Mechatronic project : Report

LINE TRACKER ROBOT

Group Member :

- MARÇAL Thomas
- KOSKAS Axel
- GOSSELIN Julie

Table of contents :

1. Project description

Initial Project:

The line follower robot project implemented with an Arduino involves the design and development of a robot capable of following a line traced on a surface using sensors and an Arduino.

The robot was built using a robot car kit, which consists of a car equipped with four motorized wheels and a chassis. The sensors used in the project are infrared sensors installed at the front of the robot, facing the surface of the line and positioned close to the ground. These sensors detect the contrasting color of the line against the surrounding surface. They send a value below 40 when they detect a dark area and a value above 1000 when they detect a light area. This allows us to determine if the sensors are located on the line or not, and to follow the line based on the activated sensors.

With the Arduino MEGA, we can control the motors and read the data from the line sensors. The Arduino program is designed to adjust the direction of the motors based on the detection of the line. If the robot goes off the line, it adjusts its motors to return to the line.

Added features for the project:

- **Obstacle detection**

To improve the functionality of the Line Tracker Robot, we added the ability to detect and avoid obstacles. The robot is equipped with an ultrasonic sensor that measures the distance between the robot and an object in front of it. When the sensor detects an object at a distance below a certain threshold, the robot switches to obstacle avoidance mode.

When the robot is in obstacle avoidance mode, it uses a combination of rotation and forward movement to navigate around the object. The robot rotates to the right to avoid the object, then moves forward until it can resume following the line traced on the surface.

The obstacle avoidance mode is only activated when the object is detected on the line that the robot is following. If the object is nearby but not on the line, the robot simply continues to follow the line. This obstacle avoidance functionality allows the robot to follow the line on the surface without being blocked by obstacles in its path. It also makes the robot more autonomous and capable of navigating in more complex environments. Additionally, the combination of line detection and obstacle avoidance offers greater potential for the robot.

- **Possibility to control the robot via Bluetooth and Python**

To enhance the functionality of the Line Tracker Robot, we introduced a manual control mode. This feature allows the robot to switch from the automated line tracking mode to a manual control mode of the robot's direction. This development highlights the robot's ability to transition from autonomous to manual control, similar to what is found in modern vehicles.

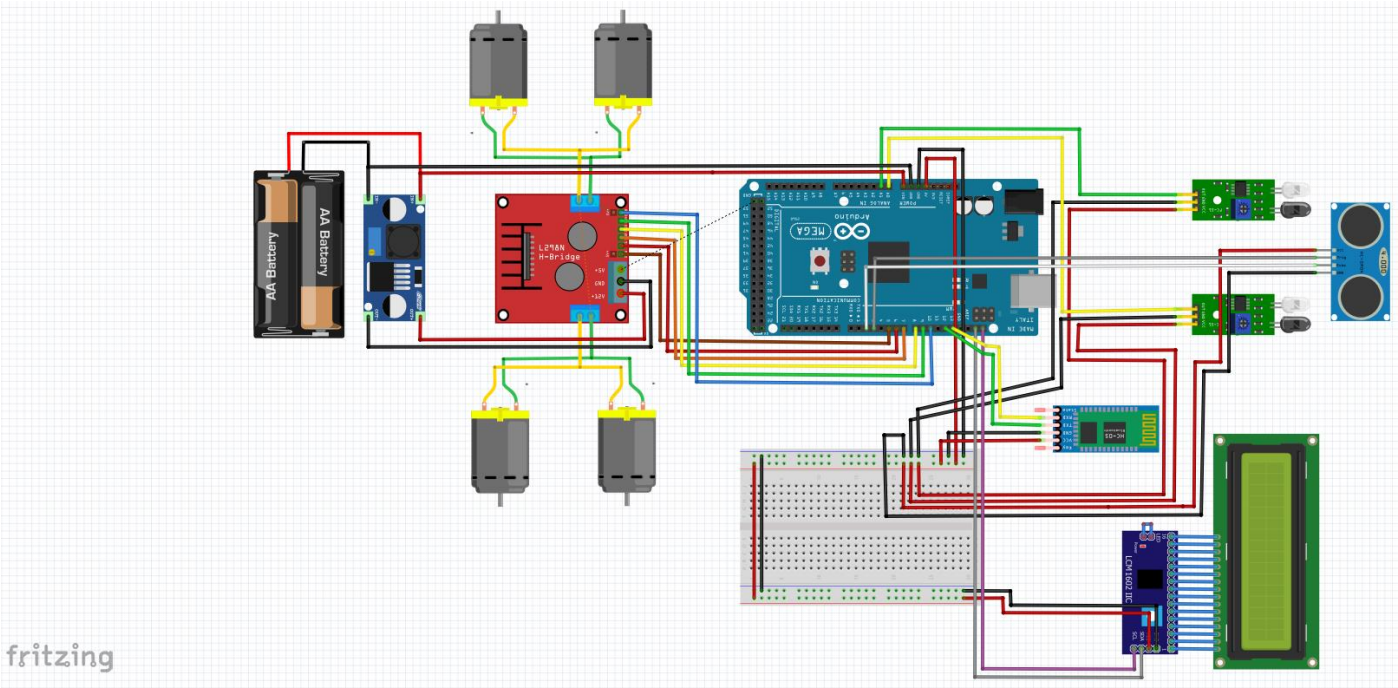
To activate this mode, we equipped the robot with a Bluetooth module and developed Python and Arduino code to allow communication between the two. By establishing a Bluetooth connection, the user can transmit steering instructions to the robot, allowing it to change direction and move outside the tracked line.

This enhancement to the robot offers greater flexibility in changing environments and provides a more interactive user experience. By switching from the automated line tracking mode to the manual control mode, the user can take full control of the robot to navigate through more complex areas and explore new terrain.

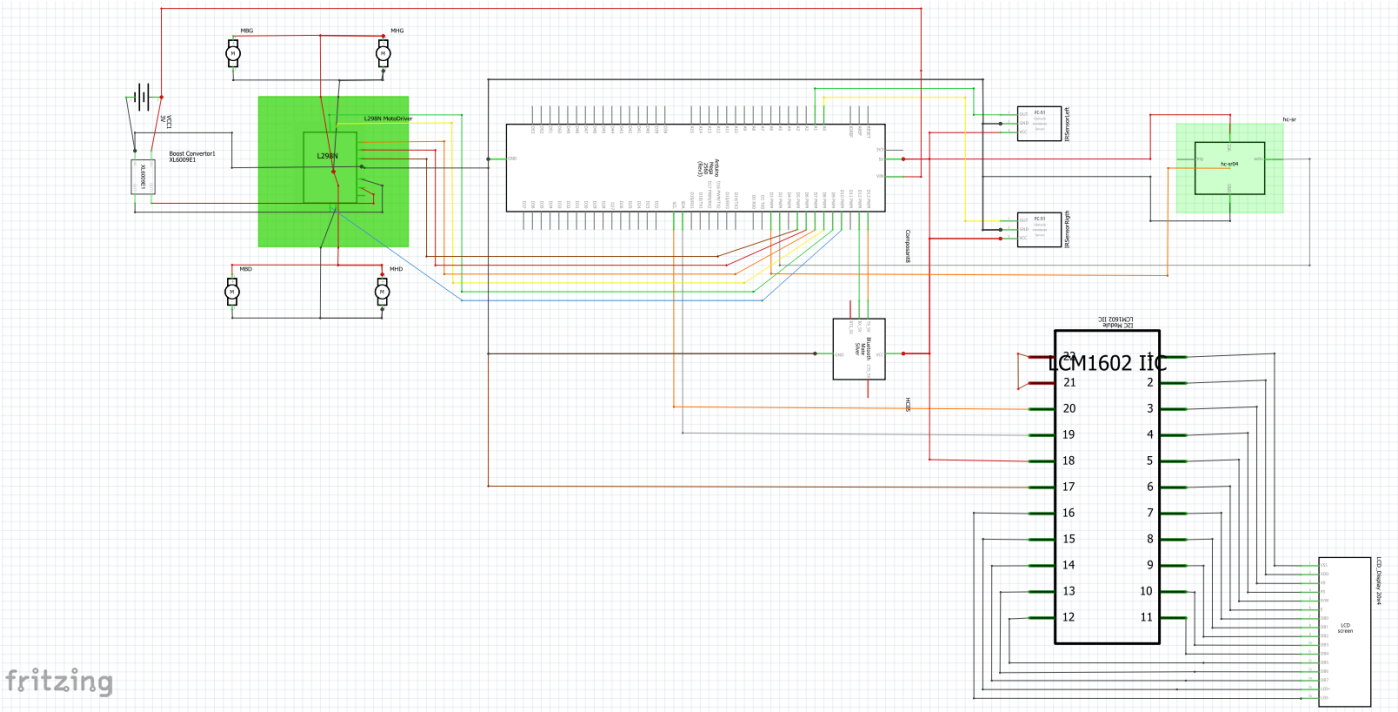
In conclusion, the addition of this manual control feature enables the Line Tracker Robot to become a more versatile and useful tool for a variety of applications, in addition to its primary function of line tracking.

2. Circuit schematic

- **Schematic of the project :**



- **Electrical schematic of the project :**



3. Source code for the Arduino

GITHUB :

https://github.com/ThomasMarcal/Mechatronic-project---Line-Tracker-Robot

CODE ARDUINO :

```
// Code by : MARCAL Thomas

// ----------------------------------------------------------

// Project Mechatronics

// Line Tracker Robot
// +  Obstacle detection and avoidance
// +  Control of the car via bluetooth module


// ----------------------------------------------------------
// ----------------------------------------------------------
// ----------------------------------------------------------

// LCD screen
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 20, 4); // 0x27 LCD screen with 20 columns and 4 rows


// ----------------------------------------------------------

// BLUETOOTH
#include <SoftwareSerial.h>
SoftwareSerial hc06(12,13); // Create a software serial port and assign the pins for TX and RX


// ----------------------------------------------------------

// Define strings

String cmd= "";
String mode = "PILOTAGE";

// ----------------------------------------------------------

// Control Motors

// MOTORS
#define enA 10//Enable1 L298 Pin enA
#define in1 9 //Motor1  L298 Pin in1
#define in2 8 //Motor1  L298 Pin in2
#define in3 7 //Motor2  L298 Pin in3
#define in4 6 //Motor2  L298 Pin in4
#define enB 5 //Enable2 L298 Pin enB

// SENSORS
#define R_S A0 //ir sensor Right
#define L_S A1 //ir sensor Left

// CONST
int ValueR;
int ValueL;

// ----------------------------------------------------------

// Define custom characters for the LCD

byte FAHG[8] = {
  0b00001,
  0b00011,
  0b00111,
  0b01111,
  0b11111,
  0b11111,
  0b11111,
  0b00111
};
byte FAHD[8] = {
  0b10000,
  0b11000,
  0b11100,
  0b11110,
  0b11111,
  0b11111,
  0b11111,
  0b11100
};
```

```
byte FABD[8] = {
  0b11100,
  0b11100,
  0b11100,
  0b11100,
  0b11100,
  0b11100,
  0b11100,
  0b11100
};
byte FABG[8] = {
  0b00111,
  0b00111,
  0b00111,
  0b00111,
  0b00111,
  0b00111,
  0b00111,
  0b00111
};
byte FArBG[8] = {
  0b00111,
  0b11111,
  0b11111,
  0b11111,
  0b01111,
  0b00111,
  0b00011,
  0b00001
};
byte FArBD[8] = {
  0b11100,
  0b11111,
  0b11111,
  0b11111,
  0b11110,
  0b11100,
  0b11000,
  0b10000
};
byte FDBD[8] = {
  0b11111,
  0b11111,
  0b11110,
  0b11100,
  0b11000,
  0b10000,
  0b00000,
  0b00000
};
byte FDHD[8] = {
  0b00000,
  0b00000,
  0b10000,
  0b11000,
  0b11100,
  0b11110,
  0b11111,
  0b11111
};
byte FDHG[8] = {
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b11111,
  0b11111,
  0b11111
};
byte FDBG[8] = {
  0b11111,
  0b11111,
  0b11111,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000
};
byte FGHG[8] = {
  0b00000,
  0b00000,
  0b00001,
  0b00011,
  0b00111,
  0b01111,
  0b11111,
  0b11111
};
byte FGBG[8] = {
  0b11111,
  0b11111,
```

```
  0b01111,
  0b00111,
  0b00011,
  0b00001,
  0b00000,
  0b00000
};
byte CarreBas[8] = {
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b00000,
  0b00000,
  0b00000
};
byte CarreHaut[8] = {
  0b00000,
  0b00000,
  0b00000,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111
};

// ----------------------------------------------------------

// Ultrasonic sensor to avoid obstacles

const byte TRIGGER_PIN = 3; // Broche TRIGGER
const byte ECHO_PIN = 2;    // Broche ECHO

const unsigned long MEASURE_TIMEOUT = 25000UL; // 25ms = ~8m à 340m/s

/* Speed of sound in air in mm/us */
const float SOUND_SPEED = 340.0 / 1000;


// ----------------------------------------------------------

void setup(){ // SETUP CODE

  // Pin initialization
  // Sensors
  pinMode(R_S, INPUT);
  pinMode(L_S, INPUT);
  // Motors
  pinMode(enA, OUTPUT); // declare as output for L298 Pin enA
  pinMode(in1, OUTPUT); // declare as output for L298 Pin in1
  pinMode(in2, OUTPUT); // declare as output for L298 Pin in2
  pinMode(in3, OUTPUT); // declare as output for L298 Pin in3
  pinMode(in4, OUTPUT); // declare as output for L298 Pin in4
  pinMode(enB, OUTPUT); // declare as output for L298 Pin enB
  // Ultrasonic sensor
  pinMode(TRIGGER_PIN, OUTPUT);
  digitalWrite(TRIGGER_PIN, LOW); // The TRIGGER pin must be at LOW when not in use
  pinMode(ECHO_PIN, INPUT);

  // SETUP motor speed in 0
  analogWrite(enA, 0); // Write The Duty Cycle 0 to 255 Enable Pin A for Motor1 Speed
  analogWrite(enB, 0); // Write The Duty Cycle 0 to 255 Enable Pin B for Motor2 Speed

  // SETUP display on the LCD screen
  lcd.init(); // Initialization of the LCD screen
  lcd.cursor_on();
  lcd.blink_on();
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Mechatronic Project");
  lcd.setCursor(0,1);
  lcd.print("Line Tracker Robot");
  lcd.setCursor(0,2);
  lcd.print("+ Control Bluetooth");
  lcd.setCursor(0,3);
  lcd.print("+ Obstacle Detection");
  delay(6000);
  lcd.clear();
  delay(500);

  // Preparation of the communication with the Bluetooth module
  hc06.begin(9600);
  // Serial.begin(9600);
}


// ----------------------------------------------------------

void loop() { // LOOP Code

// Communication with the Bluetooth module
```

```
  while(hc06.available()>0){
    cmd += (char)hc06.read();
  }

// OBSTACLE DETECTION
  // Start a distance measurement by sending a 10µs HIGH pulse on the TRIGGER pin
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);
  // Measures the time between the sending of the ultrasonic pulse and its echo (if it exists)
  long measure = pulseIn(ECHO_PIN, HIGH, MEASURE_TIMEOUT);
  // Calculates the distance from the measured time
  float distance = measure / 2.0 * SOUND_SPEED;

  lcd.setCursor(0,2);
  lcd.print(distance);

// MODE OF THE CAR
  if (cmd != ""){
    if (cmd == "pilotage") {
      mode = "PILOTAGE";
    }
    else if (cmd == "suiveur") {
      mode = "SUIVEUR";
    }
  }

// Mode "PILOTAGE"
  if (mode == "PILOTAGE") {
    lcd.setCursor(0,3);
    lcd.print("Mode : Pilotage");
    // Depending on the command sent the car moves
    if(cmd!=""){
      if (distance < 150){ // If there is an obstacle in front of us we can't move forward
        if(cmd=="arriere"){
          AffichageFlecheArriere();
          backwards();
        }
        if(cmd=="droite"){
          AffichageFlecheDroite();
          turnRight();
        }
        if(cmd=="gauche"){
          AffichageFlecheGauche();
          turnLeft();
        }
        if(cmd=="stop"){
          AffichageStop();
          Stop();
        }
      }
      else{
        if(cmd=="avant"){
          AffichageFlecheAvant();
          forward();
        }
        if(cmd=="arriere"){
          AffichageFlecheArriere();
          backwards();
        }
        if(cmd=="droite"){
          AffichageFlecheDroite();
          turnRight();
        }
        if(cmd=="gauche"){
          AffichageFlecheGauche();
          turnLeft();
        }
        if(cmd=="stop"){
          AffichageStop();
          Stop();
        }
      }
      cmd=""; //reset cmd
    }
    delay(100);
  }
  // Mode "SUIVEUR"
  else if (mode == "SUIVEUR") {
    lcd.setCursor(0,3);
    lcd.print("Mode : Suiveur");

    if(distance <= 150) { // If an obstacle is detected, the robot goes around it
      Stop();
      delay(100);
      turnRight();
      delay(1300);
      forward();
      delay(600);
      turnLeft();
      delay(1500);
      forward();
      delay(1100);
```

```
      turnLeft();
      delay(700);
      forward();
      delay(300);
    }

    int sensorValueR = analogRead(R_S);
    int sensorValueL = analogRead(L_S);

    if (sensorValueR > 100){
      ValueR = 1;
    }
    else {
      ValueR = 0;
    }
    if (sensorValueL > 100){
      ValueL = 1;
    }
    else {
      ValueL = 0;
    }

    //Serial.println(sensorValueR);
    //Serial.println(sensorValueL);

    if((ValueR == 0)&&(ValueL == 0)) { // If the sensors are on black
      AffichageFlecheAvant();
      forward(); // Go straight ahead
    }
    else if((ValueR == 1)&&(ValueL == 0)) { // If right sensor on white and left sensor on black
      AffichageFlecheDroite();
      turnRight(); // Turn to the right
    }
    else if((ValueR == 0)&&(ValueL == 1)) { // If right sensor on black and left sensor on white
      AffichageFlecheGauche();
      turnLeft(); // Turn to the left
    }
    else if((ValueR == 1)&&(ValueL == 1)) { // If the sensors are on white
      AffichageStop();
      Stop(); // Stop
    }

    if (cmd != ""){
      cmd=""; //reset cmd
    }
    delay(100);
  }

  delay(200);
  lcd.clear();

}


// ---------------------------------------------------------

// Functions used
void forward(){  // Allows the car to move forward
analogWrite(enA, 60);
analogWrite(enB, 60);
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
}

void backwards(){  // Allows the car to move backwards
analogWrite(enA, 60);
analogWrite(enB, 60);
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
}

void turnRight(){ // Allows the car to turn right
analogWrite(enA, 90);
analogWrite(enB, 60);
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
}

void turnLeft(){ // Allows the car to turn left
analogWrite(enA, 60);
analogWrite(enB, 90);
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
}
```

```
void Stop(){ // Allows the car to stop
analogWrite(enA, 0);
analogWrite(enB, 0);
digitalWrite(in1, LOW);
digitalWrite(in2, LOW);
digitalWrite(in3, LOW);
digitalWrite(in4, LOW);
}


// Arrow display
void AffichageFlecheAvant(){
  lcd.createChar(0, FAHG);
  lcd.createChar(1, FAHD);
  lcd.createChar(2, FABG);
  lcd.createChar(3, FABD);
  lcd.setCursor(0,0);
  lcd.write((byte)0);
  lcd.setCursor(0,1);
  lcd.write((byte)2);
  lcd.setCursor(1,0);
  lcd.write((byte)1);
  lcd.setCursor(1,1);
  lcd.write((byte)3);
}
void AffichageFlecheArriere(){
  lcd.createChar(0, FABG);
  lcd.createChar(1, FABD);
  lcd.createChar(2, FArBG);
  lcd.createChar(3, FArBD);
  lcd.setCursor(0,0);
  lcd.write((byte)0);
  lcd.setCursor(0,1);
  lcd.write((byte)2);
  lcd.setCursor(1,0);
  lcd.write((byte)1);
  lcd.setCursor(1,1);
  lcd.write((byte)3);
}
void AffichageFlecheDroite(){
  lcd.createChar(0, FDHG);
  lcd.createChar(1, FDHD);
  lcd.createChar(2, FDBG);
  lcd.createChar(3, FDBD);
  lcd.setCursor(0,0);
  lcd.write((byte)0);
  lcd.setCursor(0,1);
  lcd.write((byte)2);
  lcd.setCursor(1,0);
  lcd.write((byte)1);
  lcd.setCursor(1,1);
  lcd.write((byte)3);
}
void AffichageFlecheGauche(){
  lcd.createChar(0, FGHG);
  lcd.createChar(1, FDHG);
  lcd.createChar(2, FGBG);
  lcd.createChar(3, FDBG);
  lcd.setCursor(0,0);
  lcd.write((byte)0);
  lcd.setCursor(0,1);
  lcd.write((byte)2);
  lcd.setCursor(1,0);
  lcd.write((byte)1);
  lcd.setCursor(1,1);
  lcd.write((byte)3);
}
void AffichageStop(){
  lcd.createChar(0, CarreHaut);
  lcd.createChar(1, CarreHaut);
  lcd.createChar(2, CarreBas);
  lcd.createChar(3, CarreBas);
  lcd.setCursor(0,0);
  lcd.write((byte)0);
  lcd.setCursor(0,1);
  lcd.write((byte)2);
  lcd.setCursor(1,0);
  lcd.write((byte)1);
  lcd.setCursor(1,1);
  lcd.write((byte)3);
}

// END OF THE CODE
```

CODE PYTHON :

```
import serial
import time
from pynput import keyboard

result = 'stop'
mode = "pilotage"

print("Start")
port="COM6"
bluetooth=serial.Serial(port, 9600)
print("Connected")
bluetooth.flushInput()

def on_key_release(key):
    global result
    if result != 'stop':
        result = 'stop'
        print(result)
        result_bytes = result.encode('utf_8')
        bluetooth.write(result_bytes)

def on_key_pressed(key):
    global result, mode
    if key == keyboard.Key.space:
        mode = "pilotage"
        result = 'pilotage'
        print("Mode pilotage activé")
        result_bytes = result.encode('utf_8')
        bluetooth.write(result_bytes)
    elif key == keyboard.Key.shift:
        mode = "suiveur"
        result = 'suiveur'  # envoyer "suiveur" pour activer le mode suiveur
        print("Mode suiveur de ligne activé")
        result_bytes = result.encode('utf_8')
        bluetooth.write(result_bytes)
    elif mode == "pilotage":
        if key == keyboard.Key.up:
            result = 'avant'
            print(result)
        elif key == keyboard.Key.down:
            result = 'arriere'
            print(result)
        elif key == keyboard.Key.left:
            result = 'gauche'
            print(result)
        elif key == keyboard.Key.right:
            result = 'droite'
            print(result)
        result_bytes = result.encode('utf_8')
        bluetooth.write(result_bytes)
    elif mode == "suiveur":
        if key == keyboard.Key.space or key == keyboard.Key.shift:
            mode = "pilotage"
            result = 'pilotage'
            result_bytes = result.encode('utf_8')
            bluetooth.write(result_bytes)
            print("Mode pilotage activé")
        elif key not in [keyboard.Key.up, keyboard.Key.down, keyboard.Key.left, keyboard.Key.right]:
            # Si la touche pressée n'est pas valide pour le mode suiveur, passer automatiquement en mode "pilotage"
            mode = "pilotage"
            result = 'pilotage'
            result_bytes = result.encode('utf_8')
            bluetooth.write(result_bytes)
            print("Mode pilotage activé")
        else:
            result = mode
            result_bytes = result.encode('utf_8')
            bluetooth.write(result_bytes)

    print(result_bytes)
with keyboard.Listener(on_release=on_key_release, on_press=on_key_pressed) as listener:
    listener.join()
```

4. Difficulties you encountered.

During the setup of this project, we encountered three major problems.

The first was related to the installation of the infrared sensors, which proved to be complex. Indeed, the sensors did not differentiate between light and dark colors enough, making it difficult for our robot to track the line. We tried several tests with the provided IR sensors, but without success. To solve this problem, we opted to purchase a new model of IR sensor, which matched the model used in all YouTube videos about Line Tracker Robots. This change in sensor allowed our robot to accurately detect color differences, greatly improving its ability to follow a line.

The second problem we encountered was related to false contacts during assembly. Although this was an easily correctable problem, it took us some time to resolve. In the end, we were able to remedy this problem and the assembly was successfully completed.

And finally, we encountered a major problem at the end of our project. Indeed, the battery provided to power our system did not provide enough power to start our robot. Every time we asked for too much power, the Arduino would restart, preventing us from testing our robot properly. So we decided to continue using the wire power supply provided earlier in the project, when we could not use the batteries due to a bad converter.

In summary, despite the difficulties encountered in setting up the infrared sensors and false contacts during assembly, we were able to resolve these issues and successfully complete our project. These experiences allowed us to learn and develop our technical expertise, which will be beneficial for our future projects.

5. What is missing in your project and your suggestions to make the project better.

To improve our project, there are several points to consider.

Firstly, to improve obstacle detection, we could have installed the ultrasonic sensor on a servo motor, which would have allowed it to pivot at a 180-degree angle. This way, our robot would have been able to detect and avoid obstacles that were not necessarily on the line. Additionally, by improving obstacle avoidance management, our robot would have been able to navigate more smoothly and efficiently.

Secondly, we decided to opt for keyboard controls in Python to control our vehicle. This was because no one in our group had an Android device, making it difficult to set up an application for piloting. However, we could have opted for a user interface with Python to pilot the robot, although it would have been less practical than keyboard controls. Furthermore, by adding an option to change the wheel speed, our robot would have been more adaptable and could have been used in a wider variety of situations.

6. Record a video and upload it, e.g., in Youtube and provide the link.

https://youtu.be/JblUAZHYq9g