Advanced Autonomous Systems Lab

Path Control – Final Report

Presented by-

Omri Canfi and Yarden Elnir

Contents

Introduction

The purpose of this report is to present the findings and outcomes of the mission to control the ZumoPi robot using closed-loop algorithms. The objective of this mission was to develop and implement a control system that enables the ZumoPi robot to autonomously perform precise movements, specifically forming a perfect square, circle, and infinity sign. By utilizing closed-loop algorithms, we aimed to achieve accurate and repeatable trajectories, showcasing the capabilities of the robot and highlighting the effectiveness of the control strategies employed.

Goals

1. To design a closed-loop control system for the ZumoPi robot that enables precise and accurate movements.
2. To implement the control system to achieve specific geometric shapes, including a perfect square, circle, and infinity sign.
3. To evaluate the performance of the control system by analyzing the robot's ability to maintain desired positions, follow prescribed paths, and adjust for external disturbances.
4. To explore the limitations and challenges associated with the closed-loop control approach and propose possible improvements.

By addressing these goals, we aimed to enhance our understanding of closed-loop control algorithms and their practical application in robotics. The successful achievement of these objectives would demonstrate the effectiveness of the implemented control strategies and their potential for real-world robotic applications.

ZumoPi Robot

The ZumoPi robot [1]is a fusion of the Zumo robot by Pololu and a Raspberry Pi computer equipped with a camera. This robot was designed and developed by Arkadi Rafalovich. This combination of hardware components provides a versatile and capable platform for our closed-loop control experiments. The following sections detail the features, sensors, and data acquisition capabilities of the ZumoPi robot.

Zumo Robot by Pololu:

The Zumo robot serves as the physical base for the ZumoPi robot.

- It features two independent drive motors, allowing for differential steering and precise maneuvering.
- The robot is equipped with an array of infrared (IR) sensors on its front, enabling proximity detection and obstacle avoidance capabilities.
- It incorporates a line-following sensor array, consisting of multiple reflectance sensors, which enables the robot to follow lines and detect edges.
- The Zumo robot includes an onboard gyro and accelerometer (IMU), providing measurements for orientation and motion detection.

Raspberry Pi Computer:

The ZumoPi robot utilizes a Raspberry Pi computer as its main processing unit. The Raspberry Pi provides computational power and serves as the brain of the robot.

- The Raspberry Pi is equipped with a 8MP camera module that enables visual perception and image processing capabilities.
- It runs a Linux-based operating system, facilitating the implementation of control algorithms and data acquisition.

Data Acquisition:

The ZumoPi robot has the ability to acquire various types of data during its operation.

- Sensor data from the Zumo robot, including infrared proximity measurements, line-following sensor readings, and motion data from the onboard gyro and accelerometer, can be collected.
- Additionally, the camera captures visual data, such as images or video frames, which can be processed and analyzed for closed-loop control.

The integration of the Zumo robot and Raspberry Pi computer, along with the camera module, provides a comprehensive platform for closed-loop control experiments. The combination of motor control, proximity sensing, line following, and visual perception capabilities enables the ZumoPi robot to interact with its environment and execute precise movements as required for our mission.

[1] More information and documentation about the ZumoPi can be found in the Git Repository - https://github.com/TAU-Robotics/ZumoPi

Theoretical background

Odometry

Odometry is a technique used to estimate the position and orientation of a robot by analyzing the motion and measurements of its wheels or other motion sensors. In the case of the ZumoPi robot, odometry plays a crucial role in determining its position and trajectory as it moves. This section provides an overview of odometry and its importance in robot navigation.

Principles of Odometry

Odometry relies on the assumption that the motion of a robot can be accurately represented by the motion of its wheels or other motion sensors. By measuring the distance traveled and the change in orientation of these motion sensors, it becomes possible to estimate the robot's displacement.

The most common approach (and the method that we used) to odometry is based on the use of wheel encoders. Wheel encoders measure the rotation of the robot's wheels, providing information about the distance traveled. By combining the encoder data from both wheels, the robot's forward motion, rotation, and overall displacement can be calculated.

Odometry Error and Correction

While odometry provides a relatively simple and cost-effective way to estimate robot motion, it is prone to error accumulation. Factors such as wheel slippage, uneven terrain, and inaccuracies in the encoder measurements can lead to significant discrepancies between the estimated and actual robot positions.

To mitigate these errors, various techniques can be employed. One common approach is to use sensor fusion, where odometry data is combined with information from other sensors, such as an inertial measurement unit (IMU) or visual sensors like the camera on the ZumoPi robot. Sensor fusion algorithms, such as the Extended Kalman Filter (EKF) or the Particle Filter, can improve the accuracy of odometry estimates by integrating multiple sources of information.

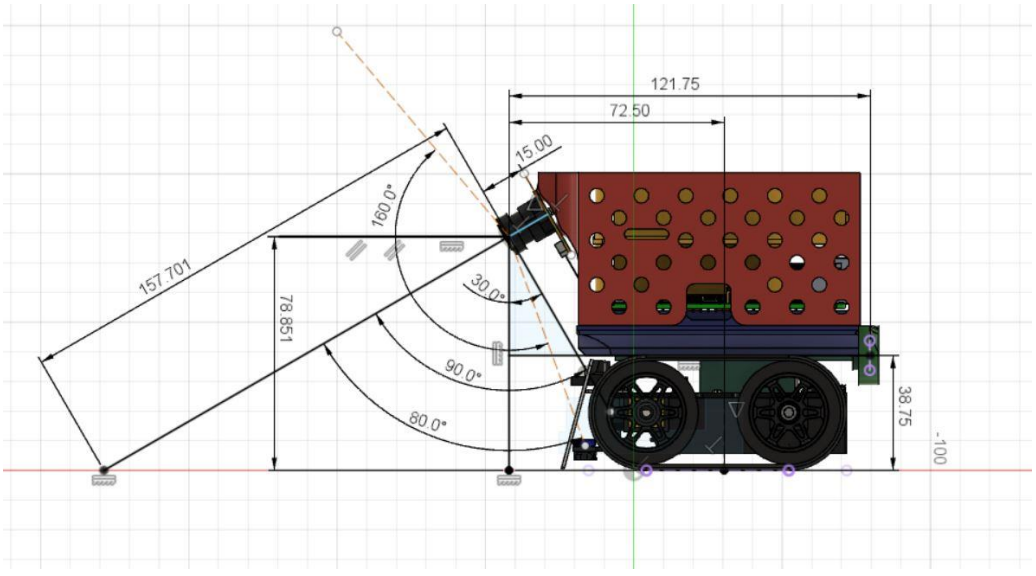

Figure 1 - ZumoPi Dimensions

Point-to-Point (P2P) Algorithms

Point-to-Point (P2P) algorithms, also known as path planning algorithms, are used to determine the optimal path or trajectory for a robot to reach a specific target or destination. These algorithms are essential for guiding the ZumoPi robot in accomplishing its predefined tasks of moving along specific paths to create shapes. This section provides an overview of P2P algorithms and their significance in robot control.

Path Planning in the ZumoPi Robot

In the case of the ZumoPi robot, P2P algorithms are crucial for generating the desired paths to create shapes accurately. By defining the target coordinates, a suitable P2P algorithm can be employed to compute the optimal path that the robot should follow.

To achieve smooth and precise movement, the P2P algorithm can be combined with the odometry data obtained from the wheel encoders and other sensors. By continuously updating the estimated position and comparing it with the desired trajectory, the robot can make real-time adjustments and corrections to ensure it accurately follows the planned path.

The successful implementation of P2P algorithms in the ZumoPi robot's control system enables it to navigate complex environments and execute tasks requiring precise motion, such as creating shapes with minimal errors and achieving the desired patterns.

Project Block Diagrams

ZumoPi Block Diagram

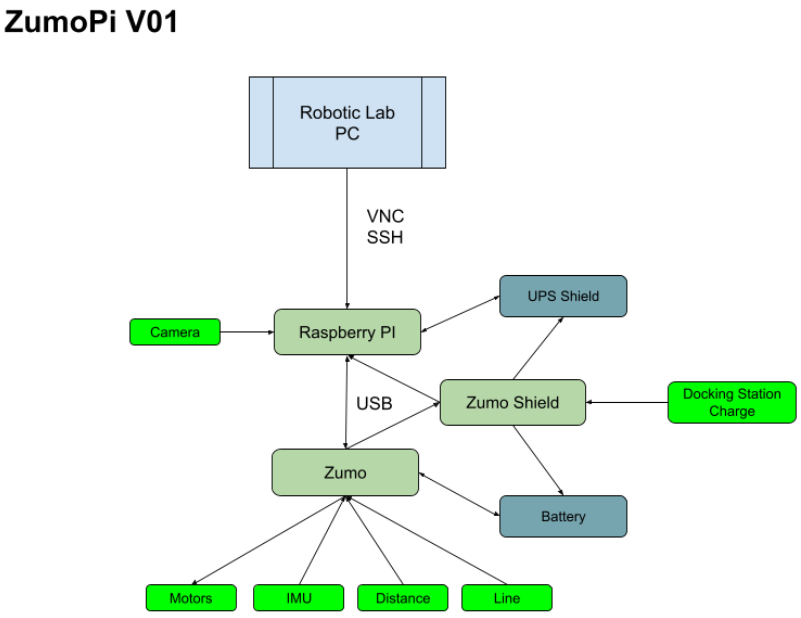

Figure 2 - ZumoPi Block Diagram

The block diagram of the ZumoPi robot illustrates the interconnected components and their connections for efficient control and operation. The diagram depicts a couple of elements, we will focus on the one we used:

1. PC: The ZumoPi robot is remotely controlled and managed by a PC through a Secure Shell (SSH) connection. This connection allows the PC to communicate with the Raspberry Pi, which serves as the main control unit for the robot.

2. Raspberry Pi: The Raspberry Pi acts as the central processing unit for the ZumoPi robot. It is connected to various peripherals and sensors to facilitate the robot's functionality. The Raspberry Pi is responsible for executing control algorithms, processing sensor data, and coordinating the overall operation of the robot.

3. Zumo Robot: The Zumo Robot, manufactured by Pololu, is a key component of the ZumoPi robot. It is connected to the Raspberry Pi via a USB connection, establishing communication and control between the two. The Zumo Robot integrates various essential features for robot locomotion and sensing.

4. IMU: The Zumo Robot incorporates an Inertial Measurement Unit (IMU), which consists of sensors such as accelerometers, gyroscopes, and magnetometers. The IMU provides information about the robot's orientation, acceleration, and angular velocity, aiding in navigation, motion control, and stabilization.

5. Motors: The Zumo Robot utilizes motors for its locomotion. The motors are responsible for driving the wheels, enabling the robot to move forward, backward, and turn. By controlling the motors, the ZumoPi robot can execute precise movements and follow predetermined paths.

Software block diagram

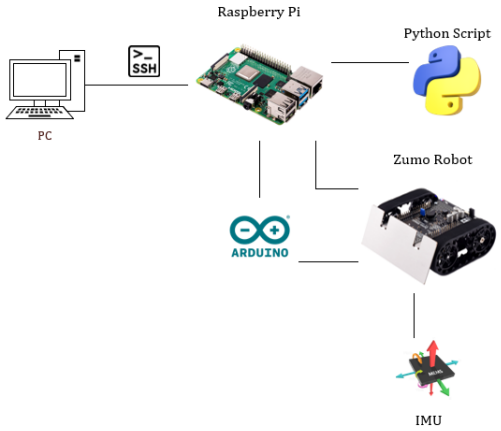

Figure 3 - Software Architecture

1. SSH Connection: A Secure Shell (SSH) connection is established between PC and the Raspberry Pi. This connection allows remote access to the Raspberry Pi's terminal and facilitates the control and monitoring of the ZumoPi robot.
2. Terminal: Through the SSH connection, the terminal on the Raspberry Pi is accessed. The terminal provides a command-line interface where various commands and scripts can be executed.
3. Arduino GUI: Within the terminal, the Arduino GUI is launched. The Arduino GUI is a development environment that allows programming and uploading code to the Arduino microcontroller, which is an integral part of the ZumoPi robot.
4. Arduino Script: Using the Arduino GUI, a script (C^{++}) is written to control the motors and obtain sensor data. The script sends commands to the motors to control the robot's movement, and it retrieves the calculated angle from the gyro sensor and the x and y position from the odometry calculations. This data is then printed to the serial monitor, which displays information on the terminal.
5. Python Script: On the PC, a Python script is created to set desired points for the robot to follow. These desired points represent the desired path or trajectory for the ZumoPi robot. The Python script communicates with the Raspberry Pi via the SSH connection.
6. P2P Algorithm: The Python script utilizes a Point-to-Point (P2P) algorithm to determine the desired motion of the robot based on the desired points set in the previous step. The P2P algorithm calculates the necessary control signals or commands to guide the robot along the desired path, taking into account its current position and orientation.
7. Serial Communication: Serial communication is established between the Raspberry Pi and the ZumoPi robot. The Python script sends commands and receives messages through this serial communication interface. It sends instructions to the ZumoPi robot, such as the desired points and control signals calculated by the P2P algorithm. It also receives feedback or sensor data from the robot, allowing for real-time interaction and control.

By leveraging the SSH connection, Arduino GUI, Python scripting, and serial communication, this block diagram demonstrates an effective code environment for controlling the ZumoPi robot. It enables the integration of motor control, sensor data acquisition, path planning, and feedback communication, facilitating the execution of precise and autonomous robot motions

Implementation

The complete code we developed to achieve the project's goal can be accessed in the following repository: https://github.com/canfick/ZumoPi_PathControl. Now, let's provide a description of what occurs in the code-

The Python script consists of several functions and a main control loop. Here is a brief overview of the main components and their functionality:

1. **P2P_CTRL_NEW**: This function implements a point-to-point control algorithm. It takes the desired position, current position, heading angle, and maximum acceleration as inputs and calculates the forward velocity, desired heading angle change, distance error, remaining distance, and a finished flag.
2. **pid_controller**: This function calculates the control signal using the PID controller formula, which takes the error, integral term, and derivative term as inputs.
3. **control_loop**: These functions implement a control loop that takes the target velocity, actual velocity, and time step as inputs. They calculate the error, integral, derivative terms, and call **pid_controller** to compute the control signal.
4. **parse_arduino_line** and **print_arduino_line**: These functions read and parse data from the Arduino board through a serial connection.
5. The main program starts by initializing the serial connection with the Arduino board. It then defines a path for the robot to follow by specifying a sequence of points. It sets some constants and starts a control loop to regulate the robot's motion. Within the control loop, it receives sensor data from the Arduino board, calls **P2P_CTRL_NEW** to calculate the desired velocities, performs scaling and normalization, applies speed limits, and sends the control signals to the Arduino board.

On the other hand, the provided C++ script for the Arduino board includes code for integrating odometry and gyro data to estimate the robot's position and heading angle. Here is a brief overview of the main components:

1. The script includes the necessary libraries for working with the Zumo32U4 robot, including encoders, motors, and the inertial measurement unit (IMU).
2. It defines a sampling rate, **SAMPLERATE**, which determines the frequency of data acquisition and control loop execution.
3. The script initializes the communication with the Zumo32U4 board and sets up the main loop.
4. Within the loop, it reads data from encoders and the IMU to obtain odometry and gyro data. The time elapsed since the last loop iteration is calculated to determine the time step, **dt_time**.
5. The script also includes code to handle serial communication with the Python script. It reads commands and parameters sent from Python, which include target velocities for the left and right motors.

6. Finally, the script uses the motor controllers to set the desired speeds for the left and right motors based on the received target velocities.

In summary, the Python script performs the high-level control of the robot's motion, while the C++ script running on the Arduino board handles the low-level control of the motors and integrates sensor data for odometry and gyro estimation. The two scripts work together to enable the robot to follow a predefined path while maintaining stable control using PID control.

Output examples:

To visualize the data obtained from the Arduino board, we utilized the parse and print Arduino line functions. Below is an example of the initial message printout after initializing the board:

```
Left Motor|Right Motor|Battery|dt_time|pos x|pos y|ody ang|gyro ang
0.00      |0.00       |3.82   |0.00   |0.00 |0.00 |0.00   |0.00
```

Upon examining the output, we can observe the following information: left and right motor speeds, battery level, dt_time, x and y positions, gyro data, and odometry angle.

Results

In our code, we have implemented a couple of functions that read the output text file. This allows us to obtain all the necessary data for debugging and presenting the results of the algorithm, as well as tracking the robot's movement.

Robots path

The following plot shows the actual path (on x and y axis) as calculated by the odometry, while trying to draw a square.

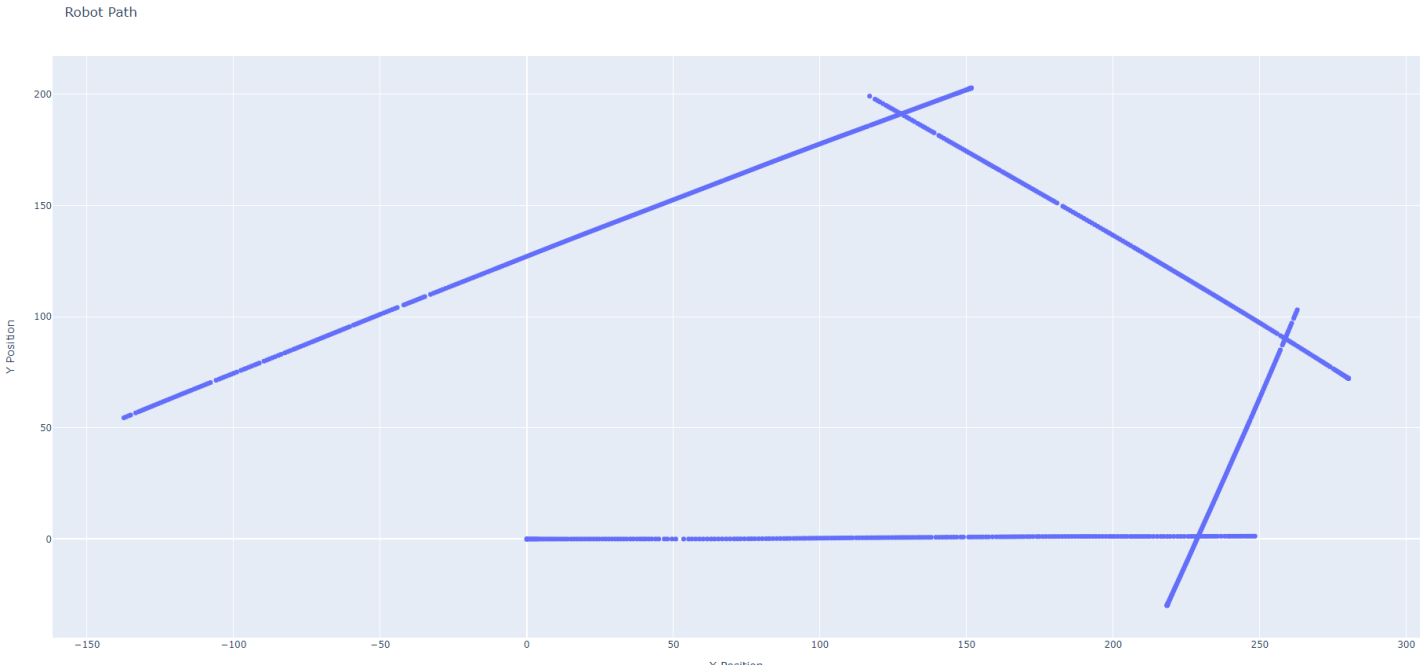

Figure 4 - Robots Path

One can clearly see that the square is not a perfect square, but we love him as it is. Is a special square. The actual square that was drawn by the robot in real-life was a much better one, the difference is caused by slipping and miscalculation.

Fixing direction

The following plot shows the robots X and Y calculated position between the starting point and the first corner. Also we can see the calculated angle the velocity difference $(V_{left} - V_{right})$

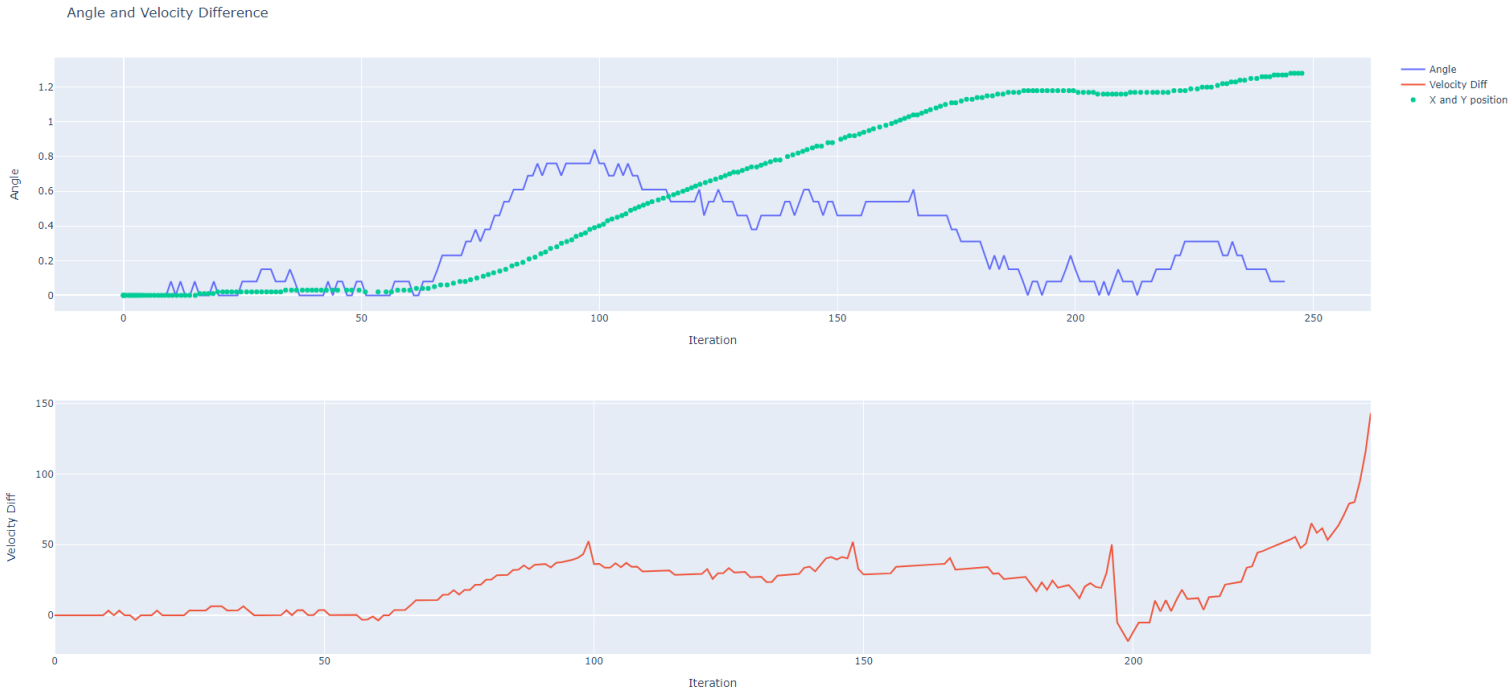

Figure 5 - Angle and Velocity correction

We can see that the robot recognize that it is not moving in a straight line, and he is trying to fix that by adjusting the speed of the motors accordingly. For example, around iterate 100 we can see that the angle is increasing (the robot moves in the y axis as well) so the robot gives more power to the left motor in order to move back in line.

Reaching Points

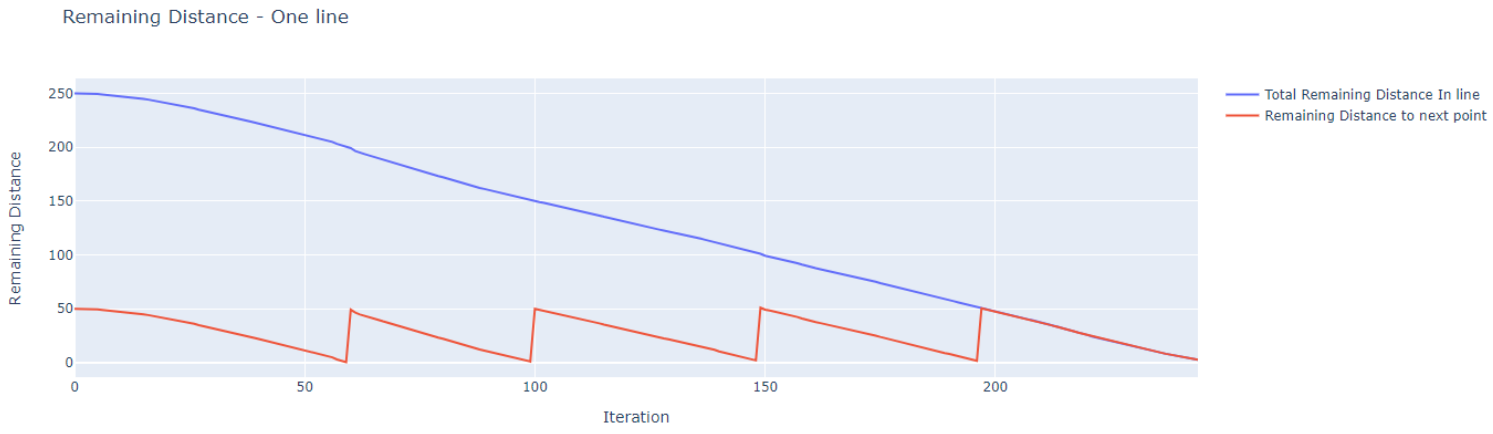

Figure 6 - Remaining distance

In the above graph, we have two lines: the red line represents the remaining distance calculated from the norm of the vector between the robot location and the next point location. The blue line represents the total remaining distance of the line.

We have set a threshold, below which the robot needs to go in order to move to the next point. This threshold is visualized in the following plot:

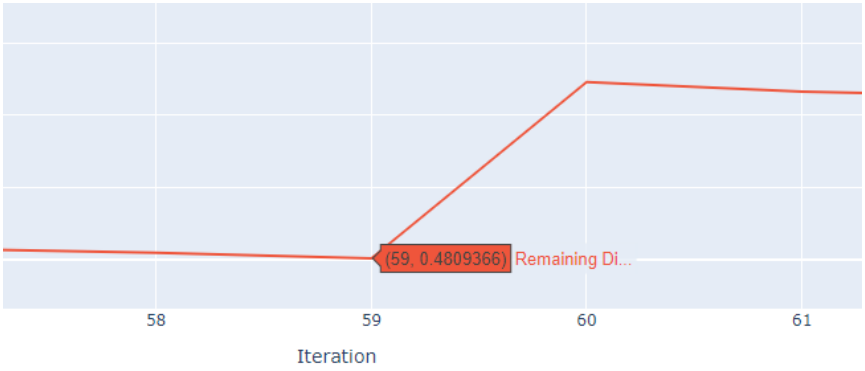

Figure 7 - Threshold example

Conclusion

In conclusion, this project aimed to develop a robust path control system for a robot using the ZumoPi platform. Through the implementation of various algorithms and sensor integration, we partially achieved our goal. The developed code, available in the provided repository, allowed us to effectively interpret data from the Arduino board, enabling us to analyse motor speeds, positional information, and other relevant parameters.

By leveraging the output text that exported from the script, we were able to visualize the obtained data and gain insights into the performance of our algorithm. Additionally, the functions implemented for reading the output text file facilitated debugging and provided a comprehensive view of the robot's movement.

Overall, this project not only showcased the successful integration of software and hardware components but also highlighted the importance of efficient data interpretation and analysis.

We had a great time in this lab, as it presented us with a real engineering challenge that required a broad set of skills. From coding and algorithm design to hardware integration and data analysis, this project encompassed various aspects of engineering.

Thanks,

Omri and Yarden.