Final Report – Autonomous Systems Lab

Idan Hagai – 209233998

Carmel Amit – 318733029

ChatGPT was utilized during this project.

Introduction

Autonomous systems are engineered to independently execute tasks by sensing their environment, making decisions, and taking actions. These systems are widely applied in navigation, obstacle avoidance, and task-specific operations. A line-following robot is a fundamental example of such systems, employing sensors to detect paths, algorithms to process data, and actuators to execute movements.

The Zumo robot is a flexible platform built for educational use, featuring a range of sensors and actuators to support autonomous navigation tasks. This project involves designing and implementing an autonomous robot capable of navigating along a black line, bypassing obstacle, and executing predefined actions at specific points. The task highlights key principles of autonomous control, including sensor integration, obstacle avoidance, and route planning.

Line following is a fundamental task in robotics where a robot uses sensors to detect and follow a path, typically a black line on a contrasting surface. The Zumo robot utilizes infrared (IR) sensors to identify the contrast between the line and the surrounding surface, allowing it to adjust its motion to maintain alignment with the path. This involves analyzing sensor readings to determine the robot's position relative to the line and modulating motor speeds to ensure accurate tracking.

Tasks such as line following and obstacle avoidance rely on the effective combination of hardware and software elements, including precise sensor adjustments, motor control strategies, and the implementation of algorithms. Additionally, optimizing parameters like PID settings, data sampling rates, and route planning is critical to achieving accurate performance and dependable navigation in a variety of conditions.

The theoretical basis of this system encompasses concepts of sensor-driven navigation, control systems, and path planning, which are fundamental for enabling autonomous functionality in mobile robotic platforms like the Zumo.

System Overview

The robot system is implemented using the Zumo 32U4 platform, which integrates an Arduino-compatible ATmega32U4 microcontroller. Key components include:

1. **Line Sensors**: To follow the black line.
2. **Proximity Sensors**: To detect obstacles in the robot's path.
3. **Motor Encoders**: For accurate movement control.

The task is divided into the following stages:

1. **Line Following**: The robot follows the black line using its line sensors.
2. **Obstacle Detection and Bypass**: Upon detecting an obstacle using proximity sensors, the robot follows a predefined bypass route to return to the black line.
3. **Reaching the Endpoint and Returning**: After reaching the endpoint of the black line, the robot executes a predefined route to navigate back to the starting point.
4. **Second Line Navigation**: The robot starts along the black line again, following it until it encounters the obstacle a second time.
5. **Parking Maneuver**: When the robot detects the obstacle for the second time, it executes a predefined route to park in the designated parking area.

To accomplish the task, the system combines sensor input, motor control, and path-planning algorithms to enable accurate navigation. While following the line, the robot adjusts motor speeds in response to sensor feedback to maintain alignment with the black line. For obstacle avoidance and navigation, the robot employs predefined routes to maneuver around obstacles and travel between target points. The solution integrates PID-based algorithms for error correction, sensor calibration for precise detection, and real-time tracking of the robot's position and speed. Additionally, the code includes logic to seamlessly switch between line following, obstacle bypassing and navigation tasks, ensuring smooth transitions based on sensor data.

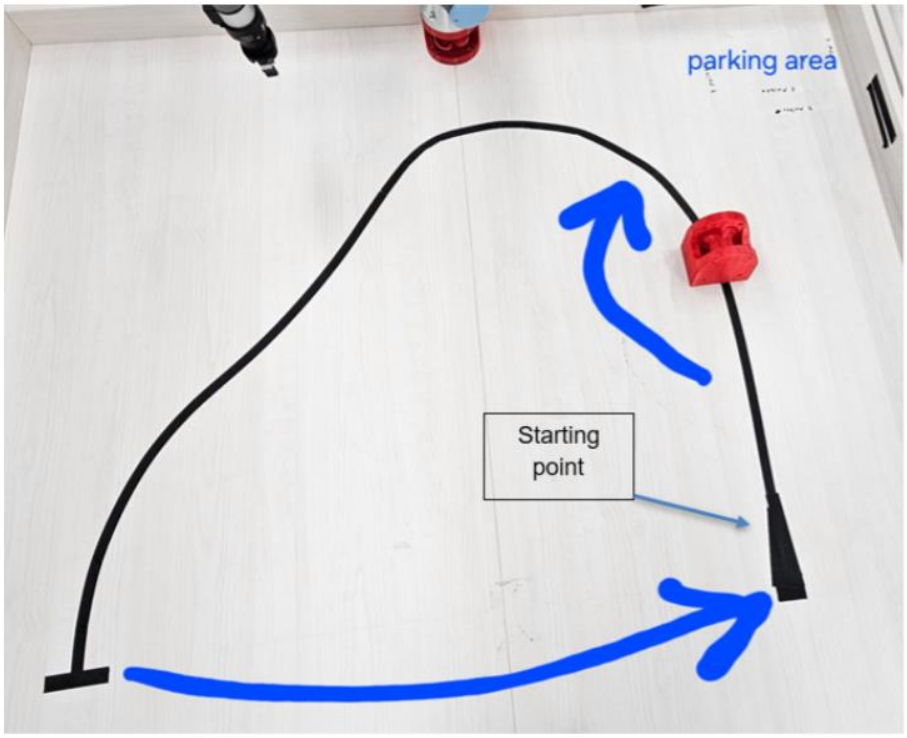


Figure 1 – route description

Block diagram of the system:

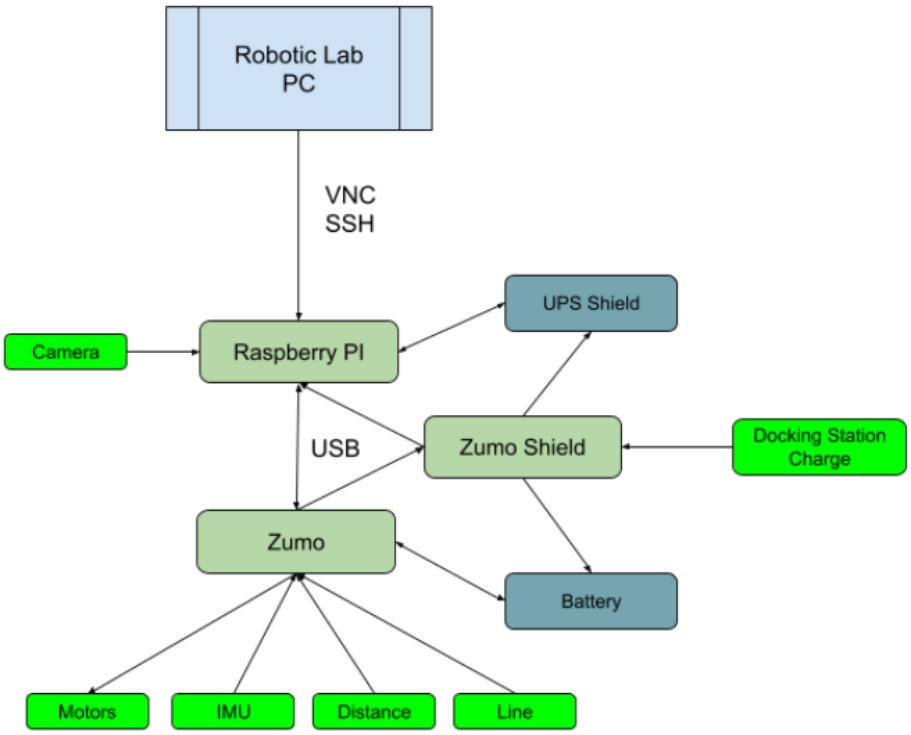


Figure 2 - Block diagram of the system

Block diagram of the task:

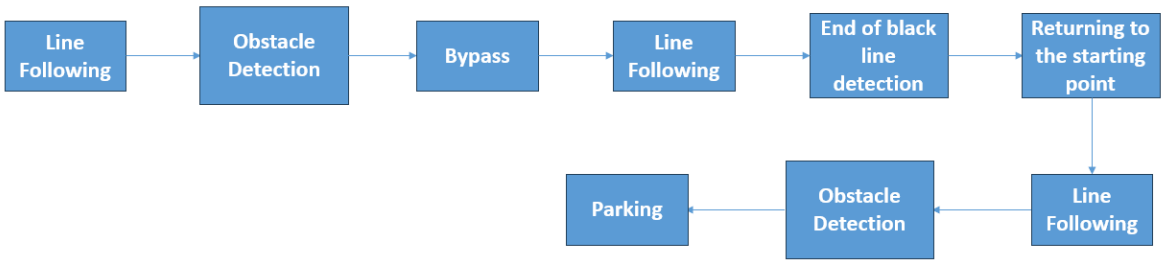


Figure 3 - Block diagram of the task solution

The course of the experiment

Line Following Integration: Incorporated code from an existing example to read sensor data and control motor speeds, ensuring the robot stayed aligned with the black line. This included implementing a PID-based algorithm for error correction, while adjusting proportional, integral, and derivative gains to achieve optimal performance within the overall system.

Obstacle Detection: Integrated logic to detect obstacles based on sensor readings. Established specific thresholds to accurately identify obstacles and initiate the bypass sequence when these thresholds were met.

Returning to Line Following Logic: After completing the bypass sequence, a while loop was implemented to monitor the IR sensor readings. When the sensors detected predefined thresholds values indicating the black line, the robot seamlessly resumed line-following behavior.

Detecting the End of the Black Line: To identify the end of the black line, the robot utilized its line sensors to monitor predefined values indicating the transition point. The sensor readings were analyzed to detect a specific condition that signaled the end of the line. Once this condition was met, the robot executed a predefined logic sequence to navigate back to its starting point. Upon reaching the starting point, the robot seamlessly resumed line-following behavior, ensuring continuous and efficient task execution.

Second Identification of the Obstacle and Parking: The same logic used for detecting the obstacle during the first encounter was applied for its identification during the second time. To differentiate between the two occurrences, a boolean flag was implemented to track whether the obstacle had been previously encountered. Upon detecting the obstacle for the second time, the robot executed a predefined parking routine instead of bypassing it. At the end of the parking routine, the code entered an infinite while loop, effectively causing the robot to halt its operations.

Testing and Optimization: Performed rigorous testing to evaluate the system's performance. Fine-tuned parameters such as sampling rates, and trajectory coordinates to improve navigation accuracy and efficiency.

By following these steps, the Zumo robot was successfully programmed to autonomously navigate along a predefined route, follow a black line, and avoid obstacles.

Challenges Encountered During Implementation

Integrating the different parts: One of the primary challenges was effectively incorporating the different algorithms into the system. This required designing robust navigation logic to maneuver around obstacles and returning to line following at the right moment afterwards.

Hardware Issues with the Zumo Robot: Hardware problems with the initial Zumo model presented significant hurdles. Switching to a different Zumo robot required modifying the code and conducting additional testing to confirm compatibility and consistent operation.

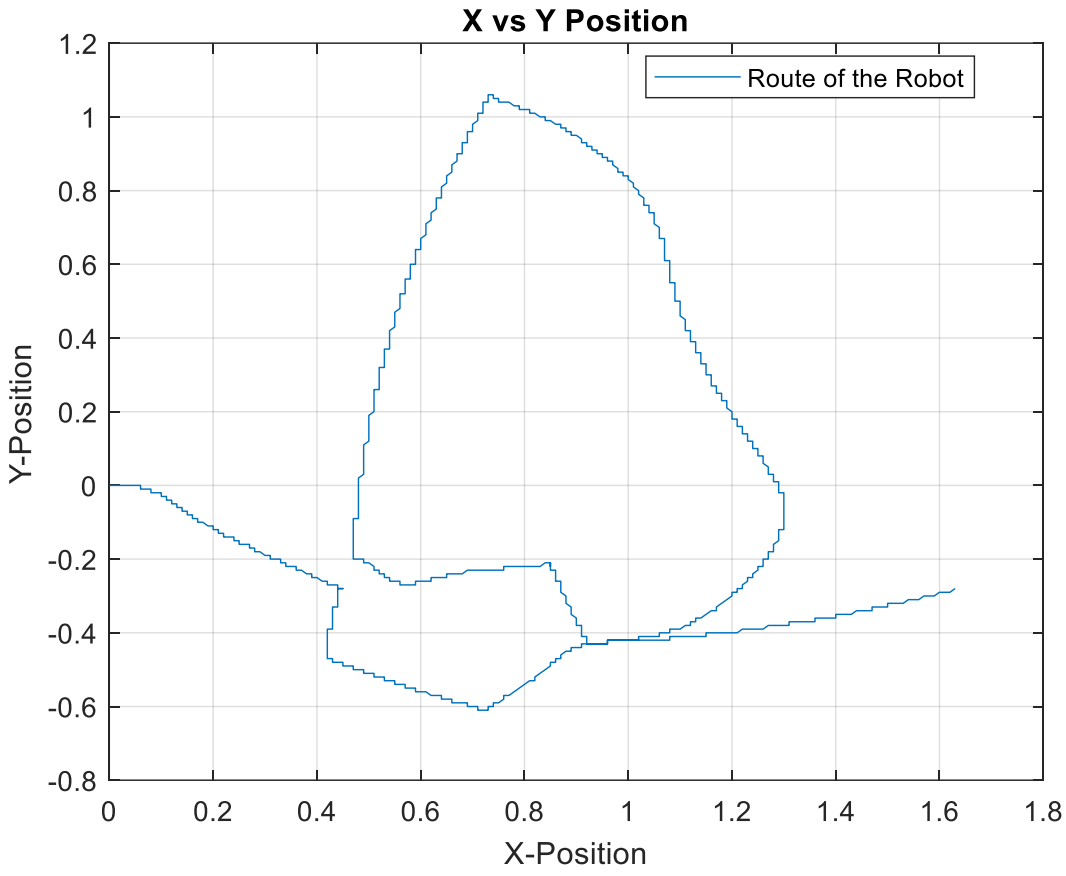
Addressing Challenges

To overcome these obstacles:

- **Code Refinement**: Debugged and fine-tuned the algorithms to ensure smooth transitions and reliable performance.
- **Iterative Testing**: Conducted multiple test cycles to identify issues, adjust parameters, and validate the functionality of both hardware and software.

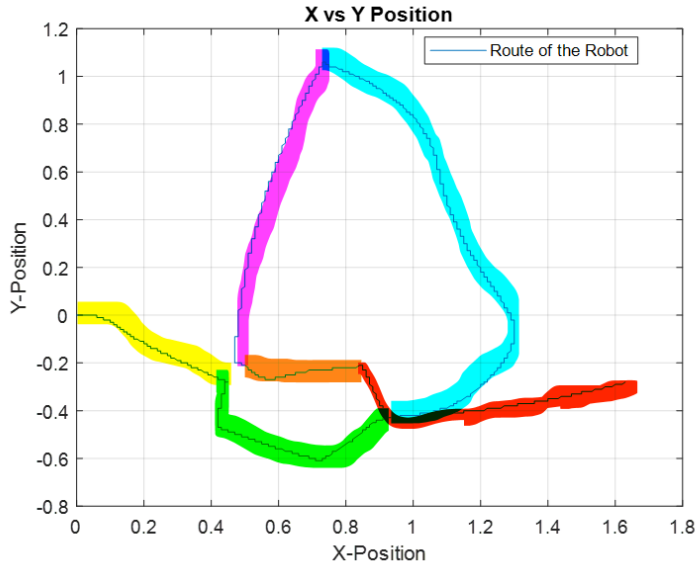
Through persistent troubleshooting, optimization, and testing, the project successfully achieved its goal of enabling the Zumo robot to autonomously navigate a black line, bypass obstacles, and complete predefined navigation tasks.

Results:



Figture 4 – X,Y position of the robot

Now we will explain about the X,Y graph:



Figture 5 – X,Y position of the robot with markers

From figure 5, we can observe the distinct segments of the robot's route. The **yellow segment** represents the robot's line-following function as it navigates the black line until the obstacle is detected. In the **green segment**, the robot executes obstacle avoidance and returns to the black line. The **blue segment** shows the robot resuming line-following behavior. The **pink segment** illustrates the robot's movement from the endpoint of the black line back to the starting point using a predefined route. In the **orange segment**, the robot performs line following once again until the obstacle is detected a second time. Finally, the **red segment** depicts the robot's obstacle avoidance maneuver followed by its predefined parking routine.

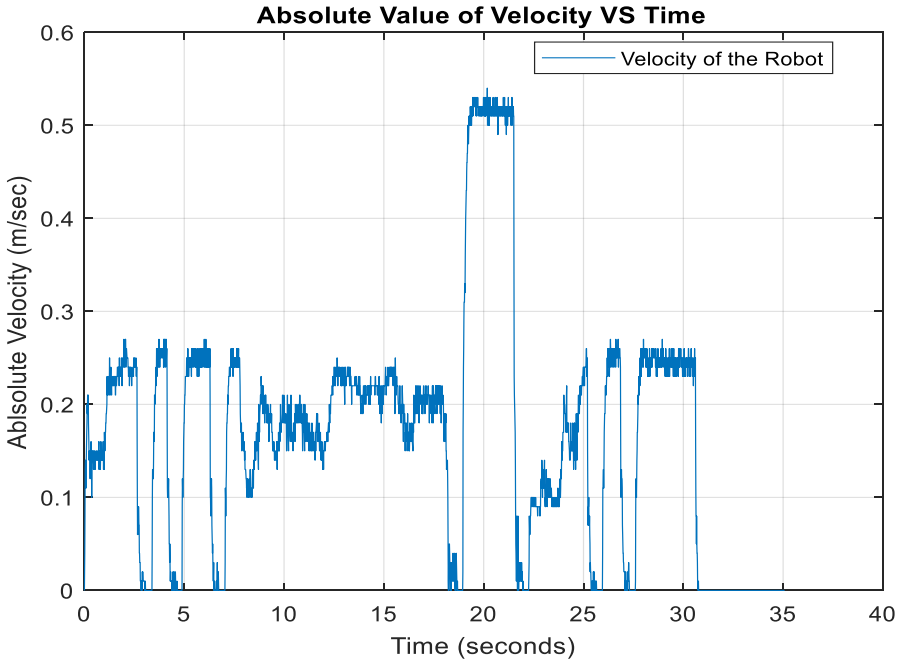


Figure 6 – Absolute value of velocity over time

In Figure 6, we observe the absolute value of the robot's velocity across different segments of its route. During the line-following phases, the robot's speed fluctuates as the controller continuously adjusts to keep the robot aligned with the black line. In the obstacle avoidance segment, significant changes in velocity are evident as the robot stops and executes a rectangular movement to bypass the obstacle.

When navigating back to the starting point, the robot's velocity increases noticeably and remains constant throughout this phase. This consistency is due to the use of an open-loop technique for this part of the route. Similarly, an open-loop technique was employed during the parking phase, resulting in a steady velocity during the final segment of the graph. At the end of the route, the robot comes to a complete stop, and its velocity drops to zero.

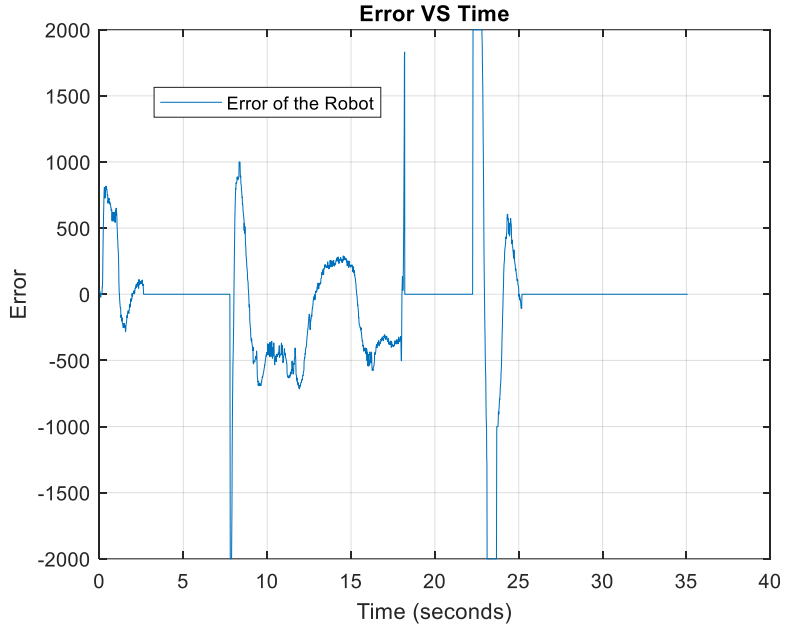


Figure 7 – Position deviation from the black line

In Figure 7, we observe the error from the black line generated by the controller during the line-following phases. This controller is active only while the robot is performing line-following tasks, adjusting the robot's movements to stay aligned with the black line. When the robot is not driving along the black line, such as during obstacle avoidance or predefined navigation, the error value drops to zero as the controller is not in operation.

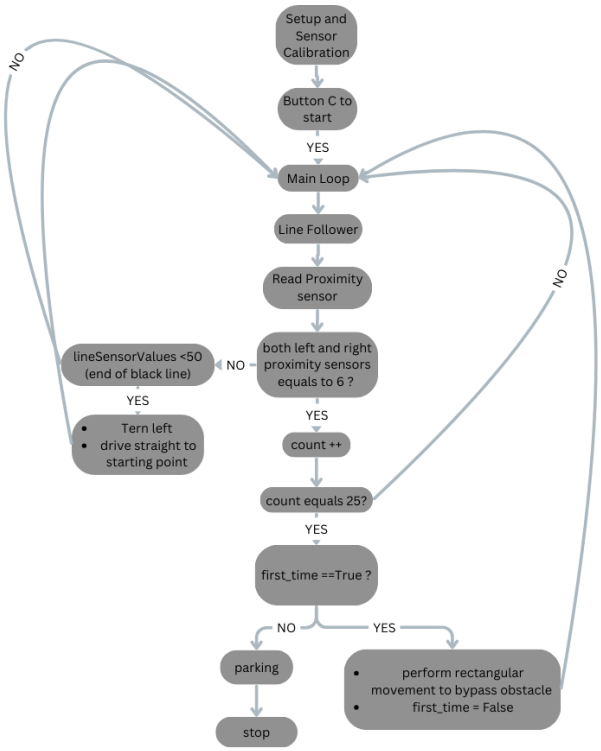


Figure 8 – block diagram of our code

Difficulties and Solutions:

- Difficulties that arose during the work process:

Transitioning from obstacle avoidance back to line following:
When the robot detected an obstacle, it switched from line following to an open-loop maneuver, executing a predefined rectangular movement to bypass the obstacle. The challenge arose in ensuring that the robot would end its movement precisely on the black line so it could detect it and resume line following. This precision was affected by the state of the battery, which influenced the robot's movement distance and consistency. As a result, it was difficult to reliably align the robot with the black line at the end of its bypass maneuver.

Data Recording:
Our robot relied on predefined maneuvers to bypass obstacles and navigate from the end of the black line back to its starting point. To achieve this, we programmed the robot to execute specific left and right turns using the delay function while maintaining a constant motor speed. However, during the execution of the delay function, the code would pause, preventing any further commands from being executed. This presented a challenge, as we needed to continue recording the robot's state (e.g., position and speed) while ensuring the predefined movements were carried out accurately.

- Solutions and Workarounds for the Identified Difficulties:

Transitioning from obstacle avoidance back to line following:
To address this challenge, we utilized the line sensor values array to detect the black line more effectively. When the robot performed its final straight movement toward the black line, it entered a while loop, allowing it to continue driving straight until the black line was detected. We set a threshold value of 500 for the sensor readings to reliably identify the black line. This approach eliminated the need to rely on the delay function to precisely end the robot's movement on the black line, improving both accuracy and consistency.

Data Recording:
To address the challenge of maintaining robot state recording during predefined maneuvers, we developed two functions: *odometry_for_delay* and *odometry_without_delay*. The *odometry_for_delay* function replaces traditional delay commands, allowing the robot to perform a delay while continuously recording its state. It takes the desired delay duration as input, calculates elapsed time using a loop, and updates the robot's odometry at regular intervals defined by the *SAMPLERATE*. During each interval, the robot's position, velocity, and timestamp are recorded. On the other hand, the *odometry_without_delay* function is used to record the robot's state during line-following movement, where no delay is needed. In this case, it updates the robot's odometry and records state information, including position, velocity, and the controller's error term, ensuring accurate tracking of the robot's behavior in real-time. These functions allowed seamless integration of odometry updates and state recording during all parts of the code.

Discussion:

The lab experience was highly enriching and enjoyable, as it provided an opportunity to work independently while exploring new concepts and platforms. We gained hands-on experience with Arduino and Raspberry Pi, along with a deeper understanding of autonomous systems, driving concepts, and utilizing data from the Zumo32U sensors to perform autonomous driving tasks. We particularly appreciated the chance to apply the theoretical knowledge we acquired to a real-world project, demonstrating the integration of learned materials in a practical and meaningful way.

The Zumo32U platform offers significant potential for practical applications in various fields, demonstrating the versatility of autonomous robotic systems. In **smart agriculture**, this platform could be developed for tasks such as navigating crop rows to monitor soil conditions, assess plant health, or perform targeted weeding, thereby reducing the dependency on human labors. For **home and consumer applications**, the platform could serve as a foundation for creating autonomous systems like robotic vacuum cleaners or small-scale delivery robots for transporting items within homes or offices. In the **military domain**, the Zumo32U could be adapted for reconnaissance missions, enabling it to navigate hazardous areas, gather critical data, or deliver supplies to locations inaccessible or dangerous for humans. However, to perform these advanced operations, the Zumo32U platform would require further development, including the integration of additional hardware and sensors, such as environmental sensors for agriculture, object detection modules for navigation, and enhanced power systems for extended operation.

For future improvements, we would suggest replacing the open-loop maneuvers with a closed-loop path control system utilizing PID control. This enhancement would mitigate the effects of battery state on the robot's movement, ensuring greater consistency and precision. Additionally, it would enable the robot to perform more complex tasks, such as dynamic obstacle avoidance and navigation in more intricate environments, significantly improving its overall capability and adaptability.