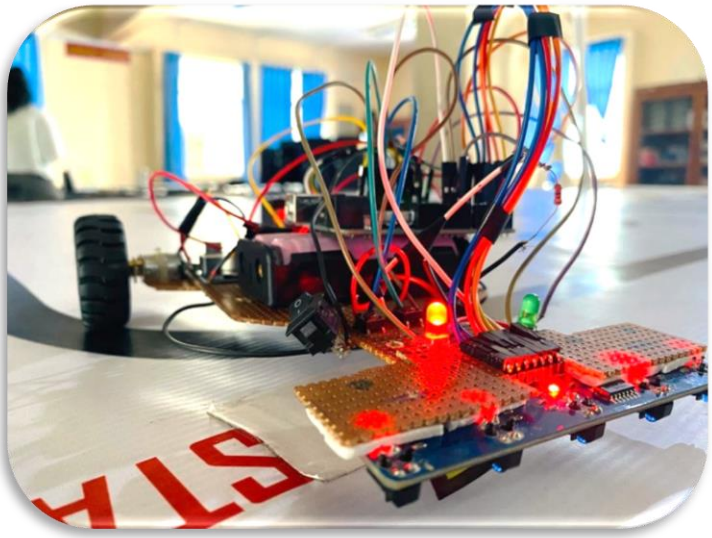
Design & Implementation of a Line Following Robot

GP 118 – Mini Project

Project Report



E/21/008	Abeyrathne S.G.G.S.
E/21/047	Bandara B.S.M.L.U.
E/21/087	Dewagedara D.M.E.S.
E/21/118	Samarakoon S.B.M.P.D.
E/21/178	Hearth H.M.M.K.
E/21/302	Perera W.S.S.
E/21/320	Rajapakshe W.R.G.S.D.
E/21/393	Sudusinghe V.A.

Table of Contents

Abstract

The primary objective of this project was to design and implement a line-following robot capable of autonomously navigating a path marked by a line. The robot was required to demonstrate precise line tracking, handle various turns and intersections, and maintain consistent speed and stability across different environmental conditions, including changes in lighting and surface texture

Introduction

In the field of robotics, line-following robots are a fundamental example of autonomous mobile robots that can navigate a predetermined path marked on the ground, typically using a black or white line on a contrasting surface. These robots serve as a practical application of various engineering disciplines, including electronics, mechanical design, and control systems.

The primary objective of this project was to design, build, and test a line-following robot capable of accurately and efficiently tracking a line on a surface. This involved the integration of various components such as sensors, motors, a control system, and a lightweight chassis, all working in harmony to achieve smooth and reliable navigation.

The design process began with selecting suitable sensors for line detection, followed by the development of a control algorithm to ensure precise movement along the path. Throughout the project, iterative testing and refinement were essential in addressing challenges such as sensor calibration, motor speed control, and path alignment. By leveraging a sensor array with a PID control system, the robot was able to adapt to different path geometries, including straight lines, curves, and intersections.

This report provides a detailed overview of the design process, the challenges encountered, the solutions implemented, and the results obtained.

Hardware Design

Components

- **Chassis and Structure:** The chassis was designed using a dot board, a versatile and easily customized material. This design choice allowed for quick prototyping and modifications. The dot board provided a sturdy base while enabling easy attachment and adjustment of other components. The design emphasized balanced weight distribution to maintain consistent contact with the surface, ensuring precise movement.

- **Sensor Array:** A 5-sensor infrared array was employed to detect the line, providing extensive coverage and enabling the robot to track the line's position relative to its center. The sensors were arranged in a linear configuration at the front of the robot, allowing for precise detection and correction, especially during sharp turns or when encountering intersections.

- **Microcontroller:** An Arduino Uno microcontroller was selected for its ease of use and compatibility with the sensor array. It processed the sensor inputs and controlled the motors based on the detected line position, ensuring responsive and accurate navigation.

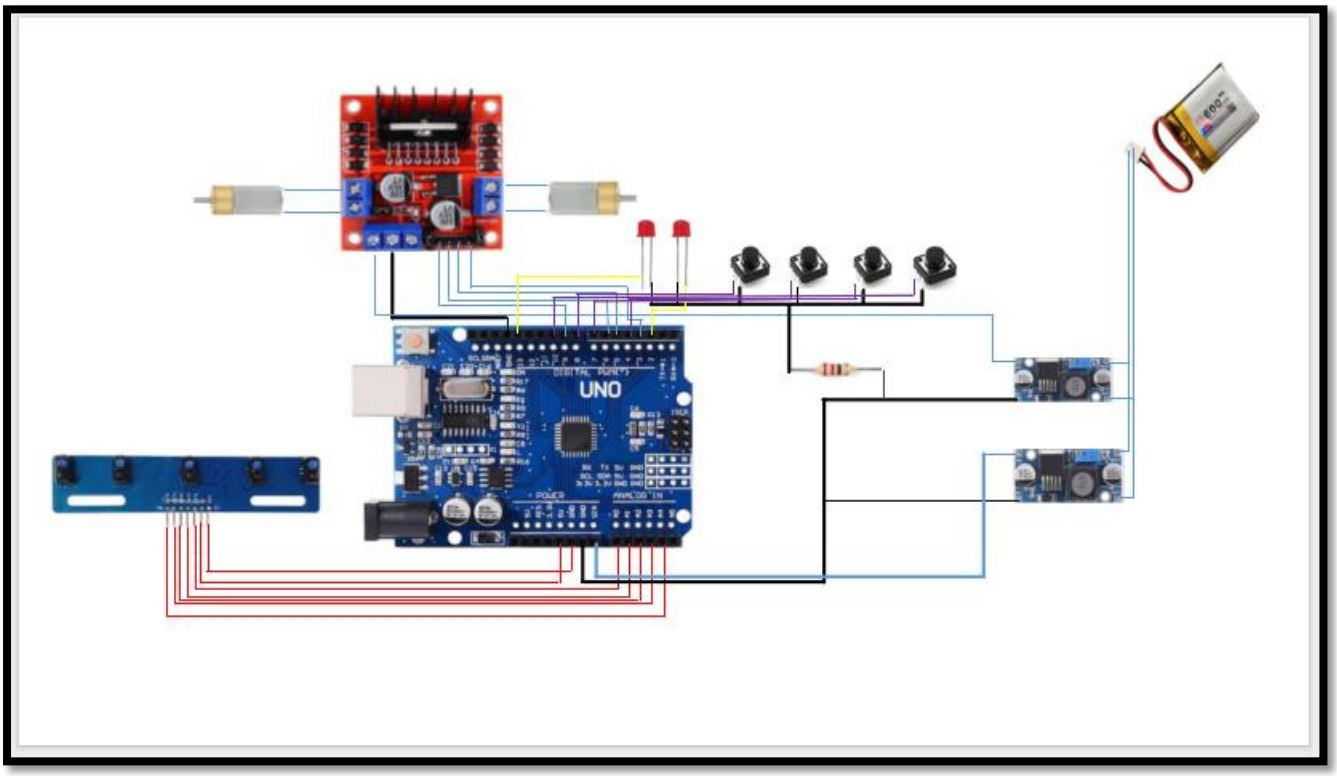
- **Motors and Actuators:** The robot utilized 300 RPM gear motors to drive the wheels, providing a balance between speed and torque. These motors were chosen for their ability to deliver consistent performance, allowing the robot to navigate quickly while maintaining precise control over its movements.

- **Motor Shield:** A motor shield was used to simplify the connection between the Arduino microcontroller and the gear motors. The motor shield allowed for easy control of the motors, providing features like current sensing and motor direction control.

- **Voltage Regulators:** The robot was equipped with both buck and boost voltage regulators to ensure a stable power supply to the components. The buck regulator was used to step down the voltage from the batteries to a suitable level for the microcontroller and sensors, while the boost regulator was used to step up the voltage for the motors, ensuring they received adequate power for consistent performance.

- **Power Supply:** The robot was powered by two 3.7V rechargeable batteries connected in series, providing a total of 7.4V. This configuration, combined with the voltage regulators, offered a stable and sufficient power supply to all components, ensuring consistent performance during the robot's operation.

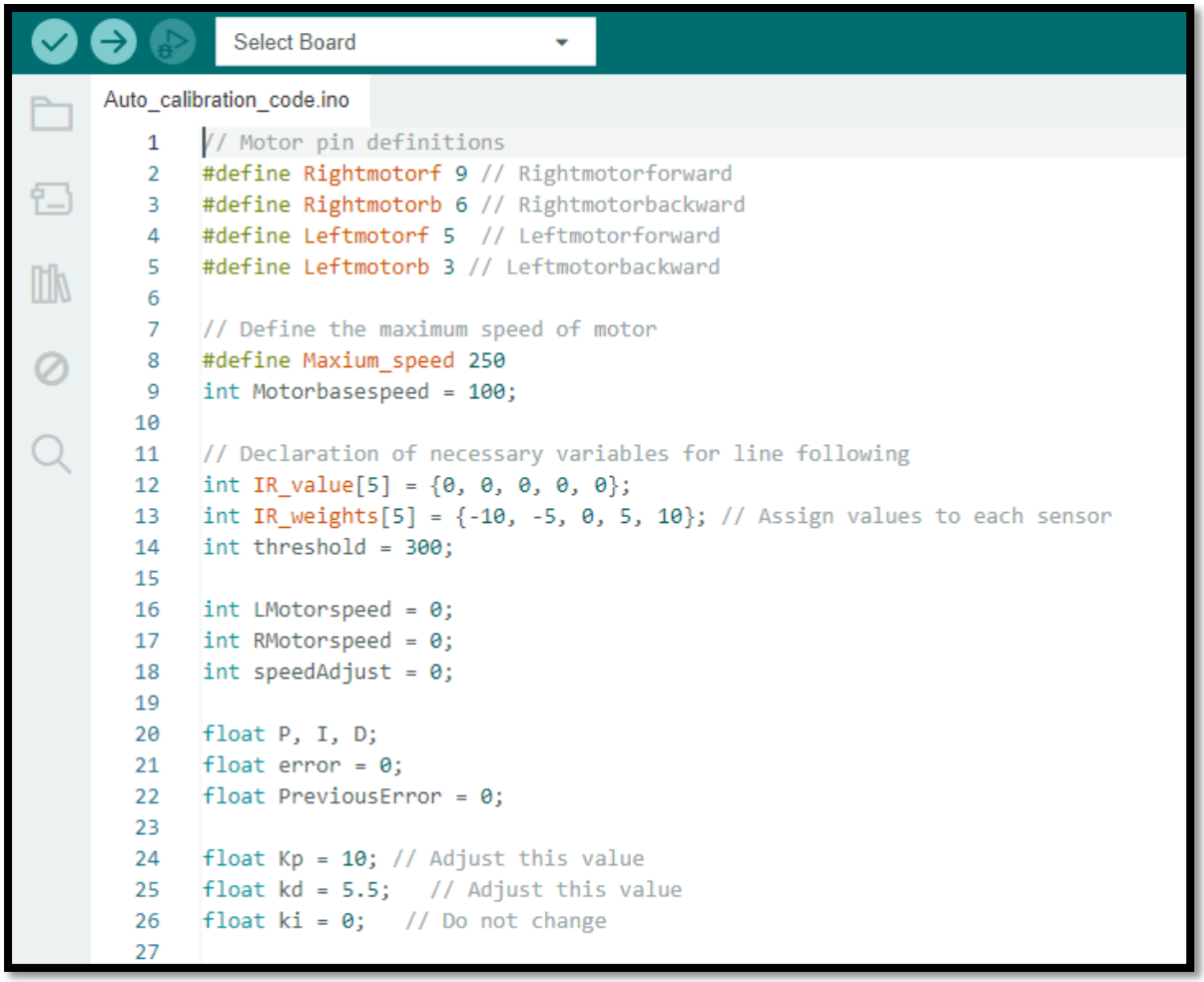
Circuit Diagram



Software Development

The software was developed in C++ using the Arduino IDE. The code included functions for reading sensor data, processing this data to determine the robot's position relative to the line, and adjusting motor speeds to keep the robot on track

Our Code




```
// Motor pin definitions
#define Rightmotorf 9 // Rightmotorforward
#define Rightmotorb 6 // Rightmotorbackward
#define Leftmotorf 5  // Leftmotorforward
#define Leftmotorb 3 // Leftmotorbackward

// Define the maximum speed of motor
#define Maxium_speed 250
int Motorbasespeed = 100;

// Declaration of necessary variables for line following
int IR_value[5] = {0, 0, 0, 0, 0};
int IR_weights[5] = {-10, -5, 0, 5, 10}; // Assign values to each sensor
int threshold = 300;

int LMotorspeed = 0;
int RMotorspeed = 0;
int speedAdjust = 0;

float P, I, D;
float error = 0;
float PreviousError = 0;

float Kp = 10; // Adjust this value
float kd = 5.5;   // Adjust this value
float ki = 0;   // Do not change

```

```
27
28  // Push button and LED
29  int button1 = 4;
30  int button2 = 7;
31  int button3 = 10;
32  int button4 = 11;
33  int led12 = 2;
34  int led13 = 12;
35  bool button1_state, button2_state, button3_state, button4_state;
36
37  // Calibrations
38  int sensorValues[5];
39  int minValues[5];
40  int maxValues[5];
41  float trus[5];
42
43  // Define the function prototypes
44  void PID_control();
45  void reading();
46  void set_forward();
47  void stop();
48  void turn();
49  void set_speed();
50  void button_status();
51  void calibrate();
52
```

```
54    // Setup the pins
55    pinMode(Rightmotorf, OUTPUT);
56    pinMode(Rightmotorb, OUTPUT);
57    pinMode(Leftmotorf, OUTPUT);
58    pinMode(Leftmotorb, OUTPUT);
59
60    // Button pins as input
61    pinMode(button1, INPUT_PULLUP);
62    pinMode(button2, INPUT_PULLUP);
63    pinMode(button3, INPUT_PULLUP);
64    pinMode(button4, INPUT_PULLUP);
65
66    // LED pins as output
67    pinMode(led12, OUTPUT);
68    pinMode(led13, OUTPUT);
69    Serial.begin(9600);
70
71    // Calibrations
72    //for (int i = 0; i < 5; i++) {
73    //  minValues[i] = 1023;
74    //  maxValues[i] = 0;
75    //}
76  }
77
```

```
78  void loop() {
79    digitalWrite(led13, HIGH);
80    button_status(); // Reading the status of push buttons
81
82    // When button 1 is pressed, the robot will start to follow lines.
83    while (button1_state != 1) {
84      digitalWrite(led13, LOW);
85      // Blink led12
86      for (int i = 0; i < 5; ++i) {
87        digitalWrite(led12, HIGH);
88        delay(100);
89        digitalWrite(led12, LOW);
90        delay(100);
91      }
92      digitalWrite(led13, HIGH);
93      digitalWrite(led12, HIGH);
94      PID_control();
95    }
96
97    // When button 2 is pressed, check if both motors are running at the same sp
98    while (button2_state != 1) {
99      digitalWrite(led13, LOW);
100     for (int i = 0; i < 3; ++i) {
101       digitalWrite(led12, HIGH);
102       delay(500);
103       digitalWrite(led12, LOW);
104       delay(500);
105     }
```

```
106     set_forward();
107     delay(3000);
108     stop();
109     button_status();
110   }
111
112   // Reading IR values and calibration process
113   while (button4_state != 1) {
114     digitalWrite(led13, LOW);
115     // Calibrations
116     calibrate();
117     // Blink the LED
118     digitalWrite(led12, HIGH);
119     delay(50);
120     digitalWrite(led12, LOW);
121     delay(50);
122     button_status();
123   }
124
125   // Stop mode and print the maximum and minimum values if button 3 is presse
126   while (button3_state != 1) {
127     stop();
128     digitalWrite(led13, HIGH);
129     button_status();
130   }
131 }
```

```
133  void reading() {
134    error = 0;
135    for (byte i = 0; i < 5; i++) {
136      IR_value[i] = analogRead(i);
137      // Convert analog value to digital
138      if (IR_value[i] > threshold) {
139        IR_value[i] = 1;
140      } else {
141        IR_value[i] = 0;
142      }
143      error += IR_value[i] * IR_weights[i];
144    }
145  }
146
147  void PID_control() {
148    while (1) {
149      reading();
150      // Calculations for PID control
151      P = error;
152      I = I + error;
153      D = error - PreviousError;
154      PreviousError = error;
155      // Main equations
156      speedAdjust = (Kp * P + ki * I + kd * D);
157      // Set the speed
158      LMotorspeed = Motorbasespeed + speedAdjust;
159      RMotorspeed = Motorbasespeed - speedAdjust;
160
```

```
161      if (LMotorspeed < 0) {
162        LMotorspeed = 0;
163      }
164      if (LMotorspeed > Maxium_speed) {
165        LMotorspeed = Maxium_speed;
166      }
167      if (RMotorspeed < 0) {
168        RMotorspeed = 0;
169      }
170      if (RMotorspeed > Maxium_speed) {
171        RMotorspeed = Maxium_speed;
172      }
173      set_speed();
174    }
175  }
176
177  void set_speed() {
178    analogWrite(Leftmotorf, LMotorspeed);
179    analogWrite(Rightmotorf, RMotorspeed);
180  }
181
182  void set_forward() {
183    analogWrite(Leftmotorf, Maxium_speed);
184    analogWrite(Rightmotorf, Maxium_speed);
185  }
```

```
}

void stop() {
  analogWrite(Leftmotorf, 0);
  analogWrite(Rightmotorf, 0);
  analogWrite(Rightmotorb, 0);
  analogWrite(Leftmotorb, 0);
}

void turn() {
  analogWrite(Leftmotorf, 150);
  analogWrite(Rightmotorb, 150);
}

// Reading the status of push buttons and saving the state.
void button_status() {
  button1_state = digitalRead(button1);
  button2_state = digitalRead(button2);
  button3_state = digitalRead(button3);
  button4_state = digitalRead(button4);
}
```

```

void calibrate() {
  for (int i = 0; i < 3000; i++) {
    turn();
    for (int i = 0; i < 5; i++) {
      int value = analogRead(i);
      if (value < minValues[i]) {
        minValues[i] = value;
      }
      if (value > maxValues[i]) {
        maxValues[i] = value;
      }
    }
  }
  for (int i = 0; i < 5; i++) {
    trus[i] = (minValues[i] + maxValues[i]) / 2.0;
    Serial.print(trus[i]);
    Serial.print(" ");
  }
  Serial.println();
  stop();
}
```

Overview of the Code

Motor Pin Definitions:
The motors are controlled using four pins: two for the right motor (Rightmotorf, Rightmotorb) and two for the left motor (Leftmotorf, Leftmotorb). These pins are connected to the motor driver, which controls the direction and speed of the motors.

Speed and PID Constants:

The maximum speed of the motors is defined as 250.A base speed (Motorbasespeed) is set to 150, which is the default speed for both motors.The PID control constants Kp (proportional), kd (derivative), and ki (integral) are defined. Kp and kd are used to adjust the motor speed to keep the robot on the line.

IR Sensor Array:

The robot uses 5 IR sensors to detect the line. The IR_weights array assigns a weight to each sensor based on its position (negative values for left sensors, positive for right).

The threshold value is used to determine if a sensor detects the line.

Buttons and LEDs:

Four buttons (button1, button2, button3, button4) are used to control different modes of the robot.Two LEDs (led12, led13) are used to provide visual feedback.

Functions

setup():

This function sets up the pins for the motors, buttons, and LEDs, and initializes the serial communication. It also initializes the calibration values.

loop():

The main loop of the program checks the status of the buttons and executes different behaviours based on which button is pressed.

reading():

Reads the values from the IR sensors and calculates the error based on which sensors detect the line. The error is a weighted sum of the sensor values, which helps the robot determine how far it is from the center of the line.

PID_control():

This function implements the PID control algorithm. It adjusts the motor speeds based on the error calculated in the reading() function. The PID control adjusts the robot's direction to keep it on the line.

set_speed():

Sets the speed of the motors based on the PID control output.

set_forward():

Moves the robot forward at maximum speed.

stop():

Stops the robot by setting the motor speeds to 0.

turn():

A simple function to make the robot turn by running one motor forward and the other backward.

button_status():

Reads the status of the buttons and updates their states.

Explanation of PID Control

The PID control system adjusts the robot's motor speeds based on the error in its position relative to the line. The error is calculated as the difference between the actual sensor readings and the desired position (centered on the line). The PID controller uses this error to adjust the speed of the motors to minimize the error and keep the robot on the line.

- **P (Proportional)**: Adjusts the speed based on the current error.
- **I (Integral)**: Accumulates the error over time to correct for any consistent bias (not used in this case).
- **D (Derivative)**: Reacts to the rate of change of the error to dampen the response and prevent overshooting.

Step-by-Step Explanation of the Calibration Function

Purpose of Calibration

The calibration ensures that the robot's sensors can correctly detect the line in specific environment where it will operate. By determining the minimum and maximum sensor values during calibration, the robot can set accurate thresholds for distinguishing between the line and the surface. This step is crucial for the robot's ability to follow the line accurately, as the correct identification of the line by the sensors directly impacts the effectiveness of the PID control system.

Initialization:

Before the calibration loop begins, the *minValues* array is initialized to the maximum possible value (1023), and the *maxValues* array is initialized to the minimum possible value (0). This setup ensures that any sensor reading during calibration will correctly update these arrays.

Calibration Loop:

The robot performs 10,000 iterations where it turns in place. The turning allows each sensor to scan different parts of the surface under various conditions (e.g., over the line, off the line, on the edge of the line). This movement is essential to cover a wide range of sensor readings, which helps in accurately determining the minimum and maximum values.

Sensor Reading and Comparison:

- During each iteration, the robot reads the analog value from each of the 5 IR sensors using *analogRead(i)*.

- The sensor reading is then compared against the current *minValues* and *maxValues* for that sensor:
  - If the current reading (value) is lower than the stored *minValues[i]*, the *minValues[i]* is updated with the current reading.
  - If the current reading (value) is higher than the stored *maxValues[i]*, the *maxValues[i]* is updated with the current reading.

Calculating Thresholds:

- After completing all the iterations, the thresholds *(trus[i])* for each sensor are calculated as the average of the minimum and maximum values:

$$\text{trus}[i] = \frac{\text{minValues}[i] + \text{maxValues}[i]}{2}$$

- These thresholds represent the midpoint between the darkest and lightest readings that each sensor detected during the calibration process. This midpoint will later be used to distinguish whether a sensor is detecting the line or the background.

Outputting Thresholds:

- The calculated thresholds are printed to the Serial Monitor for each sensor. This output allows you to verify the calibration process and see the specific threshold values that will be used during line detection.

Stopping the Robot:

- Finally, the robot is stopped using the *stop()* function, ending the calibration process.

Testing

With different environments

Surface- The robot was tested on a variety of surfaces to simulate different real-world conditions. These included smooth, flat surfaces such as a whiteboard and textured surfaces like a rubber mat.
The goal was to assess the robot's performance across different types of floor conditions

Line Patterns- The line-following path included several different patterns:

- Straight Lines- To evaluate the robot's ability to maintain a straight path.
- Curved Lines-To test the robot's handling of gentle and sharp curves.
- Intersections- To assess how well the robot navigated intersections and re-aligned itself after crossing junctions.

With different Test cases

- Scenario 1: Varying Line Widths

- The robot was tested with lines of different widths to evaluate how well it could maintain its path with broader and narrower lines.

- Scenario 2: Sharp Turns

- The robot was subjected to sharp turns to observe its manoeuvrability and how quickly it could correct its path after a turn.

- Scenario 3: Intersection Navigation

- The robot was tested on a path with multiple intersections to examine how effectively it could navigate from one segment to another.

Design Challenges

- **Chassis Weight:** Designing the chassis with a dot board aimed to reduce the overall weight of the robot but introduced challenges related to structural integrity and stability. The lightweight nature of the dot board required additional considerations to ensure the robot remained balanced and durable.

    - **Mitigation:** Reinforced the dot board with additional supports and carefully planned component placement to distribute weight evenly. Ensured that the chassis design-maintained stability while taking advantage of the reduced weight.

- **Motor Selection:** Initially, DC motors were used but had limitations with speed changes not being instantaneous, affecting the robot's performance, especially in dynamic conditions with sharp turns and rapid direction changes.

    - **Mitigation:** Switched to 300 RPM gear motors, which provided more consistent speed control and better performance in managing sharp turns and maintaining stability during line following.

- **Car Length Adjustment:** Initially, the length of the robot made it difficult to keep the sensor array aligned with the path, leading to frequent deviations from the intended path. The length contributed to challenges in making sharp turns and maintaining alignment during curves.

    - **Mitigation:** The length of the car was reduced, which helped in aligning the sensor array more effectively with the path. This adjustment prevented the robot from moving out of the path during operation and significantly improved its ability to follow the path accurately.

- **Sensor Array Accuracy:** Initially, a sensor array was built using single sensors, but the accuracy was very low, which hindered the robot’s performance in detecting and following the line reliably. The limited sensor array could not provide the necessary precision for effective line tracking.

    - **Mitigation:** Upgraded to a TCRT5000 sensor array with 5 sensors, which significantly improved accuracy and reliability in line detection. The new sensor array provided better coverage and more precise feedback, enhancing the robot’s overall performance.

- **Initial Sensor Setup:** At the beginning of the project, digital sensors were used without a PID control system. This setup resulted in the inability to control the speed of the robot effectively, leading to issues with stability and accuracy in following the line.

    - **Mitigation:** Recognizing the limitations, the project team switched to a sensor array with a PID control system. This change allowed for precise speed control and significantly improved the robot's performance in following the line accurately.

- **Sensor Height Calibration:** Determining the optimal height for the sensor array from the path was critical for accurate line detection. Using a serial plotter, the required height was found to be between 1.3 to 1.5 cm, which was crucial for ensuring consistent sensor readings and effective line following.

    - **Mitigation:** Adjusted the sensor array's mounting to maintain the optimal height range of 1.3 to 1.5 cm. Continuous monitoring with the serial plotter ensured that the height remained within the desired range during operation.

- **Sensor Calibration:** Achieving accurate sensor readings with the 5-sensor array proved challenging, especially when detecting different color intensities on the board. The calibration system was added to identify color intensity differences, which was crucial for maintaining accurate line detection.

    - **Mitigation:** Developed a calibration routine that adjusted the sensor thresholds based on the color intensity variations. The system was tested extensively to fine-tune sensor calibration and ensure reliable performance under various conditions.

- **Bend Radius Impact:** The radius of the bend along the path was found to affect the robot’s alignment. When the curvature was large, the robot tended to move out of the path, especially during sharp turns. This was due to the difficulty in maintaining alignment with the line during tight curves.

    - **Mitigation:** Adjusted the control algorithm to better handle sharp curves, ensuring that the robot could adjust its position more effectively when encountering bends with large curvatures. Additional testing was conducted on different bend radii to refine the robot's response and minimize the risk of moving out of the path.

Future Developments

1. **Advanced Sensor Array:**

Current Limitation: The TCRT5000 sensor array with 5 sensors works well but might struggle with more complex paths, such as paths with varying line widths or irregular patterns.

Improvement: Implement a more advanced sensor array with additional sensors or higher precision to better detect line boundaries and improve accuracy in detecting sharp curves or intersections.

2. **Enhanced PID Tuning:**

Current Limitation: The current PID parameters (Kp, Ki, Kd) may not be optimally tuned for all environments, which can lead to instability or slow response times.

Improvement: Introduce an adaptive PID control system that adjusts the parameters in real-time based on the robot's performance. Alternatively, use advanced control algorithms like Fuzzy Logic or a neural network to improve the robot's adaptability to different line conditions.

3. **More Powerful Batteries:**

- **Current Limitation:** The existing rechargeable batteries provide sufficient power but may limit the robot's operational time and performance under higher loads.
- **Improvement:** Future versions of the robot could use more powerful batteries, which would allow for extended run times, higher speeds, and the ability to power more advanced sensors or control systems without compromising performance.

4. **Improved Chassis Design:**

Current Limitation: The chassis is currently designed using a dot board, which helps reduce weight but might not offer optimal durability or modularity.

Improvement: Design a more robust and modular chassis that allows easy upgrades or modifications. Using lightweight yet durable materials like carbon fiber or aluminum could improve both the weight and strength of the chassis.

5. **Speed and Motor Control:**

Current Limitation: The 300 RPM gear motors offer good performance, but their response to speed changes could be further optimized, especially for more complex paths.

Improvement: Upgrade to motors with higher torque or introduce a more precise motor control system, such as using encoders to provide real-time feedback on motor speed and position. This would enable more accurate control during sharp turns or when adjusting speed.

References

Websites

1. Arduino.cc

The official Arduino website provides documentation and tutorials related to using Arduino boards, which likely played a central role in your robot's control system.

[Arduino Documentation] (https://www.arduino.cc/en/Guide/HomePage)

2. Pololu Robotics and Electronics

Pololu offers a range of resources on robotics components, including motor drivers, sensors, and controllers, which can be used to enhance your understanding of the components used in your robot.

[Pololu Resources] (https://www.pololu.com/)

Blogs

1. RobotShop Community Blog

This blog often discusses robotics projects, including line-following robots, providing practical insights and tutorials that might align with your project's design and implementation.

[RobotShop Blog] (https://www.robotshop.com/community/blog)

2. Instructables: Line Following Robot

A step-by-step guide on Instructables that discusses similar projects can provide practical tips and common troubleshooting advice.

[Instructables Guide] (https://www.instructables.com/Line-Following-Robot-1/)

Datasheets

1. TCRT5000 Reflective Optical Sensor

The datasheet for the TCRT5000 sensor array provides detailed specifications and operational characteristics that are critical for understanding the sensor's role in the robot's design.

[TCRT5000 Datasheet] (https://www.vishay.com/docs/83760/tcrt5000.pdf)

2. L298N Motor Driver Module Datasheet

The datasheet for the motor driver module used in your robot can be cited to explain the motor control mechanism and power management.

[L298N Datasheet] (https://www.st.com/resource/en/datasheet/l298.pdf)

Research Papers

1. "A Comparative Study on -Line Following Robots" by Yasir Mehmood, Muhammad Awais Javed, and Muhammad Fahad

This paper discusses different line-following techniques, sensor arrays, and control systems, which could be relevant to your project.

2. "PID Control System Design and Application in Line Following Robot" by Saranya. R, Keerthika. R, Sujitha. J

This paper provides insights into PID control systems used in line-following robots and the impact of tuning PID parameters on robot performance