Autonomous Vehicle Challenge Final Report

Jessica Meyer
Student ID: 300672526
ENGR101 - Engineering Technology
Lecturers: Howard Lukefahr, Arthur Roberts, James Quilty
Date: 2nd June 2025
https://gitlab.ecs.vuw.ac.nz/course-work/engr101/2025/project3/team14/avc

1. Abstract

A small autonomous vehicle was designed and developed to navigate a course using visual input and programmed decision making. The robot was built with a Meccano chassis, a Raspberry Pi Zero W, a Pi Camera Rev 1.3 and micro servo motors. To enable the robot to follow the track and detect coloured markers, image processing and control logic was implemented with C++. A custom 3D printed part was designed to improve the range of the camera motor, and a simplified PID control system was used to ensure line following accuracy. The robot was able to successfully demonstrate autonomous navigation through the track, showcasing the effectiveness of the hardware and software components.

2. Introduction

This report covers the construction and programming of an autonomous vehicle built to navigate a track (Fig. 1) using a camera input. While this report covers the development of software and testing, technical theory is excluded. Autonomous vehicles are a significant area of robotics with real-world applications in self-driving cars and automation. This project explores how a robot can use visual input to follow a line and respond to coloured pillars to make decisions while navigating a course. The aim of this project was to design, build, and program an autonomous vehicle capable of completing a track without human assistance.

Objectives:

- *Specific*: Design and build a durable robot using Meccano and 3D-printed components, suitable for navigating a defined track autonomously.
- *Measurable*: Ensure the robot can complete at least one full run of the track without human intervention, responding correctly to line direction and coloured pillars.
- *Attainable*: Use a Raspberry Pi Zero W and Pi Camera Rev 1.3 to implement image processing and control logic within the limitations of the provided hardware and time frame.
- *Relevant*: Focus on integrating image-based navigation and decision-making, which reflects real-world autonomous vehicle challenges.
- *Timely*: Complete the design, construction, coding, and testing of the robot within a four-week development period.

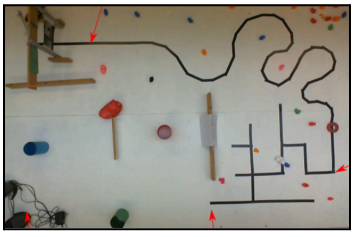

Fig.1. Robot Track

3. Background

Autonomous robots must be able to gather information and data available about their surroundings to be able to make decisions and navigate through environments without human input or interaction. A common application of this can be seen in warehouses and factories, where line following robots can be used to replace manual transportation methods such as forklifts and some cranes [1]. These systems typically rely on a range of different sensors that are sending input to the robot allowing it to respond with appropriate movements and actions.

In this project, the autonomous vehicle was equipped with a camera, allowing it to process image based input and make accurate movement based on visual data. The robot was designed to perform three key tasks: follow a line, navigate through a maze and detect and move towards coloured pillars [2]. These tasks simulate common tasks needed in industrial robots, where such capabilities are essential. All tasks relied heavily on the robots ability to interpret given real-time information which was achieved through the usage of the camera and image processing.

While image processing was chosen primarily because it was the only input method available for this project, it proved to be highly effective for each task carried out. Image processing allowed for the ability to have an advanced perception such as identifying the position of the line or detecting specific colours within the camera's frame. This mirrors other real life autonomous applications that rely on computer vision for navigation and decision making [3].

The robot was programmed in C++ as it is a very commonly used language for robotics due to its speed and ability to work with limited hardware. The custom library E101.h was also used to provide custom functions to work with the camera and robot hardware.

4. Method

4.1. Robot Body and Hardware

The robot's chassis was constructed from Meccano, allowing for a modular robot design should anything need to be changed during development. Building with steel Meccano parts also made the robot incredibly robust, ensuring minimal damage to the robot in the case of any accidents occurring, such as falling off the table during testing. During testing it was discovered that the camera motor lacked sufficient range of motion. To resolve this, a custom 3D-printed camera motor mount was designed (Fig. 2, 4), allowing for improved camera motion.

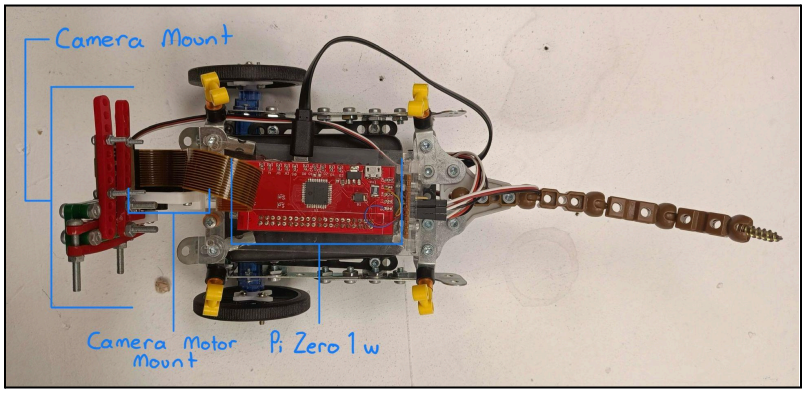

Fig. 2. Top View of Robot

Attaching the drive motors to the robot chassis presented a challenge, as the Meccano pieces were not matched to the size of the motors. A solution was developed using a single screw per motor, which was able to hold each motor securely in place. Due to only having two wheels to work with, a marble was

added to the rear of the robot as a support (Fig. 4), allowing smooth movements. Mounting the Raspberry Pi to the chassis was another significant challenge, as it needed to remain stable when attaching and removing cables. This was solved by positioning two angled pieces of Meccano at the back of the Pi and attaching a locking tab at the rear to help secure the Pi in place. This setup provided stability and contributed to the overall rigidity of the robot.

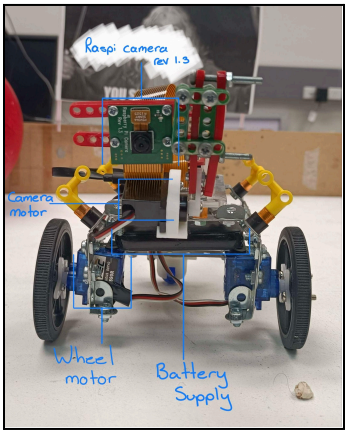

Fig. 3. Front View of Robot

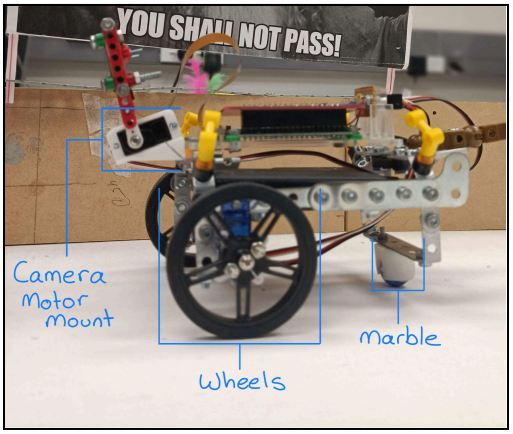

Fig. 4. Side View of Robot

The robot was controlled by a Raspberry Pi Zero W and used a Raspberry Pi Camera Rev 1.3 for visual input. A power bank stored in a compartment underneath the Raspberry Pi (Fig. 3) supplied power to the robot. This not only made the robot wireless but also minimized additional weight. Micro servo motors were used to drive the wheels and control camera movement.

4.2. Software

4.2.1. Pseudo Code

Pseudo code for two of the main functions used throughout the program, findLine() and findColor().

```
Function findLine()
        Take picture
        Set row to 239
        Set blackCount to 0
        Set blackPositionSum to 0

        For each column from 0 to 319
                Set brightness to the value of brightness component at (row, col)
                If brightness < 20
                        blackCount + 1
                        blackPositionSum = blackPositionSum + (col-160)
                End if
        End for loop
        Set position to 9999
        If blackCount > 5
                position = blackPositionSum/blackCount
        End if
        Return position
End function
```

```
Boolean Function findColor(index, compareIndex1, compareIndex2)
        Take picture
        Set colorCount to 0
        For each row from 0 to 239
                For each column from 100 to 219
                        Set color to the value of index component at (row, col)
                        Set compare1 to the value of compareIndex1 component at (row, col)
                        Set compare2 to the value of compareIndex2 component at (row, col)

                        If index = 0
                                If color > colcompare and color > compare1*1.5 and color > compare2*1.5
                                        colorCount + 1
                                        If colorCount >= 10
                                                Return true
                                        End if
                                End if
                        End if
                        If index = 1
                                If color > colcompare and color > compare1*1.5 and color > compare2*1.5
                                        colorCount + 1
                                        If colorCount >= 10
                                                Return true
                                        End if
                                End if
                        End if
                        If index = 2
                                If color > colcompare and color > compare1 and color > compare2
                                        colorCount + 1
                                        If colorCount >= 10
                                                Return true
                                        End if
                                End if
                        End if
                End for loop
        End for loop
        Return true
End function
```

4.2.2. Flowchart of Program

The flowchart in Figure 5 provides a simplified overview of the full program, highlighting the key decision-making processes and logic the robot used to navigate the track. The program was divided into four distinct quadrants, each representing a different phase of navigation. This structure is reflected in the flowchart to clearly illustrate how the robot transitioned between these sections, such as the initial line following, red marker detection, junction interpretation, and colour-based decision-making based on the camera input.

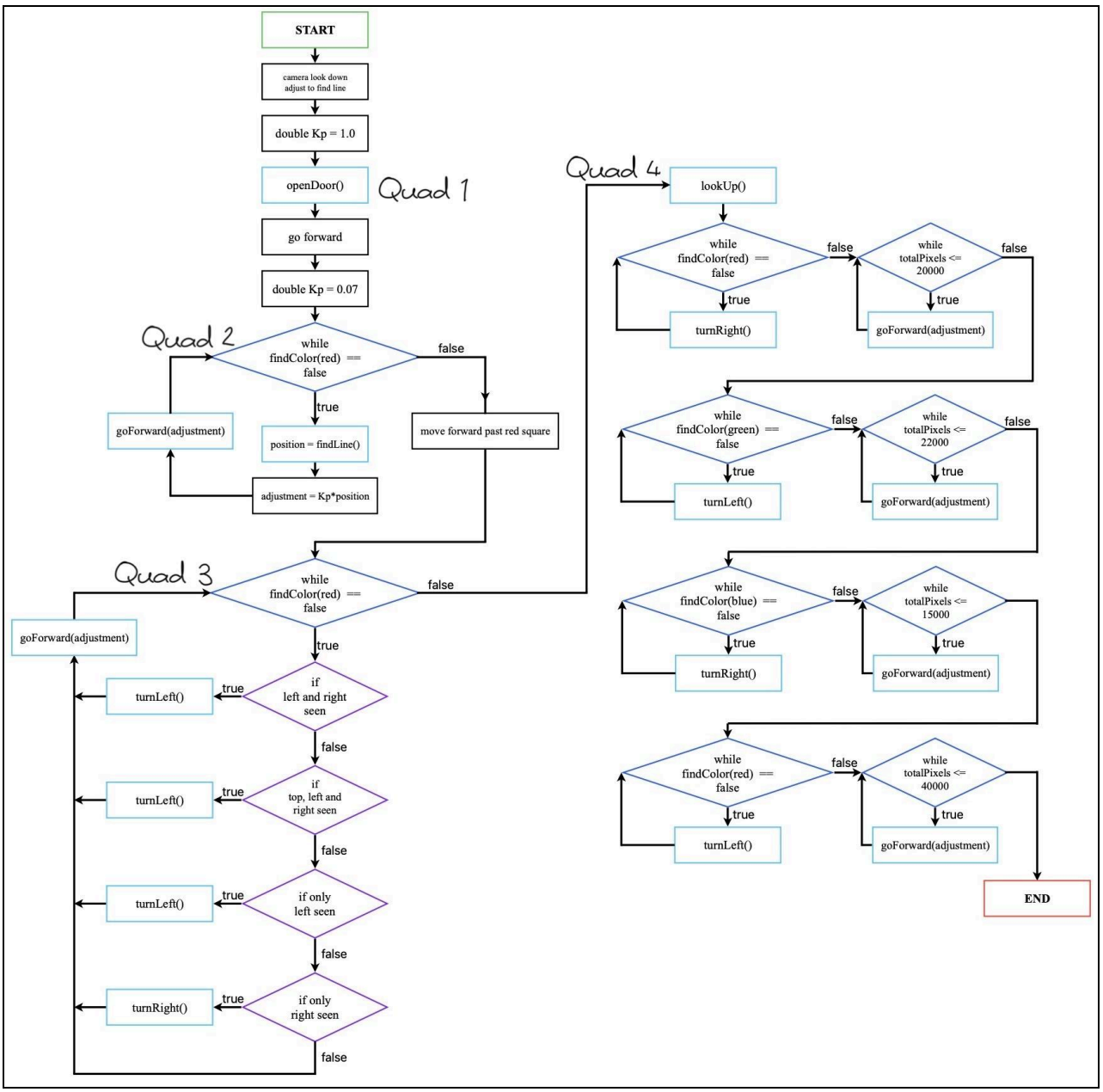

Fig. 5. Flowchart of Program

4.3. Team Arrangements

Each team member was assigned a role based on their strengths. One member acted as the coordinator, ensuring everyone was meeting project milestones and also contributed to the software development by assisting with smaller tasks. One team member was dedicated to the core software development responsible for writing the main control logic and image processing. Two team members focused on hardware including constructing the robot's chassis, troubleshooting motor issues, and 3D printing a custom part to improve camera movement. To help assist the programmers there was also a logic guru/architect who broke down complex decision making and assisted with code structure. Version control was managed using GitLab, with each team member working on individual branches to avoid any potential code conflicts.

5. Results

5.1. Testing

Table 1. Testing of Individual Sections

Date Tested	Section Tested	Result
May 6th 2025	Gate opening test 1	We were able to communicate with the door but unable to send the password back to the gate so the gate did not open
	Line detection test 1	First test for the lines work but the motors were set to the incorrect motor numbers
	Forward movement	Robot went backwards
	Gate opening test 2	Robot is able to connect to door - open it and move through
	Line detection test 2	Robot was able to see the line and follow it. At one of the turns the robot lost the line and wasn't able to find it again
May 7th 2025	Camera movement test 1	The motor on the robot was faulty - we tested and switched the motor out
	Switching between quadrants	Robot followed the line. Stopped when it saw red. Camera had to be manually adjusted downward to prevent the possibility of seeing two lines
May 8th 2025	Camera movement test 2	Adjustments have been made to make sure the camera is at the correct angle when needed due to the small range of movement we have
	Full run test 1	Due to a change in the code the robot now moves to fast and loses line and the door now shuts on the robot
	Line detection test 3	Robot is able to follow the line all the way to the red marker
May 15th 2025	Maze decisions test 1	Robot is able to start to navigate its way through the maze, but due to the position of the camera it isn't able to make decisions on directions quickly enough
May 22nd 2025	Maze decisions test 2	Robot is unable to detect the line ahead of it and therefore kept turning left
	Colour detection of pillars	Robot was able to find and go to the red and green cylinder and not hit them. Working on detecting blue pillar
	Switch wheels to front	Wheels swapped to front to make the camera closer to the wheel motors

	Maze decisions test 3	Robot is able to make correct decisions at each turn but makes incorrect turn at the cross-road
May 26th 2025	Maze decisions test 4	Robot is able to make correct decisions at each turn and make it to the end of the maze
	Full run test 2	Robot was able to complete the track from the beginning all the way to the end of the maze
May 27th 2025	Full run test 3	Robot was able to complete the entire track from the beginning all the way to the end

5.2. Line Detection Formulas

Before implementing the PID control algorithm, the robot took over two minutes to travel from beyond the door to the start of the maze section. After integrating the PID controller, the robot detected and followed the line more quickly and smoothly, significantly reducing the travel time.

5.2.1. Original Line Detection Formula

The robot used baseline edge positions to detect the boundaries of the line and decide on movement. The baseline edges are defined as:

- Top = 0
- Bottom = 239
- Left = 0
- Right = 319

Margin (tolerance) = 20

For each side, the distance between the baseline and the current detected edge is calculated as:

$$distance_{side} = |baseline_{side} - current_{side}|$$

Specifically, for the left and right edges:

$$leftDist = |baselineLeft - currentLeft|$$
$$rightDist = |baselineRight - currentRight|$$

The difference between these distances is then calculated:

$$diff = |leftDist - rightDist|$$

This difference was used to determine the robot's alignment and movement decisions.

5.2.2. New Line Detection Formula

For a smoother line detection and adjustment, changes were made to how the robot was detecting the line and how the movement adjustment was calculated. This was achieved through PID.

Definitions:

- colormargin = 20
- K_p = 0.07
- I(r, c) = brightness of pixel at row r, column c

Set *S* contains the columns with pixel brightness below the threshold at the given row:

$$S = \{c \in [0, 319]: I(linerow, c) < colormargin\}$$

Number of detected black pixels:

$$N = |S| = blackCount$$

Sum of pixel positions relative to the center column (160):

$$blackPositionSum = \sum_{c \in S} (c - 160)$$

If enough pixels are detected (N > 5), the average line position relative to the center is:

$$position = \frac{blackPositionSum}{N}$$

The position value is used to compute the robot's adjustment using proportional control:

$$adjustment = K_p \cdot position$$

5.3. K_p Values

Table 2 and Fig. 6 shows the K_P tuning values. Based on observations a score was assigned to the result for each K_P value on a scale of 1 to 5 as shown to the right of Table 2.

Table 2. K_P Value Tuning

K_P Value	Result	Score
0.01	Motor speed became unstable; the robot failed to respond effectively to line deviations.	1
0.05	Minimum value where the robot could consistently follow the line, but with delayed correction.	3
0.07	Provided the best balance of responsiveness and stability; ideal for navigating tight turns and decision points within the maze.	5
0.1	Used to align robot before starting	n/a

1	Unstable
2	Worked sometimes
3	Reliable but delays occur
4	Reliable but jerky
5	Smooth travel

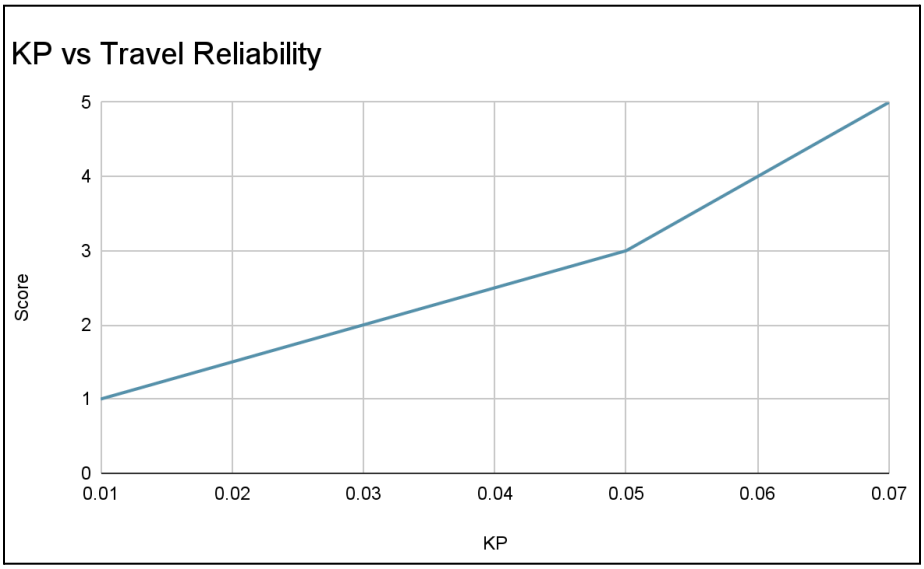

Fig. 6. Graph of K_P vs Reliability Score (Rating 1 to 5)

To optimize the robot's line-following performance, multiple values of the proportional coefficient constant K_P were tested. Each value affected the robot's responsiveness to making turns and stability differently. Only values above 0.05 enabled successful line detection and correction.

5.4. Reliability

To ensure consistent performance across all sections of the track, multiple test runs were carried out for each section to verify the reliability of the code and eliminate the possibility of one-off successes. Testing was performed at various times of day to account for changes in the lab's lighting, ensuring that lighting conditions did not affect the robot's ability to detect and follow the line.

5.5. Unexpected Findings

During testing of the colour detection, the robot had difficulty correctly identifying the colour blue. For example, when detecting red, the robot checked if the red component of a pixel was greater than 1.5 times both the green and blue components. This method worked well when detecting red and green, but proved unreliable for detecting blue. Several multiplier thresholds were tested to improve blue detection, with results that can be seen in Table 3. The most reliable approach was comparing blue without any multiplier.

Table 3. Multiplier Threshold for Blue Detection

Multiplier	Condition	Result
None	Blue > Red & Blue > Green	Worked best. Was able to consistently detect blue
1.2x	Blue > 1.2 x Red & Blue > 1.2 x Green	Only detected the blue once during testing
1.5x	Blue > 1.5 x Red & Blue > 1.5 x Green	Never detected blue
2.0x	Blue > 2.0 x Red & Blue > 2.0 x Green	Never detected blue

6. Discussion

The robot was able to successfully navigate its way through the full length of the track. It was able to do this by utilizing image processing as the primary input to make decisions regarding following the line, changes to the line direction and use of colour pillars for navigation. Along with image processing, PID control was used to refine directional changes, refer section 6.1. Evolution of the software is also discussed in section 6.2.

6.1. PID and K_P Tuning

To improve the speed and accuracy of the robot's line detection, a simplified version of proportional-integral-derivative (PID) control [4] was implemented. Rather than relying on the difference between the left and right edges of the line, the PID controller adjusted the robot's direction continuously based on its position relevant to the center of the line. This transition resulted in significantly smoother and faster navigation around the track.

Before PID was applied, the robot made sharp, abrupt corrections that often caused it to lose the line and waste time reorienting. The baseline edge-based method required constant large adjustments, which contributed to inefficiencies in navigation. In early testing, it took the robot up to 2.5 minutes to travel from the door to the start of the maze section. After implementing PID, the robot could make smoother, more responsive corrections, making the travel time significantly less than 2.5 minutes.

With PID in place, the proportional gain (K_P) needed careful tuning to find the best responsiveness from the robot. As shown in Table 2, four K_P values ranging from 0.01 to 0.1 were tested. A value of 0.01 caused the motors to behave erratically, while 0.05 had delayed corrections causing a slower movement. The value 0.07 provided the best balance of responsiveness and stability throughout the track, particularly within the maze. A higher value of 0.1 was briefly used at the beginning of the track to help the robot align correctly before entering the main course.

6.2. Design Evolution for Maze

A major challenge encountered during development was the initial approach to maze navigation. The original code relied on hard-coded movement commands to guide the robot through the maze. While this method provided a simple path, testing revealed that the robot often failed to detect when the next turn needed to be made. This lack of adaptability meant any changes to the maze layout would cause the robot to get stuck, as it could not respond to its environment.

To address this, the robot's movement and decision making logic was changed to a flexible, sensor-based control. Using the image processing input, the robot was able to detect potential turns it could make and make decisions accordingly. To simplify decision-making, the robot followed a left-turn priority rule: at any junction, it would first check if a left turn was available, then go straight if not, and finally turn right if no other option was detected. This approach improved the robots ability to navigate the maze independently and adapt to any potential changes to the layout.

6.3. Future Improvements

One area for future improvements is increasing the robot's overall speed, particular through the maze section. During testing, the robot was required to move slowly to compensate for the processing time of the image analysis. This speed ensured reliable detection of intersections and allowed for accurate decision making but reduced the overall performance of the robot.

To address this, further tuning of the K_P constant could help the robot respond more quickly and smoothly to line position changes without sacrificing stability. Continued testing could identify the best balance between responsiveness and control that would support higher speeds.

Another potential improvement involves adding additional sensors which can provide the robot with more information and awareness about its environment. For example, using infrared sensors which emit a beam of infrared light could help detect more line information by measuring the amount of reflected infrared light [5]. Increasing the number of sensors as seen in [6], could provide more reliable and faster decision making by offering the robot additional reference points to where the line is.

7. Conclusion

This project demonstrated the successful implementation of an autonomous robot capable of completing three key tasks: line detection and following, maze navigation and colour pillar detection. These tasks mirror real-world applications found in warehouses and factories where robots must be able to operate without human intervention. The robot relied on image processing using a camera to gather information about its surroundings. Despite being the only available input method, it proved highly effective allowing the robot to make accurate decisions. The initial approach using basic edge detection was limited in efficiency and was replaced with a simplified PID controller, which significantly enhanced the robot's speed and stability. Additionally, replacing hard-coded navigation with adaptable, sensor-driven logic improved the robot's ability to handle navigation of the maze section.

References

[1] S. Oswal and D. Saravanakumar, “Line following robots on factory floors: Significance and Simulation study using CoppeliaSim,” IOP Conference Series: Materials Science and Engineering, vol. 1012, p. 012008, Jan. 2021, doi: https://doi.org/10.1088/1757-899x/1012/1/012008

[2] D. Pour Yousefian Barfeh and E. Ramos, “Color Detection in Autonomous Robot-Camera,” Journal of Physics: Conference Series, vol. 1169, p. 012048, Feb. 2019, doi: https://doi.org/10.1088/1742-6596/1169/1/012048

[3] M. L. Dezaki, S. Hatami, A. Zolfagharian, and M. Bodaghi, “A pneumatic conveyor robot for color detection and sorting,” Cognitive Robotics, vol. 2, pp. 60–72, 2022, doi: https://doi.org/10.1016/j.cogr.2022.03.001

[4] P. Shah and S. Agashe, “Review of fractional PID controller,” Mechatronics, vol. 38, pp. 29–41, Sep. 2016, doi: https://doi.org/10.1016/j.mechatronics.2016.06.005

[5] N. C. Minaya, R. Rosero, M. Zambrano, and P. Catota, “Application of Multilayer Neural Networks for Controlling a Line-Following Robot in Robotic Competitions,” Journal of Automation Mobile Robotics & Intelligent Systems, pp. 35–42, Mar. 2024, doi: https://doi.org/10.14313/jamris/1-2024/4

[6] A. Roy and M. M. Noel, “Design of a high-speed line following robot that smoothly follows tight curves,” Computers & Electrical Engineering, vol. 56, pp. 732–747, Nov. 2016, doi: https://doi.org/10.1016/j.compeleceng.2015.06.014