U.PORTO

ACE - Line Follower Robot

Final Project Report

João Pedro Sá Torres Neiva Passos
Lucien Sagot

Master´s in "Engenharia Eletrotécnica e Computadores"

Contents

1 Introduction

As part of the "Arquiteturas de Computação Embarcada" course, we were tasked with developing a Line Follower Robot Prototype. This robot must be capable of navigating both line-following tracks, avoiding obstacles in its path, and grid-based mazes, steering clear of obstructed routes.

The objective is to create a robot that successfully performs all required tasks while achieving the highest possible times in both navigation modes. To accomplish this, we integrated various embedded computing and robotics technologies.

This report details the design, implementation, and testing of the project, highlighting key engineering decisions and challenges encountered throughout development.

2 Prototype

For the prototype, we primarily used the materials recommended by our teachers, while also incorporating additional components to enhance functionality and improve overall usability.

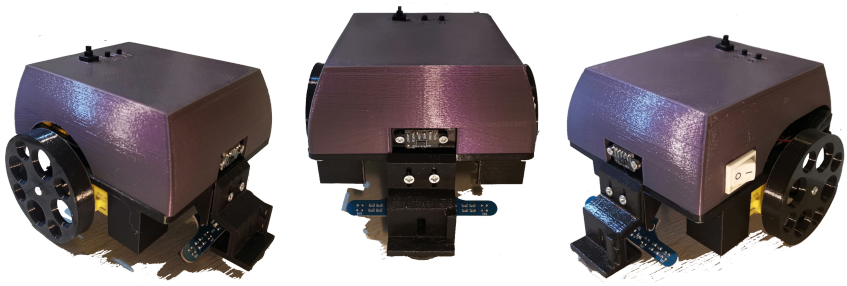

Figure 1: Prototype

To refine the prototype's aesthetics, we added a 3D-printed top cover to conceal the circuits and electronics. Additionally, we integrated buttons for easier control and a power switch to conveniently turn the robot on and off. These improvements make the final product more user-friendly and visually appealing.

2.1 Materials Used

- **Raspberry Pi Pico W**
- **Raspberry Pi Pico IO Shield**
- **L298N Motor Driver**

- **2 x Micro Motors DC with encoder and 120:1 gearbox**
- **TOF Sensor**
- **Analog 5 Sensor Bar**
- **2 x Li-Ion 18650 Battery 3,7V**
- **18650 battery holder**
- **3 x Buttons**
- **100 kOhm Resistor**
- **330 kOhm Resistor**
- **CD74HC4051 Analog Multiplexer**
- **Jumper Cables**

2.2 Schematic

Below is a schematic of the final prototype's electronic connections:

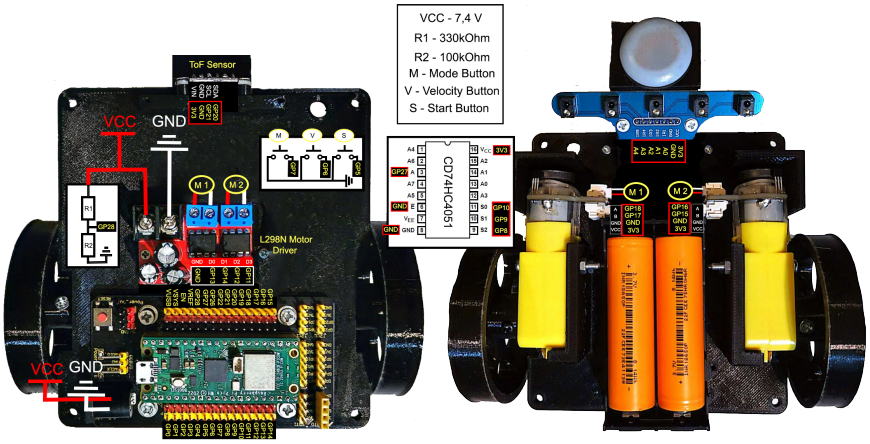

Figure 2: Prototype Schematic

3 Program

The program was developed in five distinct phases. First, we began by testing and configuring the robot's motors, sensors, and electronics to ensure proper functionality. Next, we develop the PID-based line following logic. In the third phase, we implemented the line-following functionality using a state machine for better control and efficiency. This was followed by the development of the Grid Maze Mode, also structured using a state machine. We also integrated configuration buttons, allowing mode and velocity switching, improving overall usability. Finally we developed a way to facilitate PID calibration using the Pico W Wifi capabilities.

The code is configured as follows:

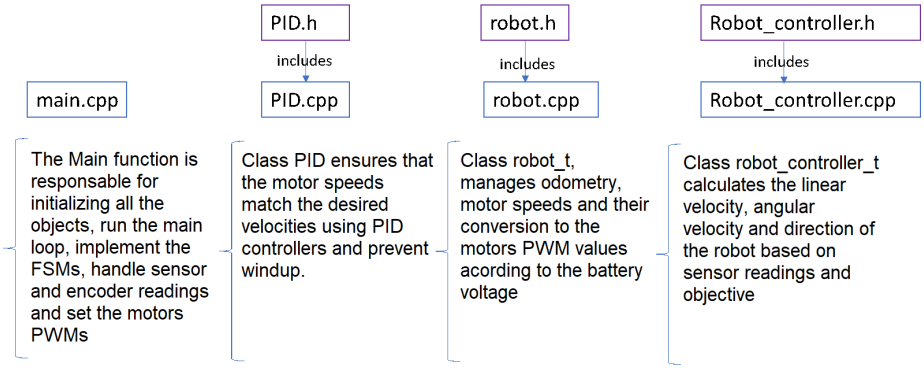

Figure 3: Code Configuration

3.1 Sensors And Motor Configuration

The first step was to test and configure all electronic components, including the sensors, motors, and microcontroller.

3.1.1 TOF Sensor

For the TOF sensor, we initially based our code on libraries and examples provided by our teachers. However, sometimes we encountered occasional misreadings that affected measurement accuracy. To improve reliability, we implemented a solution that reduces false detections. Instead of reacting to a single reading, the sensor now requires multiple consistent readings to confirm the presence of an object, ensuring greater accuracy and stability in detection.

3.1.2 IR Sensors

While setting up the IR sensors, we encountered an issue: the sensors had an analog output, but the sensor bar contained five sensors, while the Raspberry Pi Pico had only three available analog ports—fewer than needed. To resolve this, we used an analog multiplexer, specifically the CD74HC4051, which expands the available analog inputs by providing up to eight additional ports while utilizing only one of the Pico's original analog ports.

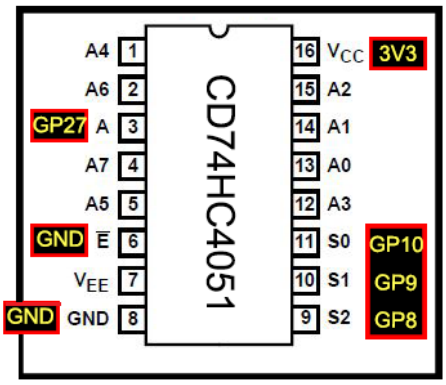

Figure 4: CD74HC4051 Multiplexer

With this multiplexer, we were able to read all five sensors sequentially, introducing a small delay between each reading.

3.2 PID Line Following

After the prototype electronics is all mounted we could start developing the line following code. We chose to use a PID-based approach to this challenge because it allowed for smooth and stable movement and faster and more accurate corrections.

3.2.1 Odometry and PWM Calculation

Our code was built upon the Robot and PID classes provided by our teachers. These classes enabled us to implement robot odometry and calculate the necessary PWM values.

To enhance the accuracy of PWM control, we integrated a voltage divider to measure the battery voltage dynamically, rather than relying on a fixed value. This ensures more precise motor control as battery levels fluctuate.

Our voltage divider consists of two resistors (R1 = 330kOhm and R2 = 100kOhm) and is represented in the schematic below:

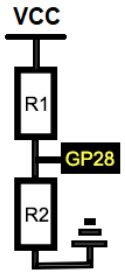

Figure 5: Voltage Divider

To calculate the PWM values, the linear velocity (v) and angular velocity (w) must first be set to guide the robot along the line. The v value remains constant, while the w value must be calculated to adjust the robot's heading and maintain alignment with the line.

3.2.2 PID Line Following Function

To achieve this, we developed a function that calculates the angular velocity (w) based on the readings from the five IR sensors.

The logic behind this function is outlined below:

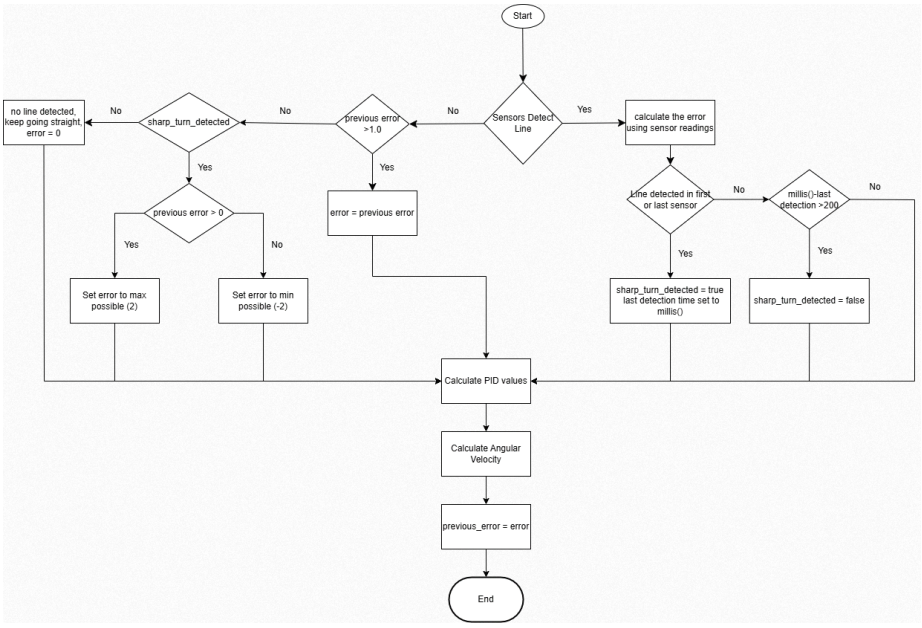

Figure 6: Follow Line PID function

This function first calculates the PID error based on sensor readings, while accounting for exceptions to handle complex behaviors such as sharp turns and missing line segments. It then computes the angular velocity by summing the individual PID components (w=P+I+D).

3.3 Line Following Mode

As said above we implemented the Line Following Mode using a state machine do to the given advantages but also to better handle complex behaviors the robot had to do like object avoidance. We also made use of the data calculated on the robot odometry to implement this kind of behaviors.

The resulting State Machine is represented below:

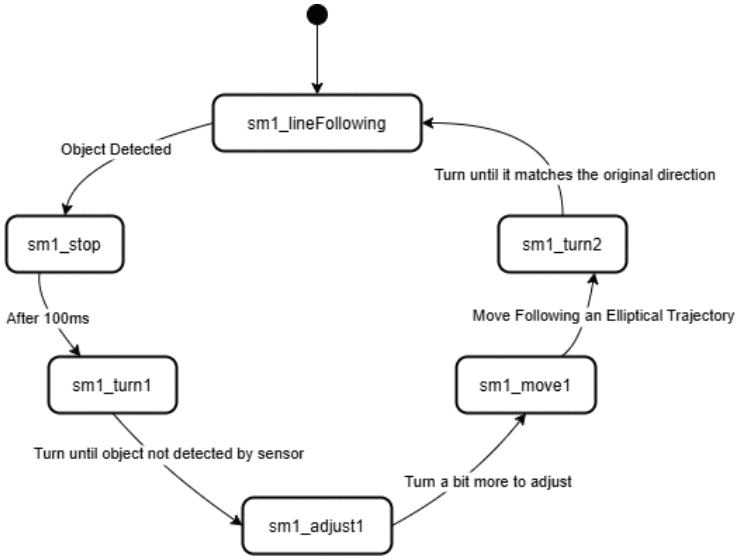

Figure 7: Line Following Mode State Diagram

3.3.1 Object Avoidance:

As shown in the state machine above, when the robot encounters an object in its path, it deviates by following an elliptical trajectory. To achieve this, it calculates the necessary angular velocity using a specific equation. This approach, along with the equation used, is illustrated in the diagram below:

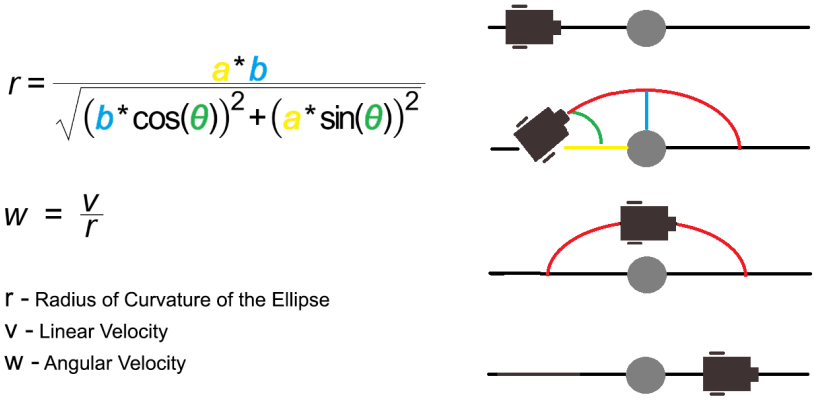

Figure 8: Elliptical Path

3.4 Grid Maze Solver Mode

Similar to the Line Following Mode in the Grid Maze Solver Mode we used a state machine due to similar reasons.

The resulting State Machine is represented below:

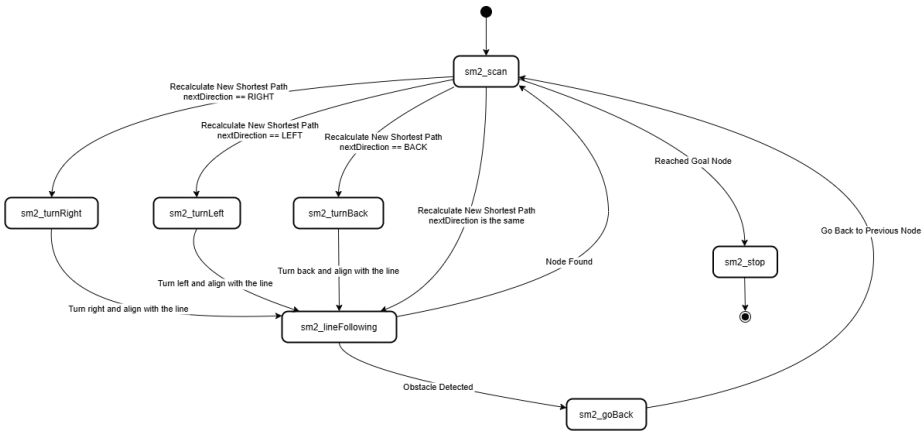

Figure 9: Grid Maze Solver Mode State Digram

3.4.1 Find Shortest Path

When the robot encounters an object, as shown in the state machine, it adds the obstacle to the objects matrix. Upon returning to the last node, the robot must recalculate the shortest path while considering the newly detected obstacle.

To achieve this, we developed a function that utilizes the Flood Fill Algorithm. Flood Fill is not only simple to implement but also more efficient at finding the shortest path compared to basic algorithms like BFS and DFS.

Our function determines the optimal direction for a robot to move toward its goal. It begins by initializing a flood-fill grid, marking empty cells and obstacles accordingly. The flood-fill process starts from the goal position, assigning each reachable cell a distance value that represents the number of steps (or nodes) required to reach the goal. Once the flood-fill completes, the function evaluates the four possible movement directions—left, down, right, and up—from the robot's current position. It then selects the direction with the lowest distance value, ensuring the most efficient path toward the goal while avoiding obstacles.

3.5 Configuration Buttons

To enhance the usability of the final prototype, we added a set of three buttons to configure and control the robot: M, V, and S.

- **M Button:** Toggles the robot's operating mode. By default, it starts in Line Following (LF) mode, but pressing the button switches between LF mode and Grid Maze Solver (GMS) mode.
- **V Button:** Adjusts the robot's speed and corresponding PID values. There are four speed levels, and each press cycles through them sequentially.
- **S Button:** Starts or stops the robot. Additionally, holding the button for three or more seconds connects the Raspberry Pi Pico W to the Internet.

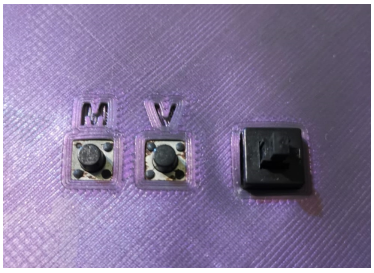

Figure 10: Buttons

These buttons provide a simple yet effective way to control the robot's functionality, making it more user-friendly and versatile.

3.6 Wifi Connection And PID Calibration

To enable dynamic PID calibration, we developed a Python program that connects to the Raspberry Pi Pico W via Wi-Fi, allowing real-time adjustments to Kp, Ki, Kd, and v (linear velocity).

We leveraged the Wi-Fi capabilities of the Pico W, basing our implementation on the provided example code and recommended libraries. To establish communication between the computer and the Pico, we opted for a simple yet effective approach: sending the updated values as a formatted string, which the Pico then interprets accordingly.

On the Python program's side, the interface features sliders to adjust the PID parameters. Each time a slider is moved, the updated value is immediately sent to the Pico. Additionally, the Pico transmits real-time data back to the program, including encoder and sensor readings, motor PWM values, battery voltage, and more.

To further aid PID tuning, the program also displays a graph of PID error over time, providing a clear visual representation of the system's response. This makes fine-tuning the robot's performance easier and more precise. The program's interface is shown below:

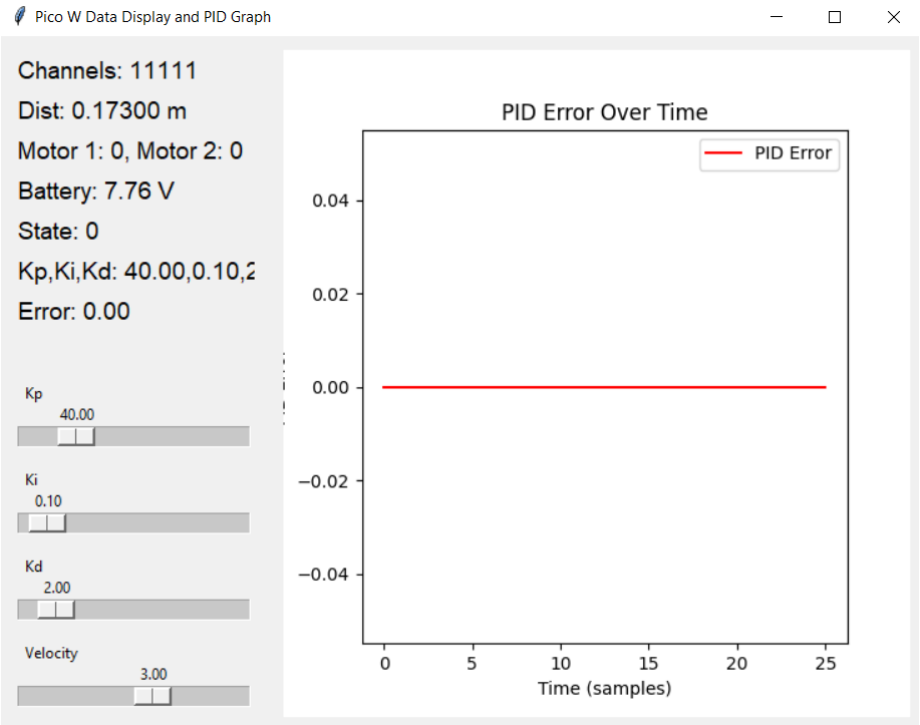

Figure 11: Python Program Interface

4 Summary

In this project, we successfully developed a Line Follower Robot Prototype capable of navigating both line-following tracks and grid-based mazes, while effectively avoiding obstacles. By leveraging embedded computing and robotics technologies, we optimized the robot for both accuracy and speed.

Despite the overall success, we encountered several challenges during development, including sensor misreadings, a limited number of analog ports on the microcontroller, and PID tuning difficulties. To address these issues, we implemented solutions such as analog multiplexing and Wi-Fi integration, which significantly improved performance and functionality.

Future enhancements could include faster motors, more precise obstacle detection and avoidance, and optimized code for higher-speed processing.

Overall, this project demonstrated the practical application of embedded systems and robotics principles, successfully meeting its objectives while laying the groundwork for further improvements.