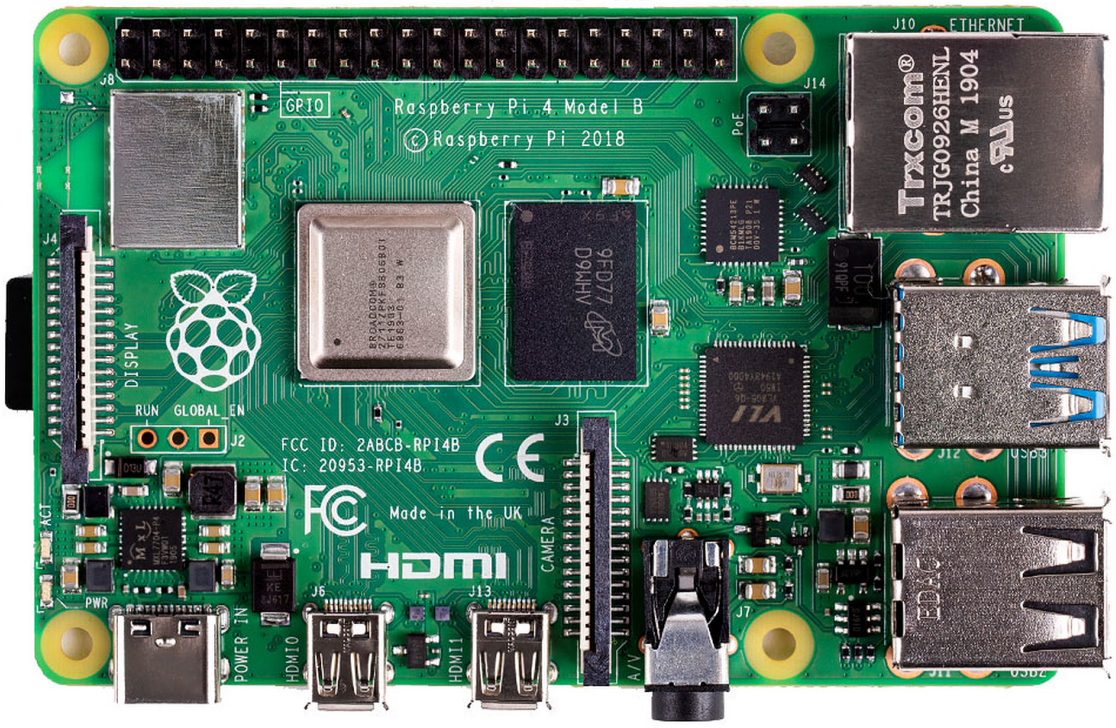
INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE



Thomas BAUER
Angelo BOUTANOUS
Ali HAMDANI
Nathan LEGENDRE

Robot suiveur de ligne

Projet d'électronique pour systèmes embarqués

Electronique - ITI - 2023-2024

Table des matières

Introduction

Dans le cadre du cours Électronique pour les systèmes embarqués nous avons réalisé un projet de robot suiveur de ligne. Le but étant d'appliquer sur un exemple simple et concret les notions de travaux dirigés, travaux pratiques et de cours.

Ce travail a été réalisé en groupe de 4 : Thomas Bauer, Angelo Bou-Tanous, Ali Hamdani et Nathan Legendre. Il a été supervisé par Mme. Laghmara.

Le projet consiste globalement en un robot qui selon la couleur perçue change de direction ou pas. À cela, nous ajoutons les contraintes suivantes. Le robot doit être capable de détecter un obstacle et de s'arrêter mais également de détecter une intersection. Les informations importantes seront également affichées sur un écran LCD ou alors sonore. Nous avons décidé d'ajouter au robot la possibilité d'être contrôlé via une manette de PlayStation 5.

Afin de mener à bien ce projet, à partir des caractéristiques techniques demandées pour le robot nous avons d'abord réfléchi aux composants adéquats pour réaliser les tâches.

Une fois les composants déterminées nous avons réfléchi à la meilleure façon de relier ces composants au Raspberry Pi (cela sera détaillé dans la section Fritzing) puis nous avons développer les programmes permettant de les faire interagir de la manière souhaitée.

Nous évoquerons évidemment les problèmes rencontrés au cours des semaines de progression sur ce projet et les solutions qui nous ont permis de finaliser le robot suiveur de ligne.

Cahier des charges

- Suiveur de ligne
- Détection des intersections
- Mesurer et afficher la distance frontale avec un objet
- Alerter si il y a un obstacle
- Arrêt d'urgence si il y a un obstacle. Repartir lorsque la voie est libre
- Contrôler manuellement

1 Répartition du travail

Voici un tableau détaillé de la répartition des tâches pendant le projet. Le projet aura duré environ 3 mois avec les phases suivantes :

1. Sélection des composants en fonction des contraintes imposées
2. Tests des composants sélectionnés
3. Développement du code
4. Montage du robot suiveur de ligne
5. Tests du robot et résolution des bugs
6. Rédaction du rapport

	Thomas Bauer	Angelo Bou-tanous	Ali Hamdani	Nathan Legendre
Sélection des composants	x	x	x	x
Test des composants :	x	x	x	x
Suiveur de ligne		x		x
Capteur ultrason		x		x
Active Buzzer		x		x
Écran LCD		x		x
Moteurs	x		x	
Conception du code :	x	x	x	x
Suiveur de ligne	x	x		
Capteur ultrason				x
Active Buzzer	x			
Écran LCD				x
Moteurs			x	
Manette de PS5	x			
main	x			
Fritzing	x			
Montage des composants		x		x
Conception 3D SolidWorks	x			
Montage final du robot	x	x	x	x
Tests	x	x	x	x
Rapport			x	x

FIGURE 1 – Répartition des tâches

2 Circuit électronique (Fritzing)

Après avoir choisi les composants à partir du cahier des charges, nous avons sélectionné les GPIO importants pour envoyer des données à chaque composants.

Afin de schématiser tout cela, nous avons utilisé le logiciel Fritzing qui nous a permis de relier les composants virtuellement assez rapidement.

Vous trouverez le schéma Fritzing ci-dessous.

3 Explication des choix techniques

Pour la sélection des composants, en plus des éléments imposés par le projet, nous avons décidé de concevoir un modèle 3D, d'opter pour l'utilisation d'une manette SONY PS5 et d'employer des câbles femelle-femelle.

Le choix de la manette s'est imposé en raison de sa précision supérieure par rapport à la télécommande. En effet, les deux gâchettes arrière émettent des valeurs comprises entre 0 et 255, tandis que les joysticks génèrent des valeurs entre 0 et 180. Couplées aux signaux PWM, ces caractéristiques nous ont offert une précision accrue tant au niveau de la vitesse que des manœuvres.

Quant au code de la manette, nous avons initialement utilisé la bibliothèque de bas niveau libevdev, spécifique à la manette PS5. Toutefois, nous avons ultérieurement opté pour la bibliothèque SDL 2.0, plus haut niveau, rendant ainsi notre code compatible avec toutes les manettes.

En ce qui concerne le choix du modèle 3D, il s'est principalement orienté vers des considérations esthétiques. Nous avons privilégié un modèle compact qui évoquerait une véritable voiture télécommandée. Cette orientation a guidé notre décision d'utiliser des câbles femelle-femelle, contribuant à compacter l'ensemble du dispositif.

4 Explication du rôle des composants

Pour répondre aux cahier des charger évoqué en introduction, nous avons choisi différents composants, parfois plusieurs fois les mêmes. Nous allons expliciter nos choix dans la section ci-dessous :

4.1 Raspberry Pi

La Raspberry Pi a été cruciale dans notre projet en tant que cerveau central du robot. Elle a pris en charge la gestion des capteurs, la logique de contrôle, et les interactions avec les composants matériels. Grâce à sa connectivité GPIO, elle a permis une intégration facile avec les capteurs de suivi de ligne, les capteurs de distance, et d'autres périphériques. De plus, la possibilité de programmer en langage C a facilité le développement du logiciel embarqué car nous suivons en parallèle un cours sur ce même language.

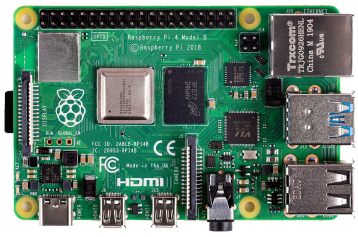


FIGURE 2 – Raspberry Pi 4

4.2 Capteur infrarouge TCRT5000

Les capteurs infrarouge, tels que le modèle TCRT5000 que nous avons choisi, jouent un rôle essentiel en tant que suiveurs de ligne dans notre robot. Leur principe de fonctionnement repose sur l'émission d'un faisceau infrarouge et la détection du signal réfléchi. Ce capteur est composé d'une LED infrarouge et d'un phototransistor, permettant de mesurer l'intensité du signal réfléchi. En suivant une ligne, le capteur détecte les variations d'intensité du signal infrarouge en fonction de la surface rencontrée, ce qui permet au robot de maintenir sa trajectoire ou non.

En effet, dans notre cas, nous avons utilisé 3 LineFinder : un à gauche, un central et un à droite. Cela permet de couvrir toute la zone nécessaire pour suivre la ligne, détecter les intersections et tourner efficacement.

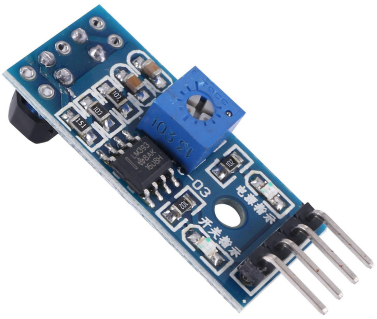


FIGURE 3 – Capteur infrarouge de suivi de ligne

Dans un premier temps, il faut savoir que nous avons travaillé avec un autre type de capteur infrarouge type "LineFinder". Cela n'a eu aucune conséquence sur la suite du projet car leur fonctionnement est quasiment identique aux capteurs finaux.

Le choix du capteur infrarouge s'aligne parfaitement avec les exigences du cahier des charges initial. La facilité d'intégration avec le Raspberry Pi, combinée à leur réactivité, offre une solution fiable pour le suivi de ligne, permettant au robot de naviguer de manière fluide tout en respectant les intersections définies dans le cahier des charges.

4.3 Moteurs et L293D

Les moteurs directement reliés aux roues de notre robot constituent le cœur de sa mobilité.

Pour assurer un contrôle précis et bidirectionnel de ces moteurs, nous avons choisi d'utiliser la carte de commande de moteur L293D. Cette carte offre une interface simple mais efficace entre le Raspberry Pi et les moteurs, permettant de réguler la vitesse et la direction de manière précise. Le L293D fonctionne en amplifiant les signaux de commande du Raspberry Pi pour fournir une alimentation adéquate aux moteurs, facilitant ainsi le contrôle des mouvements du robot.

La sélection du L293D s'inscrit parfaitement dans les exigences spécifiées dans notre cahier des charges initial. Sa capacité à gérer deux moteurs simultanément, à inverser la direction de rotation, et à ajuster la vitesse répond à notre besoin de contrôle moteur bidirectionnel pour le suivi de ligne et les manœuvres aux intersections.

À préciser qu'au début du projet, nous avons utilisé une carte MAKERDRIVE qui nous a posé beaucoup trop de problèmes à nous et aux autres groupes ce qui a justifié notree changement vers la L293D. (Cela sera détaillé dans la partie difficultés)

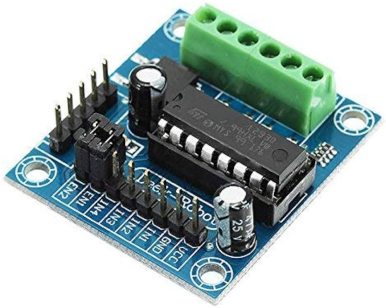


FIGURE 4 – Carte de contrôle des moteurs L293D

4.4 Capteur d'ultrasons HS-SR04

Les capteurs ultrasons, tels que le modèle HS-SR04 que nous avons intégré à notre robot, jouent un rôle crucial dans la détection des obstacles et la mesure de la distance frontale. Grâce à leur principe de fonctionnement, émettant des ondes sonores et mesurant le temps nécessaire à leur retour après réflexion sur un obstacle, ces capteurs fournissent une estimation précise de la distance entre le robot et tout objet présent sur sa trajectoire.

Le choix du capteur ultrasonique HS-SR04 découle directement des exigences spécifiées dans notre cahier des charges initial. Sa portée étendue et sa résolution précise permettent au robot de détecter les obstacles à des distances variées, contribuant ainsi à l'alerte précoce et à la prise de décision en temps réel. Cette fonctionnalité est essentielle pour respecter les critères du cahier des charges, notamment l'alerte en cas d'obstacle, l'arrêt d'urgence, et l'attente en présence d'obstacles.

FIGURE 5 – Capteur d'ultrason

4.5 Active buzzer

L'Active Buzzer, le composant d'alerte d'obstacle est essentiel pour notre robot suiveur de ligne. Conformément aux exigences énoncées dans notre cahier des charges initial, l'Active Buzzer est activé dès qu'un obstacle est détecté par les capteurs ultrasons. Cette alerte sonore, combinée à l'affichage sur l'écran LCD, permet d'assurer une notification immédiate de la proximité d'un objet. Cette fonctionnalité contribue directement à la sécurité du robot, avertissant les opérateurs et les personnes environnantes en cas de risque potentiel.

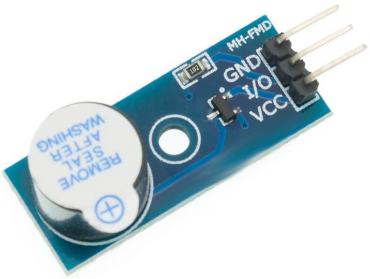


FIGURE 6 – Active Buzzer

4.6 LCD1602 et I2C Interface Module

L'écran LCD1602, contrôlé via l'interface I2C, représente un élément central dans notre robot suiveur de ligne en fournissant une interface visuelle pour afficher des informations cruciales conformes aux spécifications du cahier des charges initial. Sa capacité à présenter en temps réel les mouvements programmés du robot, les distances mesurées par les capteurs, ainsi que l'état de connexion de la manette, offre une visibilité immédiate sur le statut opérationnel du robot.

L'utilisation de l'I2C simplifie la connectivité entre l'écran LCD et le Raspberry Pi, permettant une intégration aisée dans notre système embarqué.

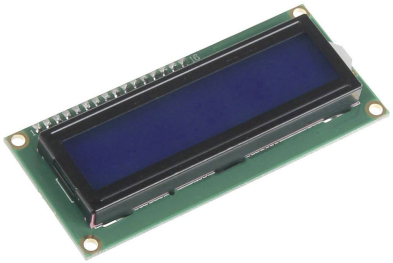


FIGURE 7 – Ecran LCD 1602

4.7 Manette de PS5

La manette de PS5 connectée en Bluetooth, avec sa configuration ergonomique et ses fonctionnalités avancées, offre une expérience de contrôle fluide et réactive. Les opérateurs peuvent ainsi ajuster la trajectoire du robot, effectuer des arrêts d'urgence, et contrôler manuellement les mouvements, répondant ainsi aux critères de contrôlabilité et d'interactivité définis dans le cahier des charges. En choisissant cette approche, notre robot suiveur de ligne intègre une solution de commande manuelle moderne et accessible, alignée sur les attentes spécifiées dès le début de notre projet.

FIGURE 8 – Manette de PS5

5 Code

Afin de développer le code, nous nous sommes inspirés des séances de travaux pratiques précédentes. Notamment, en ce qui concerne l'écran LCD et l'I2C, le buzzer et l'utilisation des moteurs. Pour le reste nous avons fait des recherches.

Vous trouverez ci-dessous le programme principal ainsi que les différents programmes annexes :

— Programme principal

```
#include "controller.h"
#include "i2cLCD.h"
#include "gpioPins.h"
#include "lineFinder.h"
#include "motors.h"
#include "buzzer.h"
#include "distance.h"
#include "speaker.h"

#include <SDL2/SDL.h>
#include <wiringPi.h>
#include <lcd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>

#define MODE_MANUAL     0
#define MODE_LINEFINDER 1

#define FORWARD         0
#define TURN_LEFT       1
#define TURN_RIGHT      2

int lcd, mode, motorState, R2, L2, LX;
bool controllerConnected, nearObstacle, exitSDL;

PI_THREAD(lcdPrintAndGetDistance) {
    while (!exitSDL) {
        lcdClear(lcd);
        if (controllerConnected) {
```

```
            if (mode == MODE_LINEFINDER) {
                int distance = getDistance();
                if (distance <= DISPLAY_DISTANCE) {
                    if (distance <= STOP_DISTANCE) {
                        lcdPrintf(lcd,"Obstacle too    close!");
                        nearObstacle = 1;
                    }
                    else {
                        lcdPrintf(lcd,"Obstacle : %d cm",distance);
                        nearObstacle = 0;
                    }
                }
                else {
                    switch(motorState) {
                    case FORWARD:
                        lcdPrintf(lcd,"Moving forward");
                        break;
                    case TURN_LEFT:
                        lcdPrintf(lcd,"Truning left");
                        break;
                    case TURN_RIGHT:
                        lcdPrintf(lcd,"Turning right");
                        break;
                    }
                    nearObstacle = 0;
                }
            }
            else if (mode == MODE_MANUAL) {
                lcdPrintf(lcd,"Speed : %d kph",abs((int)((R2-L2)/252)));
            }
        }
        else {
            lcdPrintf(lcd,"Waiting for     controller...");
        }
        delay(100);
    }
    return 0;
}
```

```
void lineFinder(int lcd){
    bool gauche = detecterLigne(PIN_SUIVEUR_GAUCHE);
    bool centre = detecterLigne(PIN_SUIVEUR_CENTRE);
    bool droite = detecterLigne(PIN_SUIVEUR_DROIT);
    printf("%d %d %d\n",gauche,centre,droite);

    if (nearObstacle) {
        stopMotors();
        buzzerOn();
    }
    else {
        if (!gauche && !centre && !droite) {
            switch(motorState) {
                case FORWARD:
                    LF_forward();
                    break;
                case TURN_LEFT:
                    LF_turnLeft();
                    break;
                case TURN_RIGHT:
                    LF_turnRight();
                    break;
            }
        }
        else if (detecterIntersection(gauche,centre,droite) || centre) {
            LF_forward();
            motorState = FORWARD;
        }
        else if (gauche) {
            LF_turnLeft();
            motorState = TURN_LEFT;
        }
        else if (droite) {
            LF_turnRight();
            motorState = TURN_RIGHT;
        }
        buzzerOff();
    }
}
```

```
void manualControl(int lcd, SDL_GameController *controller) {
    R2 = triggerValue(controller,SDL_CONTROLLER_AXIS_TRIGGERRIGHT);
    L2 = triggerValue(controller,SDL_CONTROLLER_AXIS_TRIGGERLEFT);
    LX = axisValue(controller,SDL_CONTROLLER_AXIS_LEFTX);
    if (R2 >= L2) {
        forward(R2-L2,LX);
    }
    else {
        backward(L2-R2,LX);
    }
}

int main(int argc, char* argv[]) {
    // WiringPi Initialization
    wiringPiSetupGpio();

    // LCD Initialization
    lcd = initLCD();

    // Controller Initialization
    SDL_GameController *controller = initController();

    // Motors Initialization
    initMotors();

    // Line-Finder Initialization
    initSuiveurLigne();

    // Line-Finder Initialization
    initDistanceSensor();

    // Speaker Initialization
    initSpeaker();

    // Buzzer Initialization
    initBuzzer();
    buzzerOff();
```

```
// LCD and Distance Sensor Thread Creation
piThreadCreate(lcdPrintAndGetDistance);

// Boucle principale
SDL_Event event;
exitSDL = 0;
controllerConnected = 0;
mode = MODE_MANUAL;
nearObstacle = 0;
motorState = FORWARD;
while (!exitSDL) {
    SDL_PollEvent(&event);
    if (event.type == SDL_QUIT)
        exitSDL = 1;
    else if (event.cdevice.type == SDL_CONTROLLERDEVICEADDED) {
        controller = SDL_GameControllerOpen(0);
        controllerConnected = 1;
    }
    else if (event.cdevice.type == SDL_CONTROLLERDEVICEREMOVED) {
        SDL_GameControllerClose(controller);
        stopMotors();
        mode = MODE_MANUAL;
        buzzerOff();
        controllerConnected = 0;
    }
    else if (controllerConnected) {
        if (buttonIsBeingPressed(controller,SDL_CONTROLLER_BUTTON_A)) // Press
            mode = MODE_LINEFINDER;
        else if (buttonIsBeingPressed(controller,SDL_CONTROLLER_BUTTON_B)) { /
            buzzerOff();
            mode = MODE_MANUAL;
            motorState = FORWARD;
        }

        if (mode == MODE_LINEFINDER)
            lineFinder(lcd);
        else if (mode == MODE_MANUAL)
            manualControl(lcd,controller);
```

```
            playAudio(event,argv[0]);
        }
    }

    buzzerOff();
    SDL_Quit();

    return 0;
}
```

— Défintion de tous les GPIO

```
#ifndef __GPIO_PIN__
#define __GPIO_PIN__


#define PIN_BUZZER          14

#define PIN_SPEAKER         0

#define PIN_SUIVEUR_GAUCHE  17
#define PIN_SUIVEUR_CENTRE  22
#define PIN_SUIVEUR_DROIT   10

#define PIN_TRIG            25
#define PIN_ECHO            8

#define PIN_EN1             13
#define PIN_EN2             12

#define PIN_M1A             5
#define PIN_M1B             6
#define PIN_M2A             26
#define PIN_M2B             21


#endif
```

— Gestion du buzzer

```
#include "buzzer.h"
#include "gpioPins.h"
```

```
#include <wiringPi.h>

void initBuzzer() {
    pinMode(PIN_BUZZER, OUTPUT);
}

void buzzerOn() {
    digitalWrite(PIN_BUZZER, HIGH);
}

void buzzerOff() {
    digitalWrite(PIN_BUZZER, LOW);
}
```

— Gestion de la manette

```
#include "controller.h"

#include <stdio.h>
#include <stdbool.h>
#include <SDL2/SDL.h>

SDL_GameController* initController() {
    SDL_Init(SDL_INIT_GAMECONTROLLER);
    return NULL;
}

bool buttonIsPressed(int BUTTON, SDL_Event event) {
    return event.type == SDL_CONTROLLERBUTTONDOWN && event.cbutton.button == BUTTO
}

bool buttonIsReleased(int BUTTON, SDL_Event event) {
    return event.type == SDL_CONTROLLERBUTTONUP && event.cbutton.button == BUTTON;
}

bool buttonIsBeingPressed(SDL_GameController *controller, SDL_GameControllerButton
    return SDL_GameControllerGetButton(controller,BUTTON);
}
```

```
int triggerValue(SDL_GameController *controller, SDL_GameControllerAxis TRIGGER) {
    return SDL_GameControllerGetAxis(controller,TRIGGER);
}

int axisValue(SDL_GameController *controller, SDL_GameControllerAxis AXIS) {
    int val = SDL_GameControllerGetAxis(controller,AXIS);
    if (val <= DEADZONE*MIN_AXIS || val >= DEADZONE*MAX_AXIS)
        return val;
    return 0;
}
```

— Gestion de la distance

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>
#include "distance.h"
#include "gpioPins.h"

void initDistanceSensor() {
    pinMode(PIN_TRIG, OUTPUT);
    pinMode(PIN_ECHO, INPUT);
}

int getDistance() {
        struct timeval tv1;
        struct timeval tv2;
        long start, stop;
        float dis;

        digitalWrite(PIN_TRIG, LOW);
        delayMicroseconds(2);

        digitalWrite(PIN_TRIG, HIGH);  //produce a pluse
        delayMicroseconds(10);
        digitalWrite(PIN_TRIG, LOW);

        while(!(digitalRead(PIN_ECHO) == 1));
        gettimeofday(&tv1, NULL);           //current time
```

```
        while(!(digitalRead(PIN_ECHO) == 0));
        gettimeofday(&tv2, NULL);           //current time

        start = tv1.tv_sec * 1000000 + tv1.tv_usec;
        stop  = tv2.tv_sec * 1000000 + tv2.tv_usec;

        dis = (float)(stop - start) / 1000000 * 34000 / 2;  //count the distance

        return (int)dis;
}
```

— Gestion de l'écran LCD

```
#include "i2cLCD.h"

#include <wiringPi.h>
#include <pcf8574.h>
#include <lcd.h>
#include <stdio.h>
#include <stdlib.h>

int initLCD() {

    int i;

    pcf8574Setup(AF_BASE,I2C_ADDRESS);

    int lcd = lcdInit(2, 16, 4, AF_RS, AF_E, AF_DB4,AF_DB5,AF_DB6,AF_DB7, 0,0,0,0)

    if (lcd < 0) {
        fprintf (stderr, "lcdInit failed\n") ;
        exit (EXIT_FAILURE) ;
    }

    for(i=0;i<8;i++)
          pinMode(AF_BASE+i,OUTPUT);
    digitalWrite(AF_LED,1);
    digitalWrite(AF_RW,0);

    return lcd;
```

```
}
```

— Gestion des suiveurs de ligne

```
#include "lineFinder.h"
#include "motors.h"
#include "i2cLCD.h"
#include "buzzer.h"
#include "gpioPins.h"

#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

void initSuiveurLigne() {
    pinMode(PIN_SUIVEUR_GAUCHE, INPUT);
    pinMode(PIN_SUIVEUR_CENTRE, INPUT);
    pinMode(PIN_SUIVEUR_DROIT, INPUT);
}

bool detecterLigne(int pin_capteur) {
    return digitalRead(pin_capteur);
}

bool detecterIntersection(bool gauche, bool centre, bool droite){
    return ((gauche && centre && droite) || (gauche && droite) || (gauche && centr
}

void LF_forward() {
    forward(LF_SPEED,0);
}

void LF_turnLeft() {
    forward(LF_SPEED_ROTATION,MIN_AXIS);
}

void LF_turnRight() {
    forward(LF_SPEED_ROTATION,MAX_AXIS);
```

```
}
```

— Gestion des moteurs

```
#include "motors.h"
#include "gpioPins.h"
#include "controller.h"

#include <math.h>
#include <wiringPi.h>

// Initialisation de la bibliothèque WiringPi
void initMotors() {
    pinMode(PIN_M1A,OUTPUT);
    pinMode(PIN_M1B,OUTPUT);
    pinMode(PIN_M2A,OUTPUT);
    pinMode(PIN_M2B,OUTPUT);
    pinMode(PIN_EN1,PWM_OUTPUT);
    pinMode(PIN_EN2,PWM_OUTPUT);

    pwmSetMode(PWM_MODE_MS);
    pwmSetClock(PWM_CLOCK);     // Réglage de la fréquence PWM
    pwmSetRange(PWM_RANGE);    // Réglage de la plage PWM (0-1024)
}

// Fonction pour faire avancer le moteur
void forward(int speed, int angle) {
    digitalWrite(PIN_M1A,HIGH);
    digitalWrite(PIN_M2A,HIGH);
    digitalWrite(PIN_M1B,LOW);
    digitalWrite(PIN_M2B,LOW);

    float coeffSpeed = (float)speed/MAX_TRIGGER;
    if (angle <= 0) {
        float coeffAngle = (float)(MIN_AXIS-angle)/MIN_AXIS;
        pwmWrite(PIN_EN1,(int)round(coeffAngle*coeffSpeed*PWM_RANGE));
        pwmWrite(PIN_EN2,(int)round(coeffSpeed*PWM_RANGE));
    }
    else {
        float coeffAngle = (float)(MAX_AXIS-angle)/MAX_AXIS;
```

```
        pwmWrite(PIN_EN1,(int)round(coeffSpeed*PWM_RANGE));
        pwmWrite(PIN_EN2,(int)round(coeffAngle*coeffSpeed*PWM_RANGE));
    }
}

// Fonction pour faire reculer le moteur
void backward(int speed, int angle) {
    digitalWrite(PIN_M1A,LOW);
    digitalWrite(PIN_M2A,LOW);
    digitalWrite(PIN_M1B,HIGH);
    digitalWrite(PIN_M2B,HIGH);

    float coeffSpeed = (float)speed/MAX_TRIGGER;
    if (angle <= 0) {
        float coeffAngle = (float)(MIN_AXIS-angle)/MIN_AXIS;
        pwmWrite(PIN_EN1,(int)round(coeffAngle*coeffSpeed*PWM_RANGE));
        pwmWrite(PIN_EN2,(int)round(coeffSpeed*PWM_RANGE));
    }
    else {
        float coeffAngle = (float)(MAX_AXIS-angle)/MAX_AXIS;
        pwmWrite(PIN_EN1,(int)round(coeffSpeed*PWM_RANGE));
        pwmWrite(PIN_EN2,(int)round(coeffAngle*coeffSpeed*PWM_RANGE));
    }
}

// Fonction pour arrêter le moteur
void stopMotors() {
    pwmWrite(PIN_EN1,0);
    pwmWrite(PIN_EN2,0);
}
```

— Gestion du haut-parleur

```
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#include <wiringPi.h>
#include <SDL2/SDL.h>

#include "gpioPins.h"
```

```
void initSpeaker() {
    pinMode(PIN_SPEAKER, OUTPUT);
}

void playAudio(SDL_Event event, char argv0[]) {
    char *audios[] = {"audio1.mp3","audio2.mp3","audio3.mp3","audio4.mp3"};
    int audioIndex = -1;
    if (event.cbutton.type == SDL_CONTROLLERBUTTONDOWN) {
        switch (event.cbutton.button) {
            case SDL_CONTROLLER_BUTTON_DPAD_UP:
                audioIndex = 0;
                break;
            case SDL_CONTROLLER_BUTTON_DPAD_DOWN:
                audioIndex = 1;
                break;
            case SDL_CONTROLLER_BUTTON_DPAD_LEFT:
                audioIndex = 2;
                break;
            case SDL_CONTROLLER_BUTTON_DPAD_RIGHT:
                audioIndex = 3;
                break;
        }
        if (audioIndex != -1){
            digitalWrite(PIN_SPEAKER, HIGH);

            char command[200], programPath[100];
            realpath(argv0, programPath);
            sprintf(command, "mpg123 %s/../audio/%s &", dirname(programPath), audi
            system(command); // permet de lancer l'execution d'une commande

            digitalWrite(PIN_SPEAKER, LOW);
        }
    }
}
```

6 Difficultés rencontrées

Durant notre projet nous avons rencontré certaines difficultés. La première a été la conception des codes moteurs, le premier composant que nous avions utilisés le MAKERDRIVE était difficile à manipuler et nous n'arrivions pas a envoyé des signaux PWM de valeurs différents. Problème qui n'était pas présent avec le L293D.

La deuxième difficulté a été la réalisation de notre modèle 3D, nous devions pour celui-ci prévoir à l'avance tous les trous pour le fixer et faire passer les fils tout en gardant un résultat esthétique. De plus la limite temps imposé par l'impression 3D nous a poussé à trouver le bon compromis entre un résultat compacte, esthétique et qui nous permette d'y disposer tous les composants.

La troisième difficulté est lié à la deuxième, la limite de temps d'impression et le modèle 3D compact nous a poussé à devoir optimiser l'espace occupé par les composants, en utilisant des câbles femelle-femelle, ressoudant les branches de l'I2C, raccourcir nos cables etc.

Pour finir nous avons trouver une solution à chacun de nos problèmes grâce aux idées de chacun et des conseils des encadrants de TP, ce qui nous a permis d'arriver à un résultat convenable et fonctionnel.

7 Conclusion

Pour conclure, ce projet de robot suiveur de ligne nous a permis de mettre en pratique nos connaissances acquises en cours d'Electronique notamment sur l'utilisation des composants et du Raspberry Pi. Nous avons également utilisé nos connaissances développées dans le cours d'Algorithme et Programmation en C afin de réaliser le code pour contrôler tous nos composants.

Même si au premier abord, ce projet peut paraître simple, la tâche n'a pas été aussi simple que prévu. En effet, les différentes contraintes ajoutées (contrôler le robot avec une manette de PlayStation 5, créer une coque en 3D) nous ont créer des problèmes que nous avons su résoudre non sans mal.

Évidemment, le résultat n'est jamais parfait mais nous sommes satisfaits du travail rendu et des compétences acquises lors de ce projet.

7.1 Problèmes à corriger

7.2 Améliorations