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 - 2023-2024

# Table des matières

# 1 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 quatre : 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 doit suivre une ligne noire, et donc qui selon la couleur perçue (noir ou blanc) 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 émises de façon sonore. Nous avons décidé d'ajouter au robot la possibilité d'être contrôlé via une manette.

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 déterminés nous avons pensé 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

# 2 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 ultrasons | | 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 ultrasons | | | | x |
| Active Buzzer | | | x | |
| Écran LCD | | | | x |
| Moteurs | | | x | |
| Manette | x | | | |
| Main | x | | | |
| | | | | |
| **Fritzing** | x | | | |
| **Montage des composants** | | x | | x |
| **Conception 3D SolidWorks** | x | | | |
| **Montage final du robot** | x | x | x | x |
| **Tests et résolution des bugs** | x | x | x | x |
| **Rapport** | x | x | x | x |

FIGURE 1 – Répartition des tâches

# 3 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 composant.

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.

Figure 2 – Schéma avec les composants Fritzing

# 3 CIRCUIT ÉLECTRONIQUE (FRITZING)

FIGURE 3 – Schéma électronique Fritzing

# 4 Explication des choix techniques

Pour la sélection des composants, en plus des éléments imposés par le projet, nous avons décidé d'opter d'utiliser une manette et de concevoir un modèle 3D de voiture.

## 4.1 Choix de la manette

Le choix de la manette s'est imposé en raison de sa précision supérieure par rapport à la télécommande. En effet, les deux gâchettes arrières ainsi que les joysticks émettent des valeurs comprises entre 0 et 255. 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 evdev, qui utilise directement les fichiers en mode d'accès caractère associés à la manette. Toutefois, nous avons ultérieurement opté pour la bibliothèque SDL 2.0, plus haut niveau, rendant ainsi notre robot compatible avec toutes les manettes.

## 4.2 Modèle 3D

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. La base du modèle est une voiture issue du jeu vidéo Rocket League, qui a pour caractéristiques d'être haute et large, rendant le montage des composants internes moins laborieux.

Pour concevoir ce modèle 3D, nous avons modifié un fichier .stl préexistant de la voiture en utilisant le logiciel SolidWorks. Les roues ont été prises comme point de départ pour déterminer les dimensions globales de la voiture, garantissant ainsi une cohérence visuelle avec les garde-boue et les roues fictives de devant.

En ce qui concerne les composants visibles, tels que le capteur de distance et l'écran LCD, nous avons créé des logements spécifiques dans la carrosserie. Ces logements ont été conçus pour offrir une intégration fluide et discrète, assurant que ces composants s'intègrent naturellement à la conception extérieure de la voiture. Des ajustements minutieux ont été apportés pour garantir la précision et la stabilité du montage, tout en préservant l'esthétique originale du modèle.

Simultanément, nous avons essayé d'enlever le maximum de matière à l'intérieur du modèle afin d'y intégrer les composants internes et de diminuer le temps d'impression. Des supports spécifiques ont été ajoutés pour maintenir en place l'ensemble des éléments électroniques tout en facilitant l'accès pour d'éventuelles réparations ou modifications.

Après avoir effectué ces modifications et quelques vérifications sur les dimensions des composants, nous avons envoyé le fichier .stl modifié pour l'impression 3D du modèle final, aboutissant à une voiture télécommandée compacte, esthétique et fonctionnelle.

FIGURE 4 – Captures d'écran de la conception 3D sous SolidWorks

# 5 Explication du rôle des composants

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

## 5.1 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 5 – Raspberry Pi 4

## 5.2 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 trois suiveurs de ligne : 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.

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.

FIGURE 6 – Capteur infrarouge de suivi de ligne TCRT5000

## 5.3 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 moteurs 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 à d'autres groupes ce qui a justifié notre changement vers la L293D. (Cela sera détaillé dans la partie 6. Difficultés rencontrées)

FIGURE 7 – Carte de contrôle des moteurs L293D

## 5.4 Capteur 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 8 – Capteur ultrasons HS-SR04

## 5.5 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.

Figure 9 – Active Buzzer

## 5.6 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.

## 5.7 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.

À noter que nous avons utilisé une manette de PS5 mais nous aurions pu utiliser quasiment n'importe quelle manette similaire. (Ceci a été détaillé plus haut dans la section 4.)

Figure 10 – Ecran LCD 1602

Figure 11 – Manette de PS5

# 6 Difficultés rencontrées

Lors de la réalisation de notre projet, nous avons rencontré quelques difficultés qui nous ont ralenties mais sans jamais nous arrêter.

## 6.1 Gestion des moteurs

La première a été la conception des codes moteurs. En effet, le premier composant que nous avons utilisé, le MAKERDRIVE, était difficile à manipuler et nous n'arrivions pas a envoyé des signaux PWM de valeurs différentes. Problème qui n'était pas présent avec le L293D sur lequel nous avons continué le projet.

## 6.2 Fonctionnement du Code

Durant la phase de développement nous avons rencontré certains problèmes.

L'un des problèmes était une latence entre l'action réalisée sur la manette et la réaction du robot, rendant très difficile sa maniabilité ainsi que certaines manoeuvres. Nous avons alors entâmé des recherches pour trouver une solution à ce problème. Après plusieures tentatives qui n'aboutissaient pas, la solution était plutôt simple. Pour un signal PWM nous avons une clock qui met à jour le signal PWM à chaque période de temps définit par la clock, cette latence ne venait donc ni de la manette ni de la Raspeberry Pi mais bien de la clock PWM, et en la diminuant nous avons réussi à obtenir une maniabilité très bonne avec un réponse presque instantanée du robot.

Un autre problème rencontré pendant notre phase de développement, a été le fonctionnement du mode suivi de ligne. Pour celui-ci nous avions tester le fonctionnement en dehors du circuit final et il fonctionnait correctement, notre table de vérité était correcte mais sur le circuit notre robot effectuait des sortes de zig-zags sur les lignes droites et avait certains problèmes sur les virages. Après avoir revu notre code et notre table de vérité plusieurs fois, nous en avons conclu que le problème ne venait pas d'ici. Nous avons alors observé le comportement du robot sur le circuit et avons remarqué que la vitesse du robot comparée à la taille du circuit impactait le fonctionnement de nos suiveurs de ligne. Nous avons alors réduit la vitesse globale en mode suivi de ligne jusqu'à trouver le parfait compromis entre vitesse et réalisation du circuit.

## 6.3 Modélisation 3D

La troisième difficulté majeure rencontrée lors de la réalisation de notre modèle 3D a été liée aux contraintes spécifiques du fichier .stl que nous avons choisi pour l'impression. Le recours à un fichier maillé comportant plus de 50 000 facettes a rendu l'utilisation de la fonction 'Coque' (qui permet de créer une enveloppe solide autour d'un modèle plein) de SolidWorks impossible.

Face à ce défi, nous avons dû entreprendre manuellement le processus de "creusement" en effectuant des enlèvements de matière, une tâche complexe et minutieuse.

En raison du nombre élevé de facettes, le logiciel s'est révélé extrêmement lent, nécessitant plusieurs minutes pour appliquer des fonctions telles que l'enlèvement de matière ou l'extrudage. Cette contrainte a ajouté une complexité significative à la modélisation, exigeant une attention particulière à chaque étape du processus pour garantir la précision et l'intégrité du modèle.

Une autre difficulté notable a émergé lors de la phase d'impression 3D. En raison du temps d'impression trop élevé dû aux supports nécessaires, nous avons été contraints de découper le modèle en trois parties distinctes. Cette démarche visait à optimiser l'utilisation des supports afin de diminuer le temps d'impression, tout en conservant l'esthétique du modèle final.

Par ailleurs, la réalisation du modèle 3D a exigé une anticipation minutieuse de l'emplacement des trous pour la fixation et le passage des fils, nous obligeant à trouver le juste équilibre entre compacité, esthétique et disposition efficace de tous les composants.

## 6.4 Taille de notre robot

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

## 6.5 Isolation des composants

Après le montage de notre robot, nous avons entamé la phase finale, c'est-à-dire les tests.

Certains tests qui pourtant réussissaient avant le montage ne réussissaient plus. Nous avons alors décidé de démonter le robot afin de trouver la raison de ces disfonctionnements. C'est ici, que nous avons remarqué qu'il y avait certains faux-contacts et courts-circuits dûs à la proximité de nos composants, affectant donc leur bon fonctionnement. Pour y remédier nous avons isolé chaque composant un à un et vérifier si tout était à nouveau fonctionnel.

Pour finir nous avons trouvé 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'Électronique 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, créer une coque en 3D, etc) nous ont créé 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 Améliorations

Même si nous sommes satisfaits de notre robot suiveur de ligne, il est à noter que nous aurions aimé modifier deux choses sur ce projet mais qu'il était un peu tard pour le faire.

En effet, il aurait pu être intéressant de rendre les moteurs plus puissants en optimisant l'espace à l'intérieur de la coque 3D. Cela nous aurait permis d'utiliser deux petites batteries pour mieux alimenter les moteurs et donc leur permettre de développer plus de puissance en sortie.

Par ailleurs, l'intégration d'un module sonore, tel qu'un haut-parleur, a été envisagée pour enrichir l'expérience pendant le match de foot entre les voitures. Cependant, après une analyse approfondie de la documentation et des essais de codage, nous avons identifié des défis imprévus. La nécessité d'un composant lecteur MP3, occupant un espace supplémentaire, s'est avérée contraignante compte tenu des limitations d'espace dans notre voiture. Malgré nos efforts, la complexité d'intégration a rendu cette fonctionnalité plus difficile à implémenter dans le cadre du projet actuel. Néanmoins, si le temps le permettait, l'ajout d'un haut-parleur aurait offert la possibilité d'intégrer des commentaires sonores et des effets sonores, rehaussant ainsi l'aspect ludique du match.

# 8 Code

## 8.1 Programme principal

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

// Global variables used to communicate between processes
int lcd, mode, motorState, R2, L2, LX;
bool controllerConnected, nearObstacle, exitSDL;

// LCD display and distance acquisition process
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;
}

// Direction selection function in line-finding mode
void lineFinder(int lcd){
    bool left = detectLine(PIN_LINEFINDER_LEFT);
```

```
    bool center = detectLine(PIN_LINEFINDER_CENTER);
    bool right = detectLine(PIN_LINEFINDER_RIGHT);
    printf("%d %d %d\n",left,center,right);

    if (nearObstacle) {
        stopMotors();
        buzzerOn();
    }
    else {
        if (!left && !center && !right) {
            switch(motorState) {
                case FORWARD:
                    LF_forward();
                    break;
                case TURN_LEFT:
                    LF_turnLeft();
                    break;
                case TURN_RIGHT:
                    LF_turnRight();
                    break;
            }
        }
        else if (detectIntersection(left,center,right) || center) {
            LF_forward();
            motorState = FORWARD;
        }
        else if (left) {
            LF_turnLeft();
            motorState = TURN_LEFT;
        }
        else if (right) {
            LF_turnRight();
            motorState = TURN_RIGHT;
        }
        buzzerOff();
    }
}

// Robot control function in manual-control mode
```

```
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);
    }
}

// Main function
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
    initLineFinder();

    // Line-Finder Initialization
    initDistanceSensor();

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

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

```
// Main loop
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;
    if (event.cdevice.type == SDL_CONTROLLERDEVICEADDED &&
    ↪  !controllerConnected) {
        controller = SDL_GameControllerOpen(0);
        controllerConnected = 1;
    }
    if (event.cdevice.type == SDL_CONTROLLERDEVICEREMOVED &&
    ↪  controllerConnected) {
        SDL_GameControllerClose(controller);
        stopMotors();
        mode = MODE_MANUAL;
        buzzerOff();
        controllerConnected = 0;
    }
    if (controllerConnected) {
        if (buttonIsBeingPressed(controller,SDL_CONTROLLER_BUTTON_A)) //
        ↪  Press CROSS to enter Line-Finder Mode
            mode = MODE_LINEFINDER;
        else if (buttonIsBeingPressed(controller,SDL_CONTROLLER_BUTTON_B))
        ↪  { // Press CIRCLE to leave Line-Finder Mode
            buzzerOff();
            mode = MODE_MANUAL;
            motorState = FORWARD;
        }

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

```
        }
    }

    buzzerOff();
    SDL_Quit();

    return 0;
}
```

## 8.2 Défintion de tous les GPIO

```
#ifndef __GPIO_PIN__
#define __GPIO_PIN__


#define PIN_BUZZER              14

#define PIN_LINEFINDER_LEFT     17
#define PIN_LINEFINDER_CENTER   22
#define PIN_LINEFINDER_RIGHT    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
```

## 8.3 Gestion du buzzer

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

```
#include <wiringPi.h>

// Buzzer initialization
void initBuzzer() {
    pinMode(PIN_BUZZER, OUTPUT);
}

// Function to turn on the buzzer
void buzzerOn() {
    digitalWrite(PIN_BUZZER, HIGH);
}

// Function to turn off the buzzer
void buzzerOff() {
    digitalWrite(PIN_BUZZER, LOW);
}
```

## 8.4 Gestion de la manette

```
#include "controller.h"

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

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

// Function to check if a specific button is pressed
bool buttonIsPressed(int BUTTON, SDL_Event event) {
    return event.type == SDL_CONTROLLERBUTTONDOWN && event.cbutton.button ==
     ↪  BUTTON;
}

// Function to check if a specific button is released
```

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

// Function to check if a specific button is currently being pressed on the
↪  controller
bool buttonIsBeingPressed(SDL_GameController *controller,
↪  SDL_GameControllerButton BUTTON) {
    return SDL_GameControllerGetButton(controller,BUTTON);
}

// Function to get the value of a specific trigger on the controller
int triggerValue(SDL_GameController *controller, SDL_GameControllerAxis
↪  TRIGGER) {
    return SDL_GameControllerGetAxis(controller,TRIGGER);
}

// Function to get the value of a specific axis on the controller with a
↪  deadzone check
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;
}
```

## 8.5 Gestion de la distance

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


// Distance sensor initialization
void initDistanceSensor() {
    pinMode(PIN_TRIG, OUTPUT);
```

```
    pinMode(PIN_ECHO, INPUT);
}

// Function to acquire the distance through the ultrasonic sensor
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;
}
```

## 8.6 Gestion de l'écran LCD

```
#include "i2cLCD.h"

#include <wiringPi.h>
#include <pcf8574.h>
#include <lcd.h>
```

```
#include <stdio.h>
#include <stdlib.h>

// LCD initialization
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;

}
```

## 8.7 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>
```

```
// Line finder sensors initialization
void initLineFinder() {
    pinMode(PIN_LINEFINDER_LEFT, INPUT);
    pinMode(PIN_LINEFINDER_CENTER, INPUT);
    pinMode(PIN_LINEFINDER_RIGHT, INPUT);
}

// Function to check the state of a specific line finder sensor
bool detectLine(int pin_capteur) {
    return digitalRead(pin_capteur);
}

// Function to check if the sensors are detecting any kind of intersection
bool detectIntersection(bool left, bool center, bool right){
    return ((left && center && right) || (left && right) || (left && center)
     ↪  || (center && right));
}

// Function to make the robot move forward in line-finding mode
void LF_forward() {
    forward(LF_SPEED,0);
}

// Function to make the robot turn left in line-finding mode
void LF_turnLeft() {
    forward(LF_SPEED_ROTATION,MIN_AXIS);
}

// Function to make the robot turn right in line-finding mode
void LF_turnRight() {
    forward(LF_SPEED_ROTATION,MAX_AXIS);
}
```

## 8.8 Gestion des moteurs

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

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

// Motors initialization
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);
    pwmSetRange(PWM_RANGE);
}

// Function to make the robot move forward with a specified speed and steering
↪ angle
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));
    }
}
```

```
// Function to make the robot move backward with a specified speed and
↪  steering angle
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));
    }
}

// Function to stop the motors
void stopMotors() {
    pwmWrite(PIN_EN1,0);
    pwmWrite(PIN_EN2,0);
}
```

# Table des figures