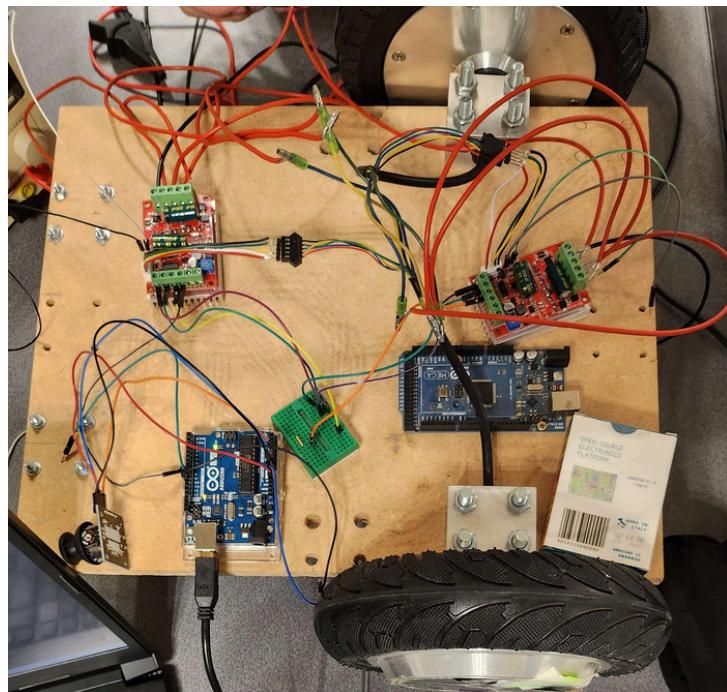


Projet BEAM - Moteurs brushless:

Document technique

(Octobre 2023 - Avril 2024)



Réalisé par:

Matthieu MALFROY (IESE4)
Arthur GAUDEBERT (IESE4)
Quentin GOIZET (IESE4)

Supervisé par:

Germain LEMASSON (Ingénieur)

I. Introduction.....	2
II. Démarche de résolution.....	2
A. 1er partie du projet : software.....	2
B. 2nde partie du projet : hardware.....	4
III. PWM.....	6
A. Contrôle de la vitesse des moteurs.....	6
B. Génération d'un PWM.....	7
IV. Carte de puissance.....	7
A. Branchement d'un moteur avec 5 fils de retours.....	8
B. Branchement d'un moteur avec 6 fils de retours (robot BEAM).....	10
V. Contrôle de la PWM.....	10
A. Joystick (filaire).....	10
B. Téléphone (Bluetooth).....	11
VI. Annexe.....	13
1. Code contrôle avec joystick.....	13
2. Code contrôle depuis le téléphone.....	16

I. Introduction

L'objectif premier est de pouvoir réutiliser les composants présents dans le robot BEAM pour pouvoir le déplacer à notre guise (sans utiliser les outils associés au robot). Pour cela, nous chercherons dans un premier temps, les codes pour contrôler la carte de puissance/les roues. Après un changement de stratégie, nous utilisons nos propres cartes de puissance par les roues, toutes deux connectées à un microcontrôleur choisi. Deux façons de commander les déplacements seront présentées: grâce à un joystick directement connecté à la carte ou grâce à un smartphone connecté en Bluetooth à la carte.

II. Démarche de résolution

Notre projet étant particulier, nous avons réalisé des travaux radicalement différents en fonction de l'avancement du projet. Dans un premier temps, notre projet a été purement software, et se composait presque essentiellement de travaux de recherche. Après plusieurs mois, nous nous sommes retrouvés dans l'impasse. A partir de là, avec l'accord de notre porteur de projet, nous avons basculé dans des tâches majoritairement hardware.

A. 1er partie du projet : software

La première étape de cette partie a été la compréhension du fonctionnement intrinsèque du robot. Nous l'avons donc démonté et découvert qu'il était composé de deux parties distinctes : une base comprenant une carte électronique et une "tête" composé d'un petit ordinateur. Nous avons donc divisé le travail en deux : l'étude de la carte et l'étude de l'ordinateur.

Etude de la carte :

Dans cette partie, nous avons dû comprendre à quoi servait chaque composant présent sur cette carte (figure IV.1). Nous avons commencé par chercher la documentation de celle-ci, mais comme le robot a été conçu par une entreprise privée, celle-ci n'est pas disponible. Après plusieurs heures de recherche, nous avons fini par comprendre ce que renfermait cette carte.

Dans l'utilisation classique du robot, l'alimentation était assurée par une batterie 12V. La première partie de la carte était destinée à gérer le courant apporté par celle-ci (en violet sur la figure IV.1). Il y a également une grande partie de la carte qui est un driver brushless destiné à adapter la puissance et à contrôler les deux moteurs brushless présents dans les roues (en rouge sur la figure IV.1). Cette carte est

également composée d'un microcontrôleur STM32 destiné à générer les signaux PWM voulu par le driver (en bleu sur la figure 1). La dernière partie de cette carte est destinée à alimenter et à communiquer avec l'ordinateur présent dans la partie supérieure du robot via une liaison série (en jaune sur la figure 1).

Bien que nous n'avions pas les détails de fonctionnement de cette carte, nous savions qu'elle était contrôlée entièrement par l'ordinateur. Notre but étant de conserver un maximum de composants d'origine, dont cette carte, nous nous sommes intéressés à la tête du robot.

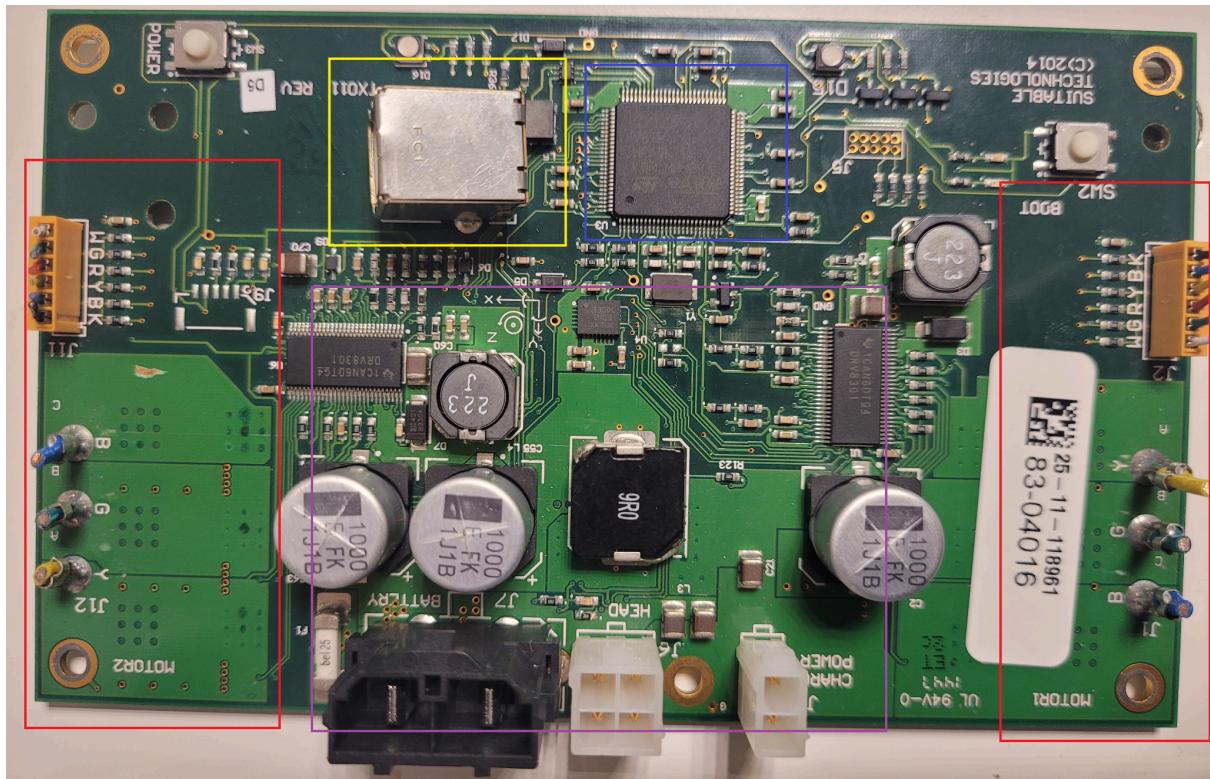


Figure 1: Microcontrôleur intégré à une carte de puissance issu du socle du robot BEAM

Etude de l'ordinateur interne du robot :

L'objectif de cette partie était de comprendre le fonctionnement de l'ordinateur interne et de pouvoir écrire un programme dans celui-ci permettant de contrôler la carte. Dans un premier temps, nous nous sommes rendu compte que le code présent dans l'ordinateur n'était pas un programme tournant sous Linux comme dans de nombreuses utilisations similaires. Il s'agissait d'un tout autre système d'exploitation créé spécialement pour le robot BEAM. Après de nombreuses recherches, nous avons trouvé un moyen d'obtenir les fichiers présents dans le robot. Nous avons créé une clé USB contenant un Boot d'Ubuntu (une version de Linux temporaire). Grâce à cela, nous avons pu booter l'ordinateur sur ce système d'exploitation temporaire. L'objectif de cette manipulation était d'identifier le disque dur interne, puis de le monter sur notre nouveau système d'exploitation dans le but

d'en extraire les données. Après plusieurs heures de manipulation, ralenties par un faux contact dans la carte d'alimentation qui redémarre l'ordinateur du robot au bout de 5 minutes d'utilisation, nous avons finalement pu récupérer ces données. Nous avons par la suite passé une grande partie de notre temps à nous renseigner sur l'architecture d'un système d'exploitation et à rechercher dans les fichiers du robot les codes permettant de contrôler la carte. Après plusieurs séances et une réunion avec Awabot, la société qui commercialise le robot, nous nous sommes rendu compte que les travaux prévus initialement, à savoir comprendre et concevoir un code à partir des composants existants, étaient hors d'atteinte dans le temps qu'il nous restait. Après une réunion avec M. LEMASSON, nous avons donc convenu de passer à une approche hardware.

B. 2nde partie du projet : hardware

Nous allons complètement changer d'approche. En effet, nous allons mettre de côté la carte électronique présente dans le robot pour utiliser nos propres cartes. L'objectif que nous nous sommes fixé ici, étant donné le temps qui nous restait, était de contrôler le robot grâce à une application mobile via Bluetooth. Dans un premier temps, nous n'avions qu'un seul driver brushless récupéré d'un ancien robot (Figure 2). Nous avons donc compris le fonctionnement global d'un driver grâce à ce dernier, en attendant que les drivers neufs arrivent. Nous avons constaté que l'on pouvait contrôler la puissance envoyée à chaque roue en faisant varier le rapport cyclique d'un PWM externe. Pour générer ce PWM et faire varier son rapport cyclique en fonction des besoins, nous allons donc devoir utiliser également un microcontrôleur. À partir de là, nous avons séparé le travail en deux : Arthur s'occupait de la gestion du microcontrôleur et du code permettant de piloter les drivers, tandis que Quentin et Mathieu s'occupaient de monter les drivers en les connectant aux roues du robot.

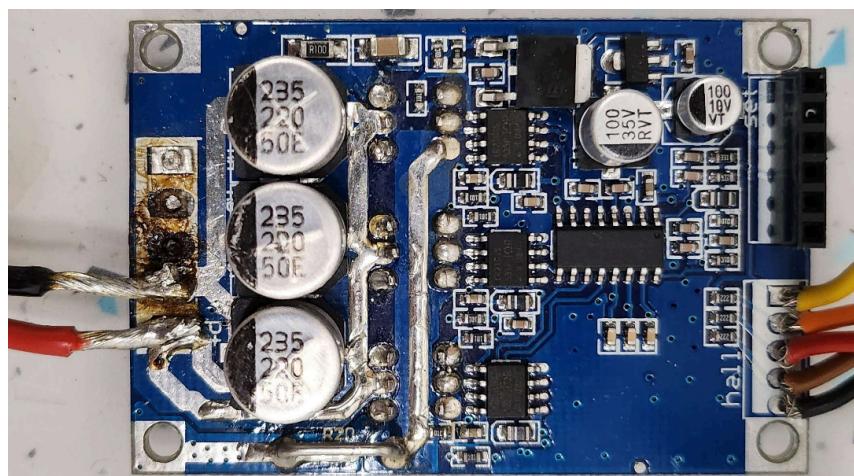


Figure 2: Ancien driver brushless

Choix du microcontrôleur et code:

Dans un premier temps, nous avons utilisé une carte Arduino Uno en raison de sa facilité d'utilisation et de notre connaissance de celle-ci. Ensuite, nous avons généré deux signaux PWM d'une fréquence de l'ordre du kilohertz (comme spécifié dans la documentation des drivers). Ensuite, nous avons conçu un programme permettant de faire varier le rapport cyclique de ces PWM en utilisant un joystick. Notre programme permettait également le contrôle du bit du driver nommé DIR, chargé de la direction des roues. Après avoir validé ce code et cette méthode, nous avons basculé vers l'utilisation du microcontrôleur ESP32 (Figure 3). La raison à cela est que ce microcontrôleur possède une carte Bluetooth, indispensable pour le contrôle du robot à distance. Nous avons donc réécrit tout le code en adaptant ce dernier pour le rendre compatible avec l'ESP32. Nous avons également implémenté la fonctionnalité Bluetooth pour que les PWM puissent être modifiés à partir d'un smartphone.

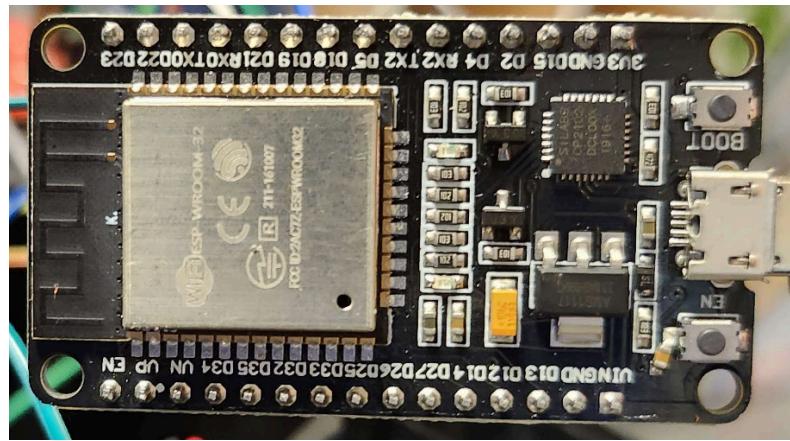


Figure 3: Carte ESP32 pour le contrôle à distance

Connection des drivers aux moteurs du robot :

Parallèlement à la conception du code, nous avons également implémenté les nouveaux drivers de roue (figure 4) sur le robot BEAM. Pour cela, nous avons dû démonter le robot et retirer son ancienne carte. Nous avons également dû démonter les roues afin de comprendre le rôle de tous les connecteurs présents. Après cette étape d'identification, nous avons connecté les drivers aux deux roues du robot. Ensuite, nous avons testé le bon fonctionnement de ces derniers grâce au potentiomètre interne au driver. À ce stade, nous avons rencontré plusieurs problèmes que nous avons mis un certain temps à identifier et à corriger. Par exemple, un faux contact dans les fils du robot nous a obligé à refaire tous les câblages et les soudures.

Après ces deux étapes, nous avons finalement tout mis en commun. Nous avons connecté l'ESP 32 aux drivers et nous avons connecté un téléphone en Bluetooth au microcontrôleur pour contrôler directement les roues. Nous avons ainsi constaté que

le programme fonctionnait correctement et permettait le mouvement des roues de manière adéquate.

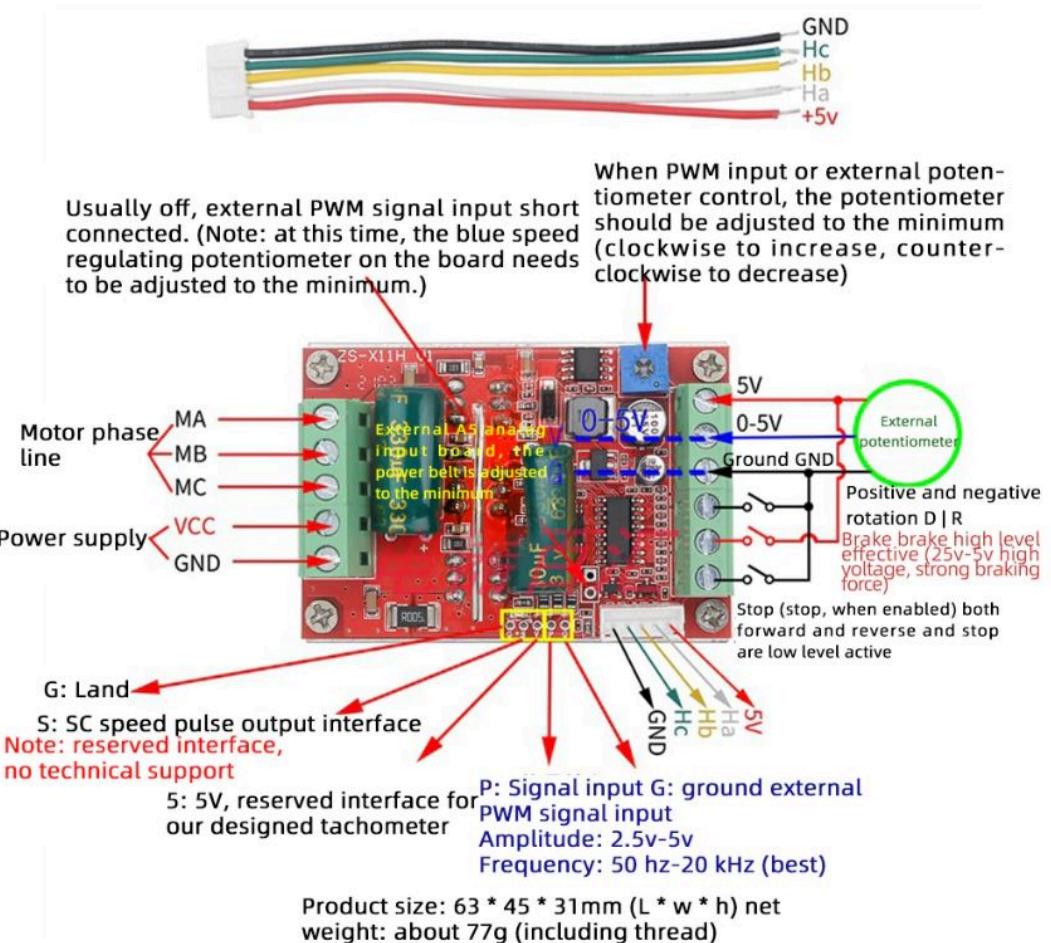


Figure 4: Nouvelles cartes de puissance (extrait de <https://www.elektrobot.hu/termek.php?filename=7776.html&i=7776>)

III. PWM

A. Contrôle de la vitesse des moteurs

Le contrôle des moteurs se fait grâce à une PWM (Pulse Width Modulation) où le rapport cyclique permet de contrôler la vitesse de rotation. En effet, la tension moyenne délivrée au moteur par la PWM en bas de la *Figure 5* est plus élevée que celle délivrée par celle d'en hautain donc le moteur alimenté par la PWM du bas aura une rotation plus élevée que celui alimenté par celle du haut.

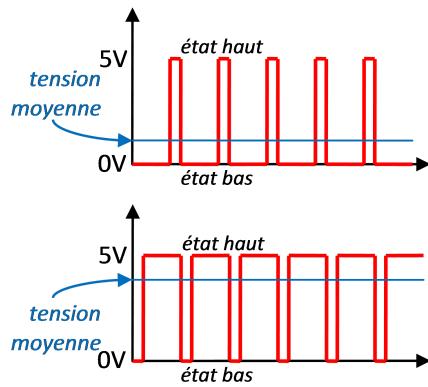


Figure 5: Exemples de PWM

B. Génération d'un PWM

Pour générer les PWM, il faut utiliser certaines pattes dédiés pour cette utilisation (souvent, il y a inscrit “pwm” à côté des pattes compatibles). Dès lors, ces pattes doivent être configurées en sortie. Puis il faut fixer la fréquence de notre PWM et ceci passe généralement l’utilisation des registres des timer. Par exemple, pour avoir une fréquence de 7.8 Hz sur la patte d’une *Arduino Uno* il faut utiliser les lignes de code en *Figure 6*.

```
// Configuration of timer1 (used by pins 9 and 10) at 7.8 kHz
TCCR1A = TCCR1A & _0xe0 | 1;
TCCR1B = TCCR1B & _0xe0 | _0x0a;
```

Figure 6: Fixer la fréquence des PWM

Attention, une plage de fréquence est imposée par la carte de puissance. Dans notre cas, la fréquence de notre PWM doit être fixée entre 50Hz et 20kHz. Il est préférable de prendre une fréquence élevée pour réduire le bruit, avoir une meilleure résolution de contrôle, réduire les vibrations acoustiques.

Finalement, on peut contrôler le rapport cyclique en écrivant une valeur analogique (*AnalogueWrite*) comprise entre 0 et 255 avec 0 pour un rapport quasi-nul et 255 pour un rapport maximal.

IV. Carte de puissance

La carte de puissance utilisée (ZS-X11H) et ses branchements sont représentés en *Figure 7*.

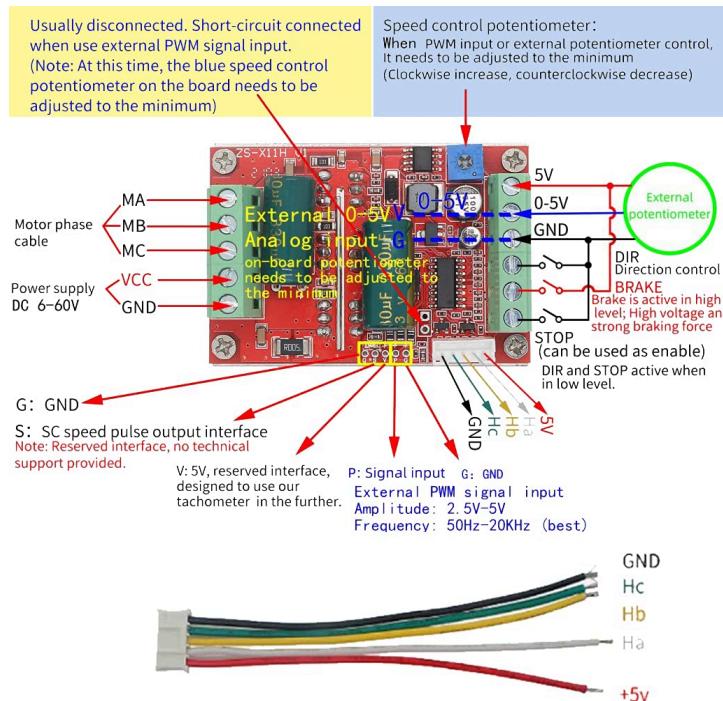


Figure 7: Carte de puissance (extrait de <https://www.elektrobot.hu/termek.php?filename=7776.html&i=7776>)

Cette carte permet de contrôler nos moteurs avec 3 entrées différentes:

- Un potentiomètre intégré
- Un potentiomètre externe (Vert)
- Une PWM (Bleu)
- VCC et GND pour notre source de tension

Dans notre cas, nous utiliserons donc les entrées:

- P,G pour notre PWM
- DIR pour contrôler le sens de rotation des moteurs
- BRAKE pour freiner

Nous utiliserons également les sorties MA, MB et MC pour les trois phases de notre moteur, il faudra donc aussi utiliser les retours des effets Halls (Ha et à Hc correspondant respectivement aux sorties MA à MC) et les 5V - GND.

A. Branchement d'un moteur avec 5 fils de retours

Les moteurs de la *Figure 8*, présente 5 fils de retours et peuvent être alimentés par une tension de 24V (voire 32V je crois ?).



Figure 8: Moteur avec 5 retours

Comme le montre la *Figure 9*, le moteur possède trois gros câbles de couleurs jaune, vert et bleu. On retrouve ces mêmes couleurs avec des petits fils avec en plus, un rouge et un noir. Les gros fils permettent de transmettre de grosse puissance, c'est donc donc grâce à ces trois là que nous alimenterons notre exemple, ainsi nous branchons, par exemple, le jaune sur MA, le bleu sur MB et le vert MC. Il faut veiller conserver le bon cycle, il en existe deux possibles:

- JAUNE -> BLEU -> VERT -> JAUNE -> BLEU -> VERT -> JAUNE ...
- JAUNE -> VERT -> BLEU -> JAUNE -> VERT -> BLEU -> JAUNE ...

Ce cycle est fixé par l'ordre des phases dans la roue (aimants qui vont être alimentés tour à tour, dans un ordre ou l'autre).

Puis, cet ordre impose l'ordre des branchements des retours hall, il faudra donc placé le jaune Ha, le bleu sur Hb et vert sur Mc pour qu'à chaque alimentation, correspond le bon retour. Finalement, le rouge est connecté au 5V alors que le noir l'est à la masse.



Figure 9: Câbles du moteur à 5 retours

B. Branchement d'un moteur avec 6 fils de retours (robot BEAM)

Les roues du robot de téléprésence BEAM (Figure 10) se différencient seulement sur un point: il y a 6 retours. Les 5 fils, dont on retrouve les mêmes couleurs sur roues précédentes, jouent le même rôle. La différence est l'ajout d'un fil blanc. Celui-ci permet de récupérer la température interne à la roue. Les branchements à la carte de puissance restent inchangés, mais nous pourrions récupérer cette donnée directement sur notre microcontrôleur.



Figure 10: Intérieur de la roue du BEAM

V. Contrôle de la PWM

A. Joystick (filaire)

Pour tester rapidement comment on règle les rapports cycliques pour obtenir les bons mouvements, on se propose d'utiliser un joystick filaire, directement connecté à une arduino Uno. Le code (104 lignes) se trouve en annexe (1. Code contrôle avec joystick). Le code permet d'initialiser une PWM à 7800 Hz, un baudrate pour vérifier nos tests sur un écran, les pins (sorties ou entrées) et la valeur initiale du joystick. Puis dans un boucle, on lit les valeurs (2D) de notre joystick, on les met à l'échelle pour qu'elle puisse correspondre à des valeurs de rapports cycliques, puis suivant les différents cas (avancer, reculer, avancer en tournant à droite...), on met à jour les

rapports. Attention à n'écrire que des valeurs positives, pour inverser le sens position, il faut inverser le bit de Dir (responsable du sens de rotation de la roue).

B. Téléphone (Bluetooth)

Nous voulons maintenant contrôler notre carte de puissance avec une technologie sans fil Bluetooth. Pour cela, nous allons laisser de côté le microcontrôleur Arduino Uno pour utiliser le ESP 32. Après avoir connecté l'ESP 32 avec l'IDE arduino, nous allons pouvoir commencer à programmer ce dernier. Le but est donc de connecter un téléphone à l'ESP32 via Bluetooth, puis d'influer sur deux sorties PWM grâce à des données envoyées via Bluetooth. Le code est donné en annexe (2. Code contrôle depuis le téléphone).

Pour cette partie, plusieurs choix techniques ont été nécessaires. Dans un premier temps, nous avons opté pour une application mobile permettant la communication en Bluetooth. Ainsi, notre choix s'est porté sur l'application Arduino Bluetooth Controller, car elle intègre directement un "gamepad" qui sera utile pour contrôler le robot (voir *Figure 11*).



Figure 11 : gamepad présent sur l'application Arduino Bluetooth controller

Cette application enverra une lettre différente sur le port série à chaque pression sur un bouton du gamepad. Une fois le bouton relâché, l'application enverra '0' sur le port.

En ce qui concerne le contrôle des PWM (Pulse Width Modulation), l'un des avantages d'avoir choisi l'ESP32 réside dans la manière dont le microcontrôleur gère ces signaux. Contrairement à l'Arduino UNO, qui nécessite de modifier la fréquence de l'horloge du Timer correspondant au PWM, l'ESP32 dispose d'une fonction qui

permet de gérer facilement et précisément la fréquence du PWM. Pour cela, nous utiliserons les fonctions suivantes :

```
ledcSetup(pwmChannel, frequence, resolution); // Initialise un PWM sur  
un channel
```

```
ledcAttachPin(pwmPin, pwmChannel); // Attache le channel à un pin de  
l'ESP 32
```

```
ledcWrite(pwmChannel, val); // Modifie la valeur du rapport cyclique du  
PWM
```

Une fois ces choix effectués, nous avons conçu un code qui prend en compte tous ces paramètres.

Explication du code :

Ce code est conçu pour être utilisé avec un ESP32 pour contrôler un robot via Bluetooth:

1. **Bibliothèques incluses** :

- `BluetoothSerial.h`: Cette bibliothèque est utilisée pour la communication Bluetooth série.

2. **Définitions et configurations initiales** :

- Le code commence par définir un code PIN pour le jumelage Bluetooth. Dans ce cas, le PIN est défini comme "1234".
- Il déclare également le nom du périphérique Bluetooth, ici défini comme "ESP32-BT-Slave".
- Il vérifie également si la configuration Bluetooth et la prise en charge du Bluetooth SPP (Serial Port Protocol) sont activées sur l'ESP32.

3. **Configuration des signaux PWM** :

- Le code initialise les signaux PWM pour contrôler les moteurs du robot. Il choisit un canal pour chaque moteur, une fréquence de PWM de 10 kHz et une résolution de 8 bits (256 valeurs possibles).
- Les broches GPIO correspondantes sont définies pour les signaux PWM et pour les pins de direction des moteurs.

4. **Fonction `setup()`** :

- Dans la fonction `setup()`, la communication série est initialisée à une vitesse de 9600 bauds.
- La connexion Bluetooth série est également initialisée avec le nom du périphérique.

- Les broches GPIO sont configurées en sortie et certaines sont initialisées avec des valeurs spécifiques, notamment pour la direction et le freinage des moteurs.

5. **Fonction `loop()`** :

- La fonction `loop()` vérifie s'il y a des données disponibles sur la connexion Bluetooth série.
- Si des données sont disponibles, la fonction `move()` est appelée pour interpréter et exécuter la commande.

6. **Fonction `move(char order)`** :

- Cette fonction est responsable de l'interprétation des commandes reçues via Bluetooth et du contrôle des moteurs en conséquence.
- Elle accepte un caractère (`order`) qui représente la commande à exécuter.
- En fonction de la commande reçue, les valeurs PWM des moteurs sont ajustées pour contrôler la direction et la vitesse du robot.

7. **Fonction `resetLast()`** :

- Cette fonction est utilisée pour réinitialiser les paramètres des moteurs lorsque la commande "0" est reçue. Elle est appelée pour arrêter les moteurs et réinitialiser les valeurs PWM.

VI. Annexe

1. Code contrôle avec joystick:

```
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output
int X0, Y0, x, y, Xmap, Ymap;

void setup() {
    // Initialise la broche 9 en tant que sortie PWM
    initTimer1_7800Hz();
    // Baudrate pour l'affichage
    Serial.begin(9600);
    initPin();
    initReadJoy();
    move_joystick(0,0);
}

}
```

```

void initTimer1_7800Hz(){ // Regler la PWM
    TCCR1A = TCCR1A & 0xe0 | 1;
    TCCR1B = TCCR1B & 0xe0 | 0x0a;
}

void initReadJoy(){ // Initialisation du joystick (joystick au centre)
    X0=analogRead(X_pin);
    Y0=analogRead(Y_pin);
    x=0;
    y=0;
}

void initPin(){ // Configuration des PINs
    pinMode(10, OUTPUT); // L wheel
    pinMode(9, OUTPUT); // R wheel
    pinMode(8, OUTPUT); // DIR Droite
    pinMode(7, OUTPUT); // DIR Gauche
    pinMode(5, INPUT);
    pinMode(6, INPUT);
    digitalWrite(8, 1); //Dir at 1 (default)
}

void move_joystick(int8_t valueX, int8_t valueY){ // Value_max = 15.5%
* 255/100 = 40, value entre -40 et 40
    if (valueX < 0 && valueX >= -40) { //En arriere
        Serial.print("Dir inverser\n");
        digitalWrite(8, 0); //Dir a 0 pour inverser le sens de rotation
        digitalWrite(7, 1); //Dir at 1 (default)
        valueX = -valueX;
    }
    else if (valueX > 0 && valueX <= 40) { //En avant
        digitalWrite(8, 1); //Dir a 1
        digitalWrite(7, 0); //Dir a 0
    }
    else if (valueX==0){ // Rotation (deux sens possibles)
        if (valueY < 0){
            valueX = -valueY/2;
        }
        else {
            valueX = valueY/2;
        }
    }
}

```

```

if (valueY < 0 && valueY >= -40) { // Right wheel faster
    valueY = -valueY;
    if (valueX - valueY/2 < 0) {
        analogWrite(10, 0);
    }
    else {
        analogWrite(10, valueX - valueY/2);
    }
    analogWrite(9, valueX + valueY/2);
}
else if (valueY <= 40){ // Left wheel faster
    analogWrite(10, valueX + valueY/2);
    if (valueX - valueY/2 < 0) {
        analogWrite(9, 0);
    }
    else {
        analogWrite(9, valueX - valueY/2);
    }
}

void move_affiche(int x, int y){ // Retour des commandes sur l'écran
    Serial.print("\n");
    Serial.print("X-axis: ");
    Serial.print(x);
    Serial.print("\n");
    Serial.print("Y-axis: ");
    Serial.println(y);
    Serial.print("\n\n");
}

void readXY(){ // Lire valeur analogique du Joystick
    if(x!=analogRead(X_pin)-X0 || y!=analogRead(Y_pin)-Y0){ // Màj de la
position seulement quand le joystick bouge
        x=analogRead(X_pin)-X0;
        y=analogRead(Y_pin)-Y0;

        Xmap=map(x,-512,512,-40,40); // Mise à l'échelle des valeurs dans
nos plages
        Ymap=map(y,-512,512,-40,40);
        move_affiche(Xmap,Ymap); // Visualisation de la commande
        move_joystick(Xmap,Ymap); // Maj des rapports cycliques de la PWM
    }
}

```

```

}

// Boucle d'utilisation
void loop() {
    readXY();
    delay(1);
}

```

2. Code contrôle depuis le téléphone

```

#include "BluetoothSerial.h"

#ifndef USE_PIN // Uncomment this to use PIN during pairing. The pin
is specified on the line below
const char *pin = "1234"; // Change this to more secure PIN.

String device_name = "Jerome";

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and
enable it
#endif

#if !defined(CONFIG_BT_SPP_ENABLED)
#error Serial Bluetooth not available or not enabled. It is only
available for the ESP32 chip.
#endif

#define Speed 50 // On choisit un duty cicle des PWMs qui vont
controler la vitesse de rotation des roues : Valeur entre 0 et 254
#define SpeedTurn 30 // On choisit un duty cicle à ajouter à une des
deux roues pour initier un virage

int pwmChannelG = 0; //Choisit le canal 0 pour la roue gauche
int pwmChannelD = 1; //Choisit le canal 1 pour la roue droite
int frequence = 10000; //Fréquence PWM de 10 KHz
int resolution = 8; // Résolution de 8 bits, 256 valeurs possibles
int pwmPinG = 23;

```

```

int pwmPinD = 22;
BluetoothSerial SerialBT;
int valG=0;
int valD=0;
char LC = '9'; //LastChar, dernier valeur connu de la commande bluetooth, initialisé à une valeur absurde pour ne pas perturber le code
char LCC = '9'; // Avant dernière valeur connu de la commande bluetooth, initialisé à une valeur absurde pour ne pas perturber le code
int dirG = 19; //Pin DIR de la roue gauche
int dirD = 21; //Pin DIR de la roue droite

void setup() {
    Serial.begin(9600);
    SerialBT.begin(device_name); //Bluetooth device name
    Serial.printf("The device with name \"%s\" is started.\nNow you can pair it with Bluetooth!\n", device_name.c_str());
    //Serial.printf("The device with name \"%s\" and MAC address %s is started.\nNow you can pair it with Bluetooth!\n", device_name.c_str(), SerialBT.getMacString()); // Use this after the MAC method is implemented
#ifdef USE_PIN
    SerialBT.setPin(pin);
    Serial.println("Using PIN");
#endif
    // Configuration du canal 0 avec la fréquence et la résolution choisie
    ledcSetup(pwmChannelG, frequence, resolution);
    ledcSetup(pwmChannelD, frequence, resolution);

    // Assigne le canal PWM au pin 23 et 22
    ledcAttachPin(pwmPinG, pwmChannelG);
    ledcAttachPin(pwmPinD, pwmChannelD);

    //Initialisation des pins
    pinMode(dirG,OUTPUT);
    pinMode(dirD,OUTPUT);
    pinMode(stopPin,OUTPUT);
    digitalWrite(dirG,0);
    digitalWrite(dirD,1);
    digitalWrite(stopPin,1);
}

```

```

}

//Boucle principal
void loop() {
    if (SerialBT.available()) {
        move(SerialBT.read());
    }
    delay(1);
}

//Valeur retourné par le contrôleur Bluetooth de l'application Arduino
Bluetooth Control
//F : flèche avant : avancer
//R L : Right et Left : demi tour sur place à droite et à gauche
//B : flèche arrière : reculer

//T (triangle) : avancer
//C (circle) : Tourner à droite à droite en roulant
//S (square) : Tourner à gauche gauche en roulant
//X : Reculer

//Le contrôleur renvoi '0' lorsqu'on arrête d'appuyer sur une touche

void move(char order) {
    if(order == 'F' || order == 'T'){
        valG=Speed;
        valD=Speed;
        ledcWrite(pwmChannelG, valG);
        ledcWrite(pwmChannelD, valD);
        LC=order;
        LCC=order;
    }
    if(order == 'R'){
        digitalWrite(dirD,0);
        valG=Speed;
        valD=Speed;
        ledcWrite(pwmChannelG, valG);
        ledcWrite(pwmChannelD, valD);
        LC=order;
        LCC=order;
    }
    if(order == 'L'){

```

```

digitalWrite(dirG,1);
valG=Speed;
valD=Speed;
ledcWrite(pwmChannelG, valG);
ledcWrite(pwmChannelD, valD);
LC=order;
LCC=order;
}

if(order == 'C'){
valG+=SpeedTurn;
ledcWrite(pwmChannelG, valG);
ledcWrite(pwmChannelD, valD);
LCC=LC;
LC=order;

}

if(order == 'S'){
valD+=SpeedTurn;
ledcWrite(pwmChannelG, valG);
ledcWrite(pwmChannelD, valD);
LCC=LC;
LC=order;
}

if(order == 'B' or order == 'X'){
digitalWrite(dirG,1);
digitalWrite(dirD,0);
valG=Speed;
valD=Speed;
ledcWrite(pwmChannelG, valG);
ledcWrite(pwmChannelD, valD);
LC=order;
LCC=order;
}

if(order == '0'){
resetLast();
LC=order;

}

Serial.write(order);

```

```

}

// Fonction qui reset les valeurs en fonction de la valeur recu avant
la valeur '0'

void resetLast() {
    switch(LC) {
        case 'F':
            valG=0;
            valD=0;
            ledcWrite(pwmChannelG, valG);
            ledcWrite(pwmChannelD, valD);
            break;

        case 'T':
            valG=0;
            valD=0;
            ledcWrite(pwmChannelG, valG);
            ledcWrite(pwmChannelD, valD);
            break;

        case 'C':
            valG-=SpeedTurn;
            ledcWrite(pwmChannelG, valG);
            ledcWrite(pwmChannelD, valD);
            break;

        case 'S':
            valD-=SpeedTurn;
            ledcWrite(pwmChannelG, valG);
            ledcWrite(pwmChannelD, valD);
            break;

        case '0':
            digitalWrite(dirG,0);
            digitalWrite(dirD,1);
            valG=0;
            valD=0;
            ledcWrite(pwmChannelG, valG);
            ledcWrite(pwmChannelD, valD);
            break;

        case 'R':
            digitalWrite(dirG,0);
            digitalWrite(dirD,1);
    }
}

```

```

    valG=0;
    valD=0;
    ledcWrite(pwmChannelG, valG);
    ledcWrite(pwmChannelD, valD);
    break;

case 'L':
    digitalWrite(dirG,0);
    digitalWrite(dirD,1);
    valG=0;
    valD=0;
    ledcWrite(pwmChannelG, valG);
    ledcWrite(pwmChannelD, valD);
    break;

case 'B':
    digitalWrite(dirG,0);
    digitalWrite(dirD,1);
    valG=0;
    valD=0;
    ledcWrite(pwmChannelG, valG);
    ledcWrite(pwmChannelD, valD);
    break;

case 'X':
    digitalWrite(dirG,0);
    digitalWrite(dirD,1);
    valG=0;
    valD=0;
    ledcWrite(pwmChannelG, valG);
    ledcWrite(pwmChannelD, valD);
    break;

default :
    break;
}

}

```