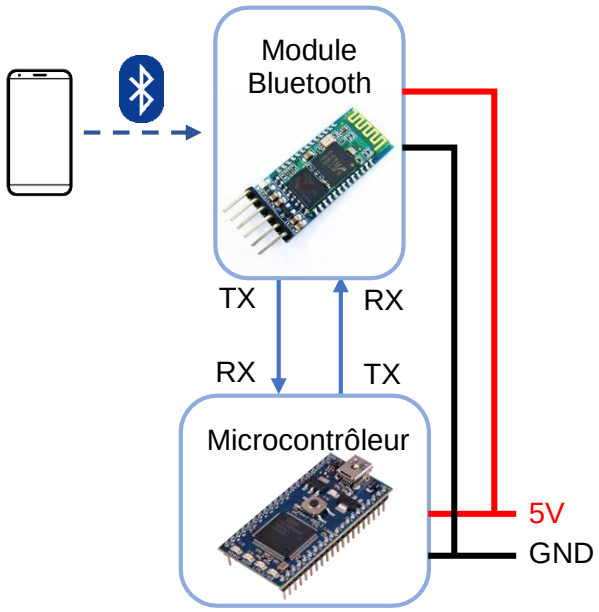

Figure 9 - Schéma connexion Bluetooth

Le module Bluetooth utilisé est le module HC-05 qui est alimenté en 5 V, tout comme le microcontrôleur.

Chacun des deux composants possède une broche RX et une broche TX. Ceux-ci sont des ports séries, c'est-à-dire qu'ils envoient des données successives bit après bit, séquentiellement. En effet, la broche TX permet de transmettre des informations tandis que la broche RX permet d'en recevoir.

Afin de comprendre le fonctionnement du module Bluetooth, j'ai tout d'abord réalisé un test sur une plaque labdec. De plus, pour communiquer avec le module Bluetooth, il a fallu installer, sur smartphone, l'application « Bluetooth Electronics », disponible sur le Google Play Store.

Pour ce test, j'ai simplement envoyé des caractères, pour ensuite les afficher sur l'HyperTerminal de l'ordinateur. Vous trouverez les lignes de code permettant de lire et afficher ces caractères :

```
// Affectation des liaisons séries
Serial bluetooth(p13,p14);
Serial PC(USBTX,USBRX);

int main() {
    // Déclaration des variables
    char c = '0';

    // Initialisation des liaisons séries
    bluetooth.baud(9600);
    PC.baud(9600);

    while (1){

        // Si on reçoit un caractère
        if(bluetooth.readable() == 1) {
            c = bluetooth.getc(); // On récupère le caractère
            PC.printf("%c",c); // On l'affiche sur l'hyperterminal
        }
    }
    return 0;
}
```

Figure 10 - Code pour la connexion Bluetooth

2.1.2 Contrôle de la direction

Après avoir finalisé le fonctionnement du module Bluetooth, j'ai décidé de m'occuper du vérin servant à diriger le robot.

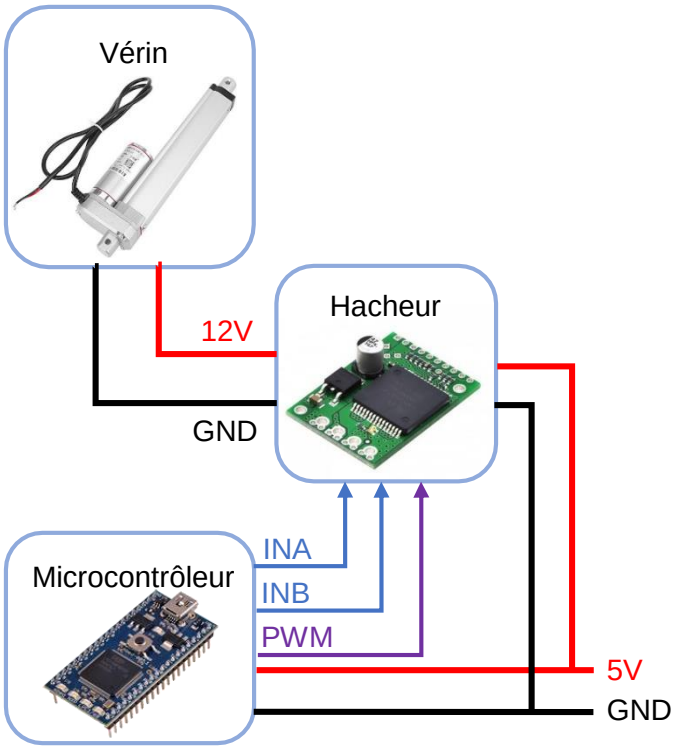

Figure 11 - Schéma contrôle de la direction

Le vérin est alimenté par une tension de 12 V. En amont du vérin, on trouve un hacheur qui permet de convertir une tension continue réglable en une tension continue fixe. Le hacheur quant à lui est alimenté par une tension de 5 V.

Pour faire fonctionner le hacheur et donc le vérin, il nous faut trois signaux : deux signaux digitaux tout ou rien pour indiquer le sens du vérin s'il doit sortir ou entrer et un signal PWM qui permet de régler la vitesse du vérin.

Avec toutes ces informations, j'ai réalisé un premier test du vérin. J'envoie les caractères « 4 » et « 2 » par Bluetooth à l'aide de mon smartphone. Le vérin s'allongera en recevant le caractère « 4 » et se rétrécira en recevant le caractère « 2 ».

```
// Affectation des sorties digitales
DigitalOut PhD(p25);
DigitalOut PhG(p24);

// Affectation des sorties PWM
PwmOut MV(p26);
        switch(c){
            case '4': // Gauche
                MV.write(0.3);
                PhD.write(0);
                PhG.write(1);
                break;

            case '2': // Droite
                MV.write(0.3);
                PhD.write(1);
                PhG.write(0);
                break;

            default: // Si aucun des cas au dessus
                MV.write(0);
                PhD.write(0);
                PhG.write(0);
                break;
        }
```

Figure 12 - Code pour la direction du robot

2.1.3 Contrôle des moteurs

Afin de faire avancer et reculer le robot, il va falloir contrôler les moteurs des trottinettes. Les moteurs brushless sont des moteurs qui fonctionnent avec des aimants. Dans notre cas, chaque moteur est constitué de trois aimants.

Le principe de fonctionnement de ce type de moteur est assez simple. Il suffit d'alimenter chaque aimant en réalisant une séquence définie. En effet, en alimentant le premier aimant, le moteur va réaliser une rotation précise, ainsi en alimentant l'aimant suivant, celui-ci va réaliser une nouvelle rotation et ainsi de suite. De ce fait, en alimentant séquentiellement chacun des aimants, le moteur va réussir à tourner.

Il me fallait donc déterminer cette séquence en réalisant plusieurs tests.

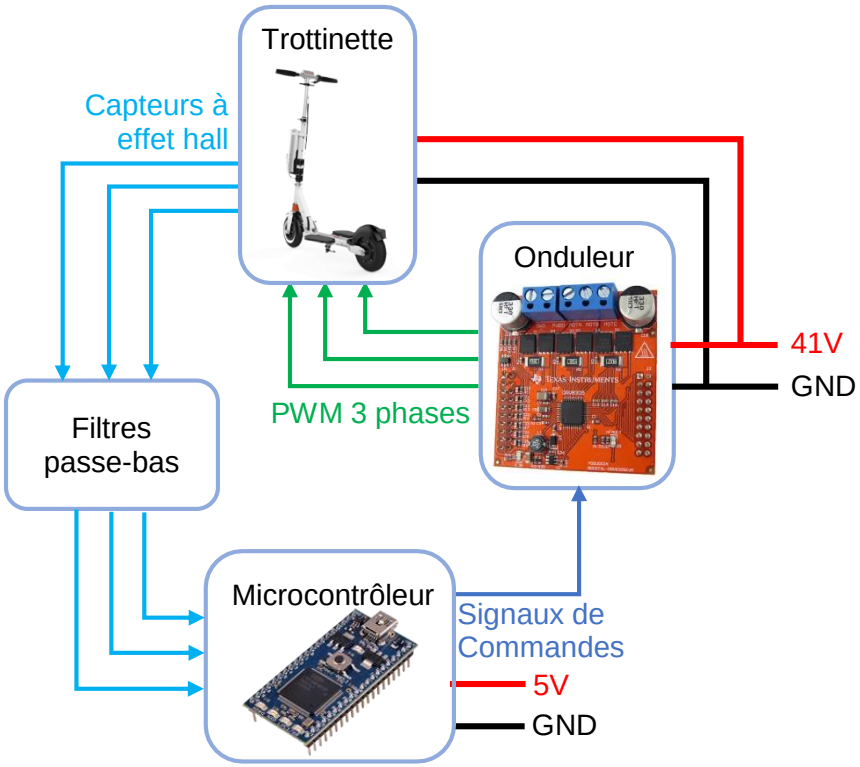

Figure 13 - Schéma contrôle des moteurs

L'onduleur BOOSTXL-DRV8305EVM et les moteurs de la trottinette sont alimentés en 41 V qui est directement fourni depuis la batterie de la trottinette. L'onduleur envoie trois signaux PWM au moteur de la trottinette, ceux-ci contenant la séquence permettant de faire tourner le moteur.

Lors de la lecture de ces capteurs, il a fallu réaliser un filtre passe-bas dû au fait que les valeurs mesurées possédaient énormément de perturbations, il m'était donc impossible de lire ces valeurs directement depuis les capteurs.

En utilisant la table de vérité de l'onduleur fournie dans la fiche technique (voir annexe), on applique tous les codes que peut envoyer l'onduleur au moteur et nous relevons les valeurs de chaque capteur à effet hall, ainsi nous observons que le moteur est à l'arrêt lorsque ces codes sont appliqués.

À la suite de cela, j'obtiens :

Capteur Hall			PWM pour marche :		
Vert	Bleu	Jaune	Arrêt	Avant	Arrière
0	0	1	1100	1010	0100
0	1	0	1000	1100	0010
0	1	1	0110	1000	0110
1	0	0	0010	0110	1000
1	0	1	0100	0010	1100
1	1	0	1010	0100	1010

Figure 14 - Table de vérité pour la mise en marche du moteur

À l'aide de la table obtenue lorsque le moteur est à l'arrêt, j'en déduis que pour faire avancer ce moteur, il faut décaler la table, tandis que pour le faire reculer, il faut lire cette table dans l'ordre inverse.

Ainsi, pour contrôler les moteurs, il faut appliquer un certain code binaire à l'onduleur en fonction des valeurs récupérées depuis les capteurs à effet hall.

```
void Moteur(float vitesse) {

    // Si c'est en marche avant
    if (vitesse > 0) {
        Pwm_INHA.write(vitesse);

        switch (Lecture_CapteursHall()) {
            case 0b001:
                Ecriture_PWM(0b1010);
                break;
            case 0b010:
                Ecriture_PWM(0b1100);
                break;
            case 0b011:
                Ecriture_PWM(0b1000);
                break;
            case 0b100:
                Ecriture_PWM(0b0110);
                break;
            case 0b101:
                Ecriture_PWM(0b0010);
                break;
            case 0b110:
                Ecriture_PWM(0b0100);
                break;
        }
    }

    // Si c'est en marche arrière
    else if (vitesse < 0) {
        Pwm_INHA.write(vitesse*(-1));

        switch (Lecture_CapteursHall()) {
            case 0b001:
                Ecriture_PWM(0b0100);
                break;
            case 0b010:
                Ecriture_PWM(0b0010);
                break;
            case 0b011:
                Ecriture_PWM(0b0110);
                break;
            case 0b100:
                Ecriture_PWM(0b1000);
                break;
            case 0b101:
                Ecriture_PWM(0b1100);
                break;
            case 0b110:
                Ecriture_PWM(0b1010);
                break;
        }
    }
}

```

Figure 15 - Programme pour le contrôle des moteurs

2.2 Conception de la carte électronique

Après avoir compris le fonctionnement de chaque composant individuellement, j'ai pu commencer la réalisation de la carte électronique. Cette dernière a été réalisée à l'aide du logiciel « Altium Designer ».

2.2.1 Schéma de la carte

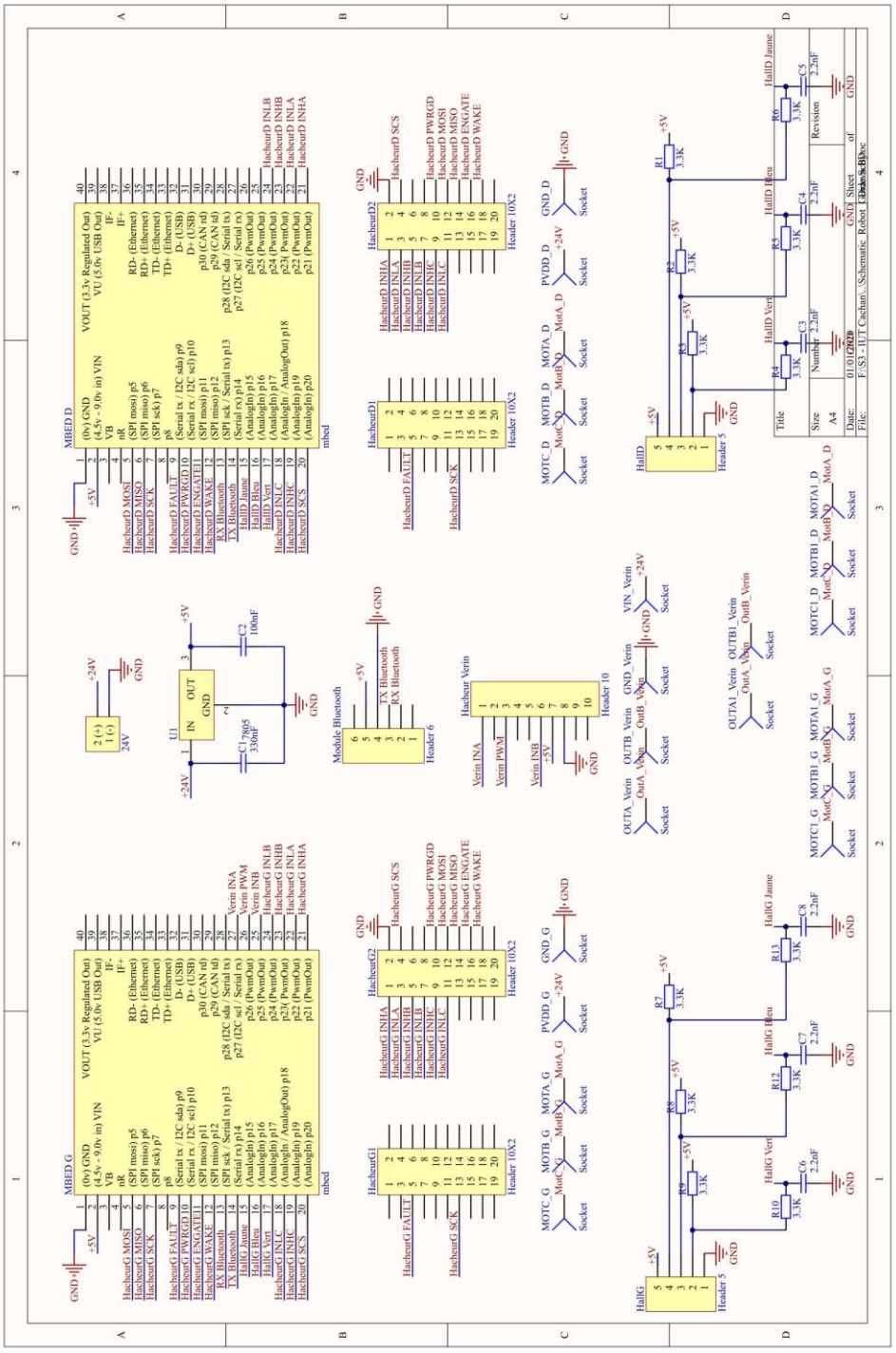

Figure 16 - Schéma électrique de la carte électronique

Ce schéma a été réalisé à partir des différents tests faits durant le projet. Étant donné que le robot est constitué de deux trottinettes et donc de deux moteurs, il a fallu doubler le nombre de composants permettant de gérer les moteurs. Vous pouvez donc remarquer que la carte possède deux microcontrôleurs, deux onduleurs et deux brochages pour la lecture de capteur à effet hall.

2.2.2 Routage de la carte

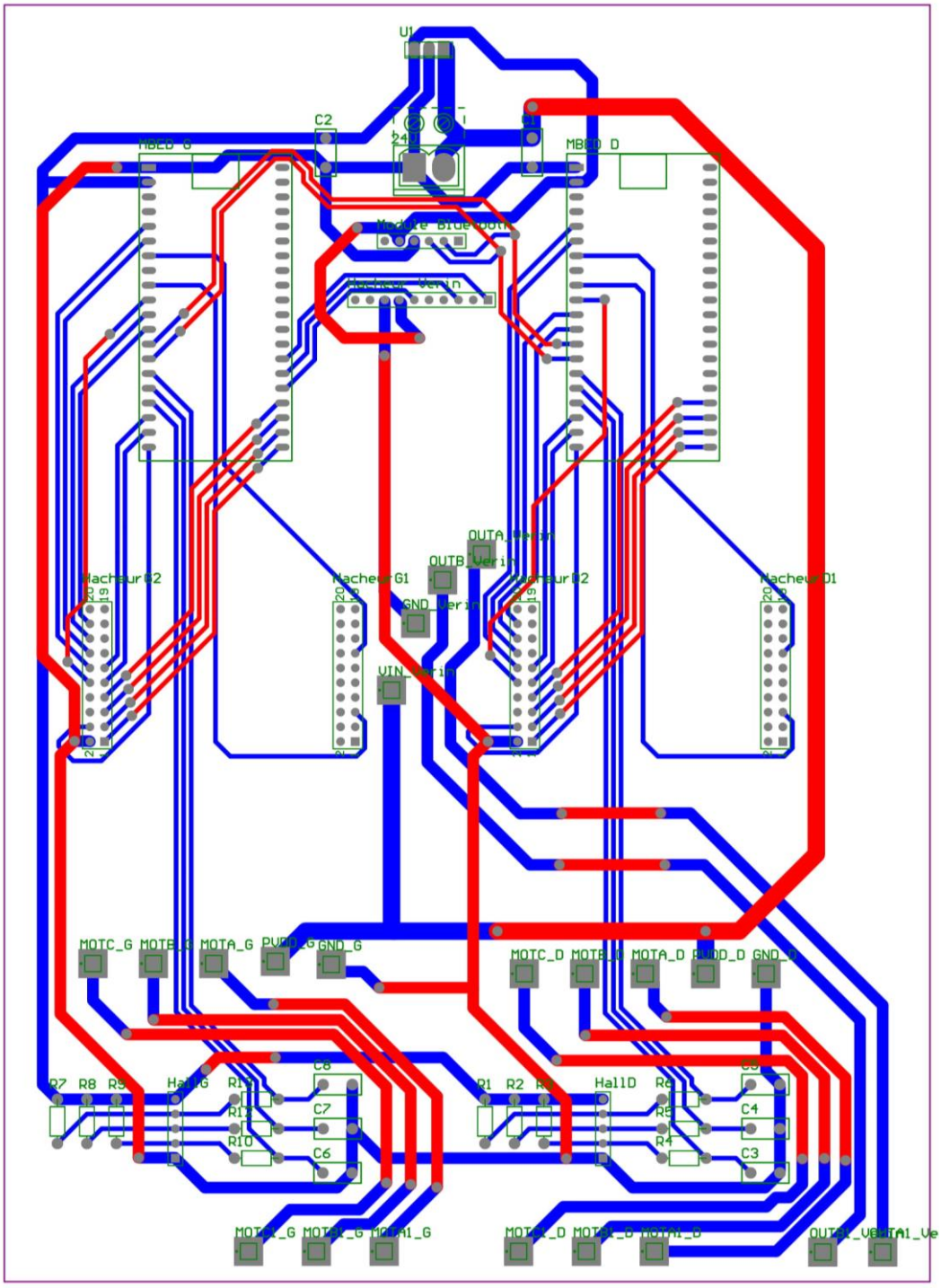

Figure 17 - Routage de la carte électronique

À l'achèvement du schéma, il m'a fallu router la carte toujours à l'aide du logiciel « Altium Designer ». Cela m'a pris un certain temps dû au nombre important de composants qui se trouve sur la carte.

2.2.3 Carte électronique

Enfin, lorsque le routage a été terminé, j'ai pu envoyer les documents de conception de la carte aux techniciens afin qu'il soit imprimé. Voici ma carte après avoir soudé tous les composants, elle mesure 16 cm de largeur et 22 cm de longueur.

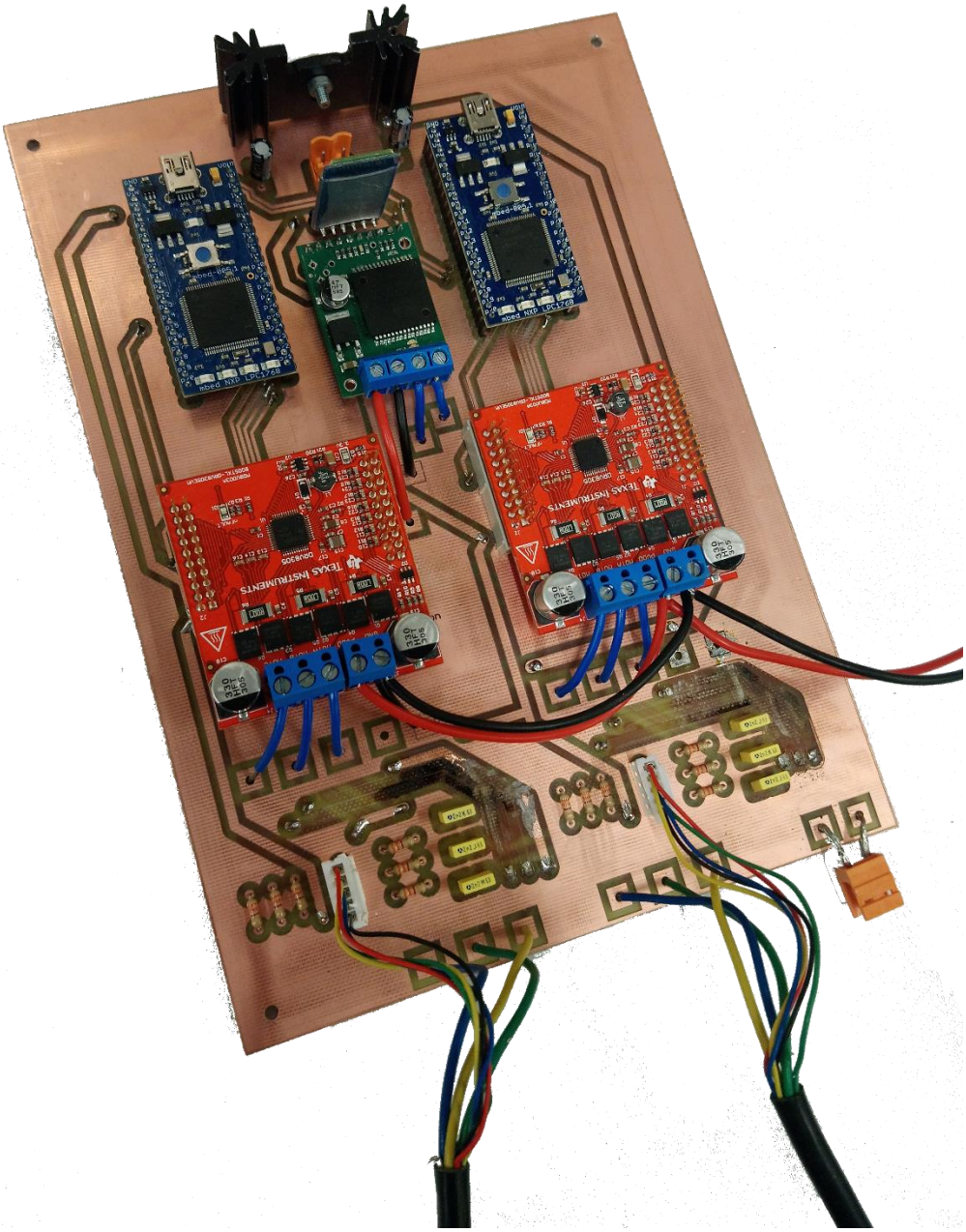

Figure 18 - Carte électronique

2.3 Les difficultés rencontrées et les solutions trouvées

Durant mon projet, j'ai dû faire face à quelques obstacles.

Dans un premier temps, afin d'alimenter le microcontrôleur et le hacheur, il me fallait une tension de 5 V, pourtant avec la batterie 41 V des trottinettes, il m'était impossible de la convertir en 5 V. J'ai donc utilisé une deuxième batterie délivrant 12 V, cette tension est alors convertie en 5 V pour alimenter les composants.

Et dans un second temps, certaines pistes de ma carte ont brûlé après qu'une personne, avec mon accord, soit montée sur le robot. Il m'a donc fallu remplacer ces pistes par des fils et ce problème a également été résolu.

Conclusion

Quatorze semaines après le commencement du projet, le robot est entièrement contrôlable par Bluetooth. Le design du robot est le seul ajout nécessaire à l'aboutissement de mon projet. Voici à quoi celui-ci ressemble actuellement :

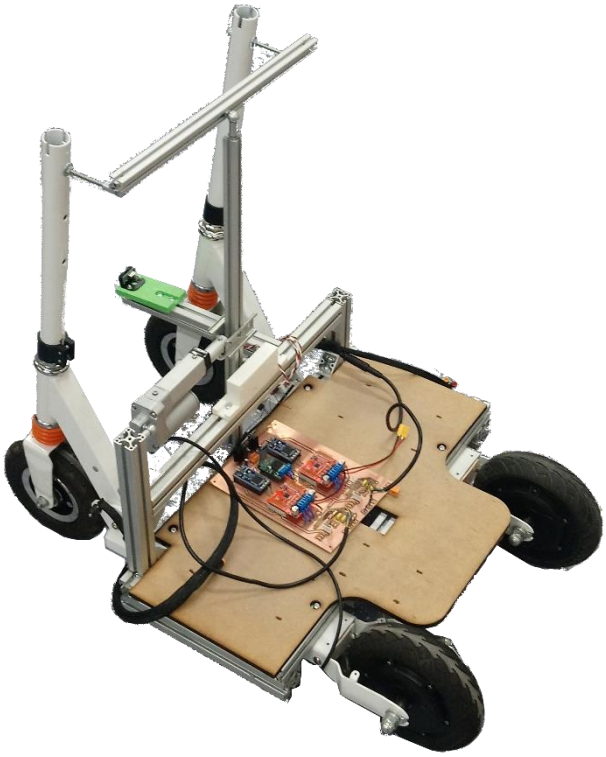

Figure 19 - Robot guide actuellement

Le projet du robot guide est une expérience riche en terme de connaissance technique.

En effet, grâce à ce projet, j'ai appris à mettre en place une communication Bluetooth, mais également à comprendre le fonctionnement de moteur « brushless », ce dernier étant énormément utilisé dans le milieu automobile.

J'ai également découvert de nouveaux composants tels que l'onduleur triphasé ou le module Bluetooth, mais j'ai aussi appris le langage C++, et je me suis perfectionné dans la conception de circuits imprimés.

Ainsi l'ensemble des compétences obtenues lors de ce projet de semestre 3 m'est utile pour la poursuite de mes études et le sera également pour ma carrière professionnelle.

Table des figures

Annexes

Table de vérité pour la mise en marche d'un moteur triphasé en utilisant l'onduleur :

Table 3. 1-PWM Active Freewheeling

STATE	INLA:INHB:INLB:INHC	GHA	GLA	GHB	GLB	GHC	GLC
AB	0110	PWM	!PWM	LOW	HIGH	LOW	LOW
AB_CB	0101	PWM	!PWM	LOW	HIGH	PWM	!PWM
CB	0100	LOW	LOW	LOW	HIGH	PWM	!PWM
CB_CA	1101	LOW	HIGH	LOW	HIGH	PWM	!PWM
CA	1100	LOW	HIGH	LOW	LOW	PWM	!PWM
CA_BA	1001	LOW	HIGH	PWM	!PWM	PWM	!PWM
BA	1000	LOW	HIGH	PWM	!PWM	LOW	LOW
BA_BC	1011	LOW	HIGH	PWM	!PWM	LOW	HIGH
BC	1010	LOW	LOW	PWM	!PWM	LOW	HIGH
BC_AC	0011	PWM	!PWM	PWM	!PWM	LOW	HIGH
AC	0010	PWM	!PWM	LOW	LOW	LOW	HIGH
AC_AB	0111	PWM	!PWM	LOW	HIGH	LOW	HIGH
Align	1110	PWM	!PWM	LOW	HIGH	LOW	HIGH
Stop	0000	LOW	LOW	LOW	LOW	LOW	LOW

Programme du robot guide :

File "/Robot_Guide/main.cpp" printed from os.mbed.com on 05/01/2020

```
#include "mbed.h"


//*************************
// Définition des constantes
//*************************
// Les pin de contrôle
#define ENGATE          p11
#define WAKE            p12
#define PWRGD           p10
#define FAULT           p9
#define MOSI            p5
#define MISO            p6
#define SCK             p7
#define SCS             p20
#define INHA            p21
#define INLA            p22
#define INHB            p23
#define INLB            p24
#define INHC            p19
#define INLC            p18

#define HALL_JAUNE      p15
#define HALL_BLEU       p16
#define HALL_VERT       p17

#define VERIN_INA       p27
#define VERIN_PWM       p26
#define VERIN_INB       p25

#define BLUETOOTH_TX    p13
#define BLUETOOTH_RX    p14


// Les configs SPI
#define SPI_frequency     1000000
#define SPI_bits     16
#define SPI_mode     1
#define SPI_read     (0x1 << 15)
#define SPI_write    (0x0 << 15)
#define SPI_reg1     (0x1 << 11)
#define SPI_reg2     (0x2 << 11)
#define SPI_reg3     (0x3 << 11)
#define SPI_reg4     (0x4 << 11)
#define SPI_reg5     (0x5 << 11)
#define SPI_reg6     (0x6 << 11)
#define SPI_reg7     (0x7 << 11)
#define SPI_reg8     (0x8 << 11)
#define SPI_reg9     (0x9 << 11)
#define SPI_reg10     (0xA << 11)
#define SPI_reg11     (0xB << 11);
#define SPI_reg12     (0xC << 11);

char PWM[6];


//*************************
// Instantiation des objets
//*************************
// Affectation des sorties digitales
DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut EnGate(ENGATE);
DigitalOut Wake(WAKE);
DigitalOut Scs(SCS);

DigitalOut Cmd_INLA(INLA);
DigitalOut Cmd_INHB(INHB);
DigitalOut Cmd_INLB(INLB);
DigitalOut Cmd_INHC(INHC);
DigitalOut Cmd_INLC(INLC);

```

```
73  DigitalOut Cmd_Verin_INA(VERIN_INA);
74  DigitalOut Cmd_Verin_INB(VERIN_INB);
75
76
77  // Affectation des entrées digitales
78  DigitalIn PwrGd(PWRGD);
79  DigitalIn Fault(FAULT);
80  DigitalIn Hall_Jaune(HALL_JAUNE);
81  DigitalIn Hall_Bleu(HALL_BLEU);
82  DigitalIn Hall_Vert(HALL_VERT);
83
84  // Affectation des sorties PWM
85  PwmOut Pwm_INHA(INHA);
86  PwmOut Pwm_Verin(VERIN_PWM);
87
88
89  // Affectation des liaisons séries
90  Serial USB_link(USBTX, USBRX);
91  Serial BLUETOOTH_link(BLUETOOTH_TX, BLUETOOTH_RX);
92
93
94  // Instantiation d'une liaison SPI en Master
95  SPI SPI_link(MOSI, MISO, SCK);
96
97  //*************************************
98  // Importation des fonctions
99  //*************************************
100 void Initialisation(void);
101 char Lecture_CapteursHall(void);
102 void Ecriture_PWM(char);
103 void Moteur(float speed);
104
105
106 //+++++++++++++++++++++++++++++++++++++++++++
107 // main() runs in its own thread in the OS
108 //+++++++++++++++++++++++++++++++++++++++++++
109 int main() {
110     // Variables du main
111     char c = '0';
112     float speed = 0;
113
114     // On commence par initialiser
115     Initialisation();
116
117     Pwm_INHA.write(speed);
118     Pwm_Verin.write(0.5);
119
120     // allumage onduleur
121     EnGate = 1;
122
123     // PWM State = Align
124     Ecriture_PWM(0b1110);
125
126     //  Boucle du main
127     while (true) {
128
129         if(BLUETOOTH_link.readable() == 1) {
130             c = BLUETOOTH_link.getc();
131         }
132
133         switch(c) {
134             case '1': // Up
135                 speed = speed + 0.05;
136                 if (speed > 1) speed = 1;
137                 c = '0';
138                 break;
139
140             case '3': // Down
141                 speed = speed - 0.05;
142                 if (speed < -1) speed = -1;
143                 c = '0';
144                 break;
145
146             case '4': // Left
147
```

```
                Pwm_Verin.write(0.5);
                Cmd_Verin_INA.write(1);
                Cmd_Verin_INB.write(0);
                break;

            case '2': // Right
                Pwm_Verin.write(0.5);
                Cmd_Verin_INA.write(0);
                Cmd_Verin_INB.write(1);
                break;


            case '0':
                Cmd_Verin_INA.write(0);
                Cmd_Verin_INB.write(0);
                break;
        }

        Moteur(speed);
    }

    return 0;
}


//*****************************************************************
// Fonctions
//*****************************************************************

void Initialisation() {
    bool PowerGood_OK = false;
    uint16_t data_in = 0, data_out = 0;

    // Initialisation PWM
    Pwm_INHA.period_us(50);

    PWM[0] = 0b1010;
    PWM[1] = 0b1100;
    PWM[2] = 0b1000;
    PWM[3] = 0b0110;
    PWM[4] = 0b0010;
    PWM[5] = 0b0100;


    //Sortie des commandes de l'onduleur en mode 1
    Cmd_INLA = 0 ;
    Cmd_INHB = 0;
    Cmd_INLB = 0;
    Cmd_INHC = 0;
    Cmd_INLC = 0;
    EnGate = 0;


    // Initialisation Communication
    USB_link.baud(115200);
    USB_link.format(8,SerialBase::None,1);
    BLUETOOTH_link.baud(9600);

    // premier message pour vérifier que tout est OK
    USB_link.printf("\r\n Liaison serie OK \r\n\n");

    // Initialisation Communication SPI
    SPI_link.frequency(SPI_frequency);
    SPI_link.format(SPI_bits,SPI_mode);
    Scs = 1;

    // vérification de l'alimentation de la carte onduleur
    do {
        PowerGood_OK = PwrGd;
        USB_link.printf("\r\n Power Good signal :%d \r\n\n",PowerGood_OK);
    } while(!PowerGood_OK);


    // lecture de la configuration du driver dans le registre 7
```

```
    data_in = SPI_read | SPI_reg7;

    Scs = 0;
    data_out = SPI_link.write(data_in);
    Scs = 1;

    USB_link.printf("\r\n l ancienne valeur du registre est :%x \r\n",data_out);

    // écriture de la configuration du driver dans le registre 7
    data_in = (0x1 << 9) | (0x2 << 7) | (0x1 << 4) | (0x1 <<2) | (0x2);
    USB_link.printf("\r\n la valeur du registre enregistree est :%x \r\n",data_in);

    data_in = SPI_write | SPI_reg7 | data_in;

    Scs = 0;
    data_out = SPI_link.write(data_in);
    Scs = 1;

    data_in = 0;
    data_in = SPI_read | SPI_reg7;

    Scs = 0;
    data_out = SPI_link.write(data_in);
    Scs = 1;
    USB_link.printf("\r\n la nouvelle valeur du registre est :%x \r\n",data_out);

    // Allumage led1

    led1 = true;

    return;
}

char Lecture_CapteursHall() {
    int HallJ, HallB, HallV;

    HallJ = Hall_Jaune.read();
    HallB = Hall_Bleu.read();
    HallV = Hall_Vert.read();

    // USB_link.printf("\n\r Effet hall : %d%d%d", HallJ, HallB, HallV);

    return (HallJ << 2) | (HallB << 1) | (HallV);
}

void Ecriture_PWM(char PWM_binary) {
    int a, b, c, d;

    a =  0 != (PWM_binary & (1 << 3));
    b =  0 != (PWM_binary & (1 << 2));
    c =  0 != (PWM_binary & (1 << 1));
    d =  0 != (PWM_binary & (1 << 0));

    // USB_link.printf("\n\r Binaire : %d%d%d%d\n\r", a,b,c,d);

    Cmd_INHC.write(d);
    Cmd_INLA.write(a);
    Cmd_INHB.write(b);
    Cmd_INLB.write(c);

    Cmd_INHC.write(1);
    Cmd_INLA.write(a);
    Cmd_INHB.write(b);
    Cmd_INLB.write(c);
}

void Moteur(float vitesse) {

    // Si c'est en marche avant
    if (vitesse > 0) {
        Pwm_INHA.write(vitesse);

        switch (Lecture_CapteursHall()) {
            case 0b001:
```

```
                    Ecriture_PWM(0b1010);
                    break;
                case 0b010:
                    Ecriture_PWM(0b1100);
                    break;
                case 0b011:
                    Ecriture_PWM(0b1000);
                    break;
                case 0b100:
                    Ecriture_PWM(0b0110);
                    break;
                case 0b101:
                    Ecriture_PWM(0b0010);
                    break;
                case 0b110:
                    Ecriture_PWM(0b0100);
                    break;
            }
        }

        // Si c'est en marche arrière
        else if (vitesse < 0) {
            Pwm_INHA.write(vitesse*(-1));

            switch (Lecture_CapteursHall()) {
                case 0b001:
                    Ecriture_PWM(0b0100);
                    break;
                case 0b010:
                    Ecriture_PWM(0b0010);
                    break;
                case 0b011:
                    Ecriture_PWM(0b0110);
                    break;
                case 0b100:
                    Ecriture_PWM(0b1000);
                    break;
                case 0b101:
                    Ecriture_PWM(0b1100);
                    break;
                case 0b110:
                    Ecriture_PWM(0b1010);
                    break;
            }
        }
    }
```

File "/Robot_Guide/main.cpp" printed from os.mbed.com on 05/01/2020