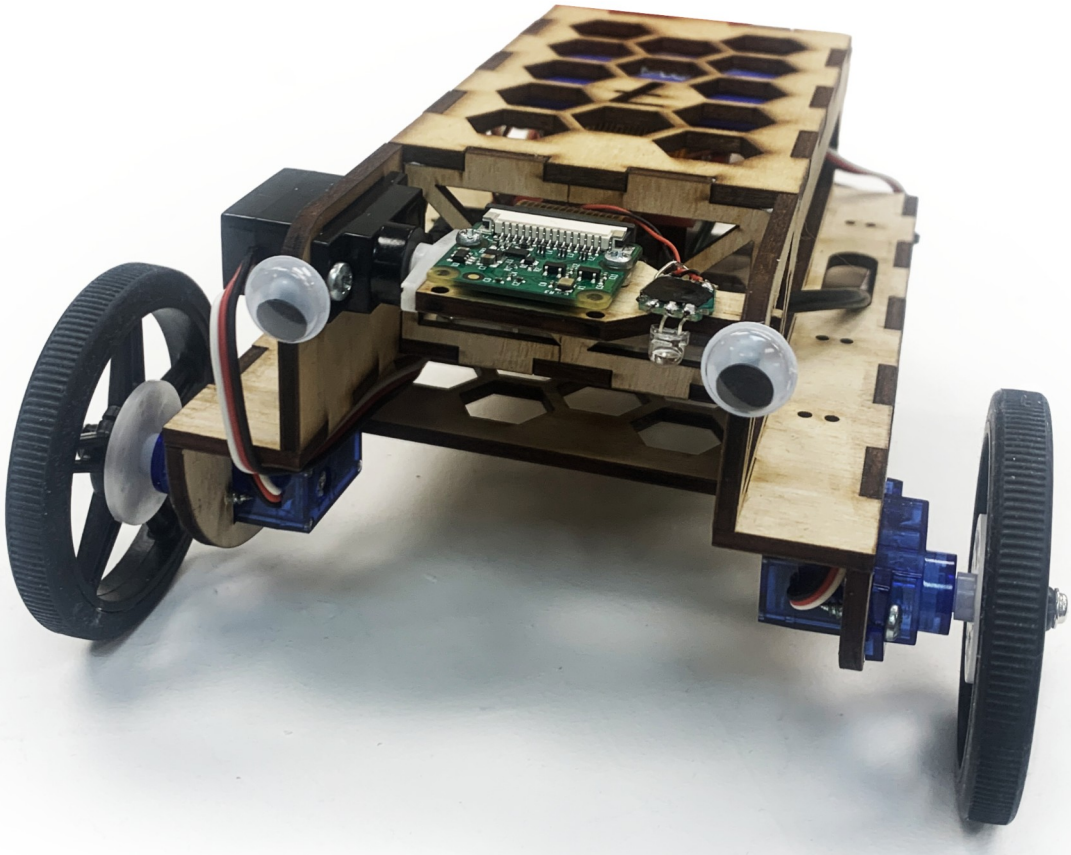
Autonomous Vehicle Challenge

Project Report



Galen Green

1 Abstract

This report summarizes the implementation and outcomes of the 2023 Trimeter 1, Team 7, ENGR101 Autonomous Vehicle Challenge (AVC) Project. The project involved four team members and focused on completing a mission comprising of four quadrants: opening a gate via a remote server, following a curvy black line on a white surface, navigating a maze, and locating coloured cylinders before pushing a red ball off a cliff. The report covers the mission objectives, hardware components, design considerations, and software structure and logic employed in the project.
It highlights the valuable insights gained into perception systems, motion planning, control algorithms, and physical hardware. Also the importance of effective teamwork and collaboration is emphasized, as the team leveraged multidisciplinary expertise to overcome challenges.
The project fostered a culture of innovation and problem-solving, enhancing technical skills and resilience. Overall, the project has laid a solid foundation for future endeavours in autonomous vehicle development, equipping the team with knowledge and skills to contribute to this transformative field.

2 Introduction

2.1 Report's Scope

This report focuses on the mission, process, implementation, outcome, and future development of the 2023 Trimeter 1, Team 7, ENGR 101 Autonomous Vehicle Challenge (AVC) Project.
All necessary Components and equipment, including continuous and non-continuous servos, a Raspberry Pi Zero, a custom pre-built PCB for motor control, and a Raspberry Pi Camera, where already supplied for the project and wont be covered in any detail in this report.

3 Background

3.1 The Team

Team 7 consisted of four students who where randomly assigned to work together:

Galen Green – Team Coordinator.
Elliot Bolter – Architect.
Joshua Perera – Tester.
Ramika Godakanda – Hardware Specialist.

3.2 Project Deadline

The project was assigned a timeframe of 5 weeks, during which the team had dedicated lab sessions and access to the facility beyond regular hours. Moreover, we were granted permission to take the robot home, However as team members also

3.3 Mission Objectives

The mission objectives can be divided into four distinct quadrants, each presenting unique software and hardware challenges. To successfully achieve maximum points and accomplish the mission, different approaches must be employed for each quadrant. Additionally, the vehicle must be designed and constructed using the given components as well as any personal materials.

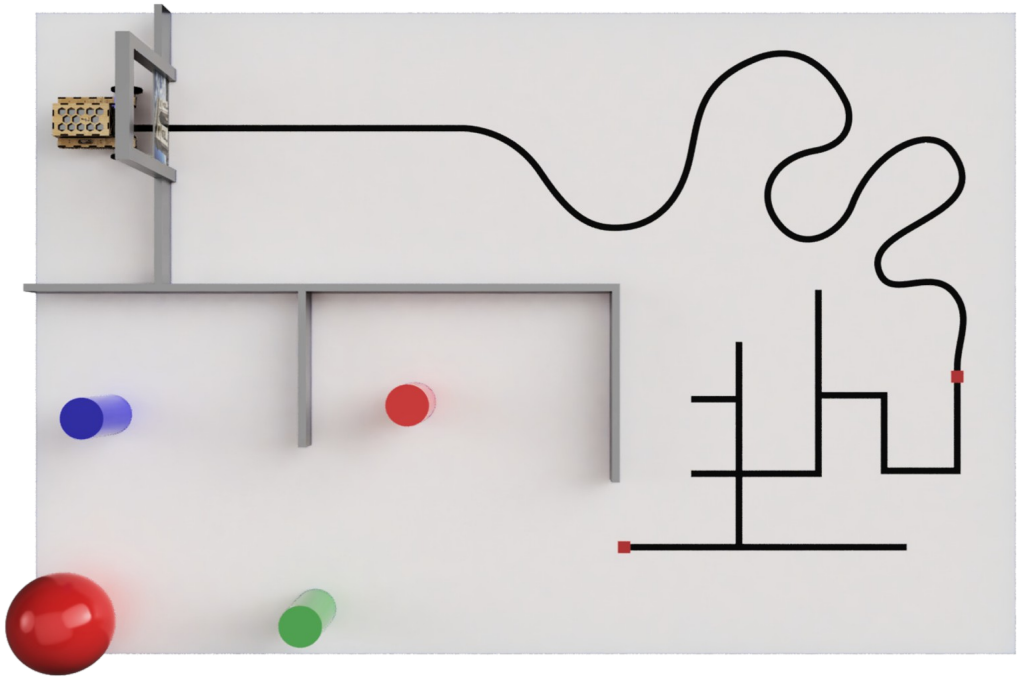


Figure 1: Course overview model

3.3.1 **Quadrant 1**

Open the gate by requesting to access and exchanging a password with the server over WiFi before preceding through the open gate before the gate closes after 4 seconds and avoiding decapitation by guillotine.

3.3.2 Quadrant 2

In this quadrant, the robot is required to independently track a continuously tightening curvy black path on a white surface, ensuring it doesn't deviate from it, and come to a halt upon detecting a red square.

3.3.3 Quadrant 3

Subsequently, the vehicle must solve the line maze by adhering to the path, identifying corners and intersections, and autonomously determining the appropriate direction to turn, if necessary. It must then navigate around them with precision before advancing to the subsequent intersection. Additionally, the robot is must also be able to halt upon detecting a red square once again.

3.3.4 **Quadrant 4**

Find and locate the coloured cylinders in the proper sequence (red, green, blue) and approach them without collision, and come to a halt nearby. Then, reverse and proceed towards the subsequent target. The final objective is then to push the red ball off the table than stop to avoid falling off after the ball.

4 Method

4.1 Hardware

4.1.1 Component List

- Raspberry Pi Zero
- Raspberry Pi Camera
- Custom Motor Control HAT
- 9G Micro Servo Motors
- 2 x Continuous 9G Micro Servo Motors
- 25 cm Right angle USB cable
- 2 x White LEDS
- 2 x Large wheels
- Power bank
- 5V Battery
- Multi direction ball caster wheel
- Custom laser cut plywood frame

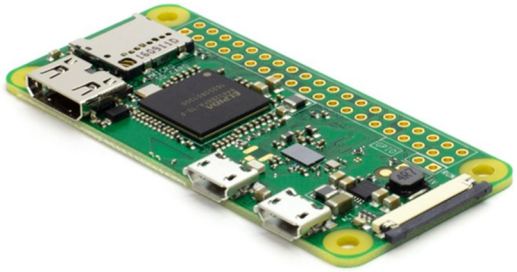


Figure 2: Raspberry Pi Zero

4.1.2 Raspberry Pi Zero

The Pi Zero is a self-contained computer equipped with 512MB of RAM and a 1GHz single-core CPU. It serves as the central processing unit, it is responsible for handling the input from the connected camera and generating commands to control the motors.

4.1.3 Servos

This vehicle utilizes two distinct servo variants. Firstly, there is a single 9G Micro Servo that can handle 1.6kg/cm of torque at 4.8V. It possesses 90 degrees of freedom, allowing it to rotate the camera module from a horizontal to vertical position. Secondly, there are two 9G analog servos capable of 360 degree continuous rotation. These servos can achieve a speed of 130 RPM each, enabling them to rotate the two active wheels responsible for the vehicle's movement.

4.1.4 Raspberry Pi Camera V2

The Raspberry Pi Camera v2 is equipped with an 8-megapixel sensor, which can capture images at a resolution of 3280×2464 pixels. It offers a horizontal field of view of 62.2 degrees and a vertical field of view of 48.8 degrees. In this particular project, the images are downscaled to 240×320 pixels to enable faster processing. The camera is connected to the Raspberry Pi using a ribbon cable.

4.1.5 Custom Motor Control HAT

The purpose of this custom PCB is to function as a 10-bit ADC, facilitating the conversion of digital input into analog signals. These analog signals are then used to communicate with and power the servos.

4.1.6 Custom laser cut plywood frame

The 3mm Birch plywood frame was precisely crafted with a Genesis G640L 80Watt CO2 laser cutter. This cutting-edge technology ensured the creation of a robust, lightweight, and visually appealing chassis platform suitable for securely mounting the hardware and ensuring dependable mobility.

4.2 Design

4.2.1 Camera Placement

The inspiration for the physical design stemmed from analysing several technical limitations. The primary consideration was the notion that the camera module should be positioned as close to the vehicle's rotational centre as feasible.
This placement facilitates smoother control of movement, particularly when making corrective turns. By ensuring that the line remains centred even after a correction turn, the angle of the line passing through the centre becomes indicative of the robot's required adjustment angle for course correction. Additionally, an important factor was the camera's field of view, which measured 62 degrees. To adequately capture an area of 83mm x 65mm, it was determined that the camera needed to be positioned at least 7 cm above the table's surface. This height, in conjunction with the camera's minimum focal distance, was deemed sufficient for reliable line tracking.

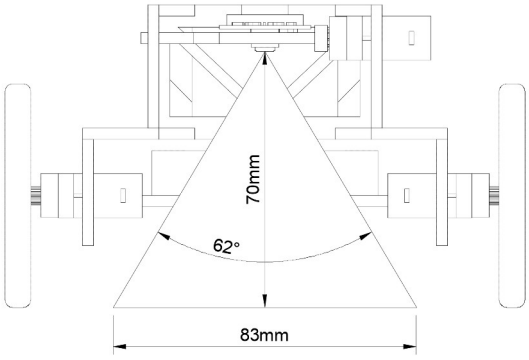


Figure 3: Camera field of view.

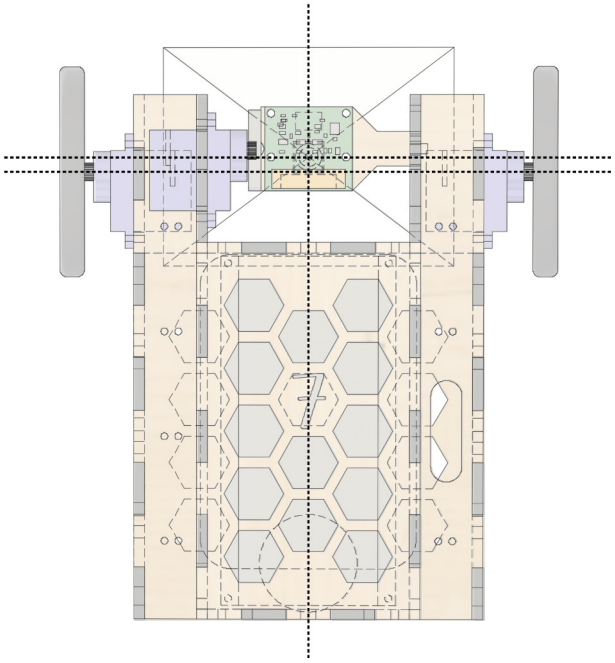


Figure 4: Top Camera Alignment

4.2.2 Wheel Placement

The powered wheels, with a diameter of 60mm, are directly attached to the continuous rotation servos using a threaded screw and adapter plate. For stability, a third ball caster wheel with a diameter of 28mm is positioned at the rear of the vehicle. Since this wheel is unpowered and experiences significant friction, the powered wheels are strategically placed wider apart to maximize leverage during turns and minimize the impact of drag.

4.2.3 Body Design

The primary chassis design incorporates the necessary criteria for camera and wheel placement while ensuring the Raspberry Pi and the battery bank are securely enclosed. To optimize weight distribution, the battery, being the heaviest component, is positioned as close to the centre of rotation as possible without obstructing the camera. The orientation of the Raspberry Pi was determined by the requirement for the CSI camera connector to face forward and maintaining a horizontal and upright alignment to avoid bending the ribbon cable.
The design decisions were also influenced by the selection of materials and manufacturing methods. Initially, 3D printing was considered for creating the chassis, but it was soon decided to rather use 3 mm laser cut plywood. This decision was based on factors such as cost, weight, flexibility, precision, and the speed of design iterations.
Since laser cut parts are primarily 2D, the chassis design incorporates finger joints to assemble the necessary 3D framework. In addition most of the panels in the design feature a support structure with either hexagonal or triangular patterns to decreases weight, facilitate cable routing, provide convenient access to internal electronics, and also provide aesthetic appeal.

5 Software

5.1 Overview

The C++ code's structure is divided into five sections: four quadrants representing different aspects of the mission, and a main section serving as a timeline. Each quadrant is responsible for specific tasks related to its mission aspect.

In addition, the main section controls the overall flow of the program, determining when and in what order the quadrants are executed based on the mission's requirements.

By dividing the code into sections, it becomes more modular and organized, improving readability and maintainability. This structure allows for easier collaboration between developers and makes troubleshooting and debugging more efficient.

Additionally, the code includes various sub-methods that can be called by any quadrant as needed. These methods provide additional functionality and optimize code execution for improved efficiency.

5.2 Quadrant 1 – Gate

This initial and brief section is responsible for initiating the robot, facilitating the opening of the gate, and ensuring that the robot passes through the gate within the allotted time before it closes again.

To achieve this, several actions are performed. First, all motors are set to their starting positions and speeds. This includes adjusting the camera servo to point downwards and setting the movement motors to a neutral position with zero speed. Additionally, the robot executes a straightening manoeuvrer (refer to section 5.8) to align itself parallel to the line.

The gate opening process involves sending a request to the server located at the IP address 130.195.3.91 using port 1024. In response, the server returns the correct password, which is then utilized in another request to the server to open the gate.

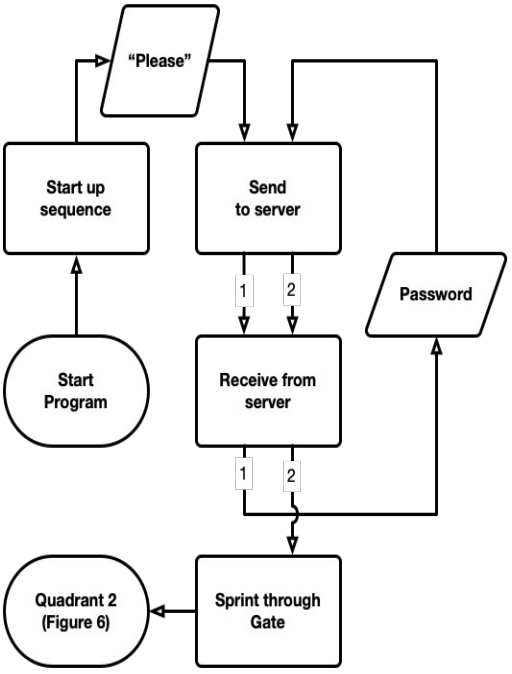


Figure 5: Quadrant 1

Once the final open command is dispatched, the moveDistance (refer to section 5.6) method is invoked to propel the robot forward at full speed for a distance of 30cm, allowing it to pass through the gate before it automatically closes after a few seconds. Subsequently, quadrant 1 is considered complete, and the main program proceeds to call quadrant 2.

5.3 Quadrant 2 – Curvy Line

Quadrant 2 is response for following the curvy line from the gate till the red square.

The robot utilizes its camera to locate the black line within its field of view, enabling it to determine the necessary correction magnitude to align the vehicle in the direction of the line.
The analysis of the video stream is performed using the findPosition method (section 5.7).

The code operates in an infinite loop. Initially, it checks if the robot has reached the red square by calling the isRed method. If the red square is detected, the loop is broken, and the robot proceeds to quadrant 3. However, if no red square is detected, it then verifies the presence of a line to follow. If no line is found, the program exits since it indicates that the user has moved the robot away from any line.

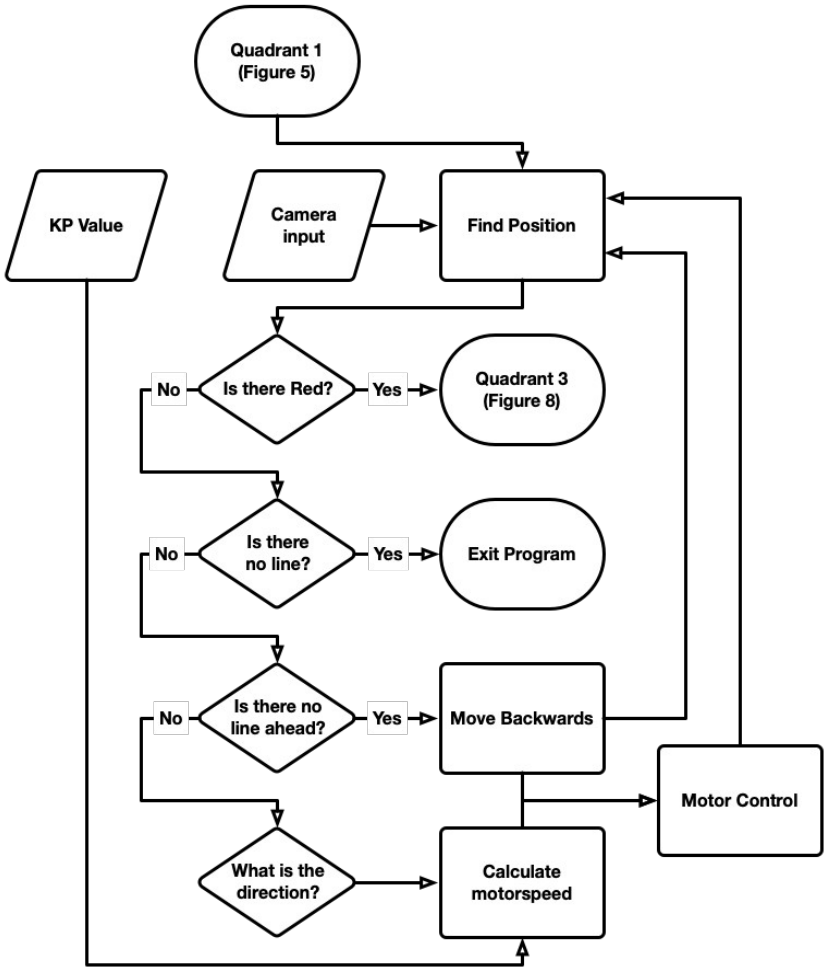


Figure 6: Quadrant 2

However, if the robot detects a line but not directly in front of it, it will take a step backwards until a line is present in front again. In such cases, the direction value obtained from the findPosition method (refer to section 5.7) is used. By multiplying this value by the scalar kp (set to 0.3), the magnitude of the turn can be calculated. This calculated value is then added to the base forward moving speed before being sent to the motors for adjustment.
This approach works because if the direction value is close to 0, indicating that the line is straight ahead, the robot moves at its default forward speed (set to 30%). However, if the direction value is significantly positive or negative, the resulting adjustment value is large. This adjustment slows down one wheel while speeding up the opposite wheel, causing the robot to turn in the direction indicated by the sign of the direction value.

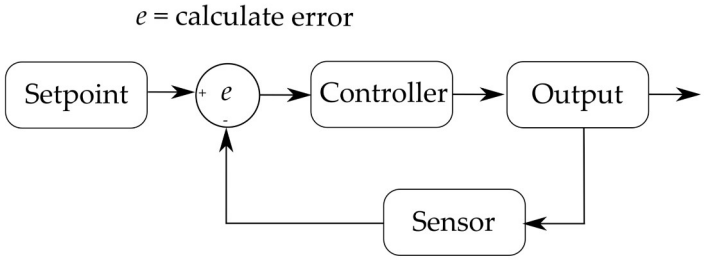


Figure 7: General Control System

5.4 Quadrant 3 – Maze

Quadrant 3 presents a similar challenge to quadrant 2, but with the additional difficulty of sharp corners and intersections that the previous system cannot reliably navigate. Prior to initiating the movement, the robot ensures its alignment with the line by executing a straightening manoeuvrer (see section 5.8). It then proceeds past the red square detected in quadrant 2 before commencing a loop similar to the previous one seen in quadrant 2. However, when the system detects any type of intersection, instead of following the line, it invokes a separate turn method (refer to section 5.9) that executes a series of manoeuvrers to consistently make the desired turn. Afterward, it resumes following the line.

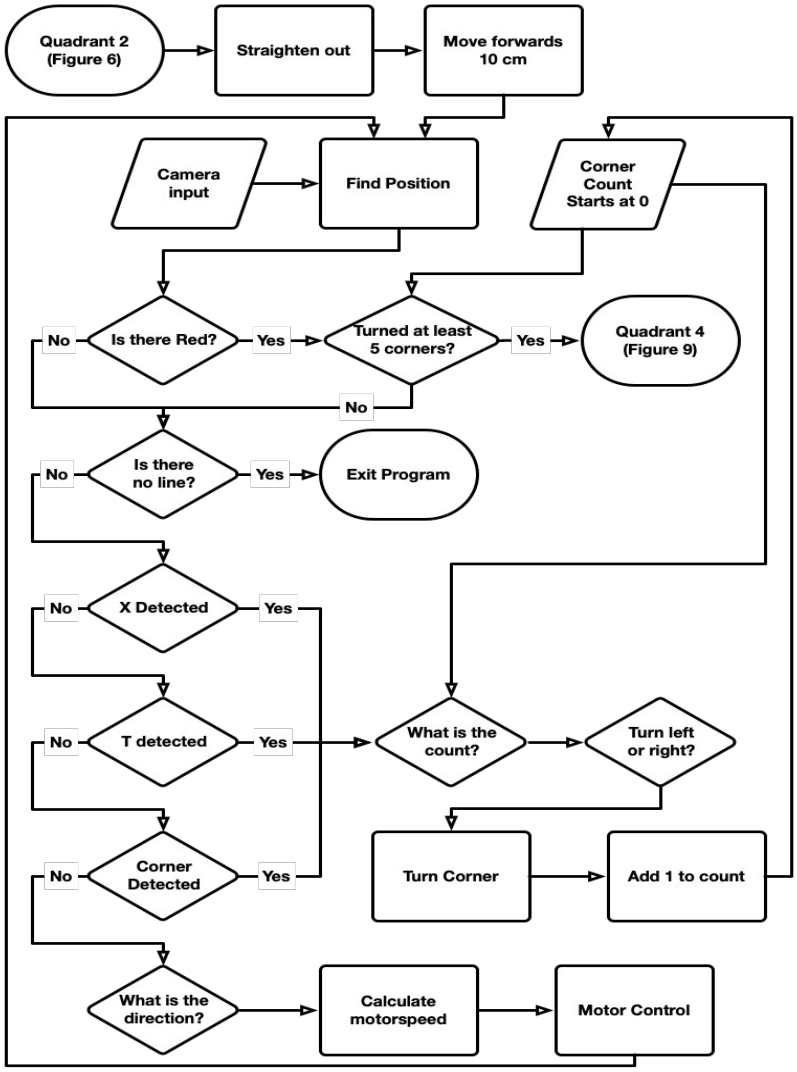


Figure 8: Quadrant 3

The system detects intersections through various means by employing different variations of the findPosition method (see section 5.7). This method enables the system to determine the presence and direction of lines on all four sides of the camera's field of view. Consequently, if lines are detected on all four sides, or on three sides excluding the front, or on two sides excluding the front, it signifies the presence of a four-way intersection, a T-junction, or a turn, respectively. However, if a line is only detected at the front and back, it indicates the absence of an intersection, allowing the robot to proceed straight using the same method employed in quadrant 2, additionally, if no line is detected on three sides, with only a line detected behind, the robot will take a step backwards to find the line again.

To determine the correct direction for turning, the system keeps track of the number of corners it has turned and matches that count with a set of preprogrammed turns. This approach offers several advantages, including a reduction in the number of required turns, thereby minimizing potential failure points. Additionally, the red detection system is activated only after the completion of the last turn. This precaution prevents accidental triggers when passing over the first red square.

5.5 Quadrant 4 – Coloured Cylinders

To navigate toward the coloured cylinders, the robot employs several steps. Initially, the camera module is adjusted to face forward, ensuring a clear view of the surroundings. Secondly, since the first red cylinder is positioned to the right, the robot is rotated approximately 45 degrees (refer to section 5.6) in that direction to align itself roughly and insure a clear line of sight.

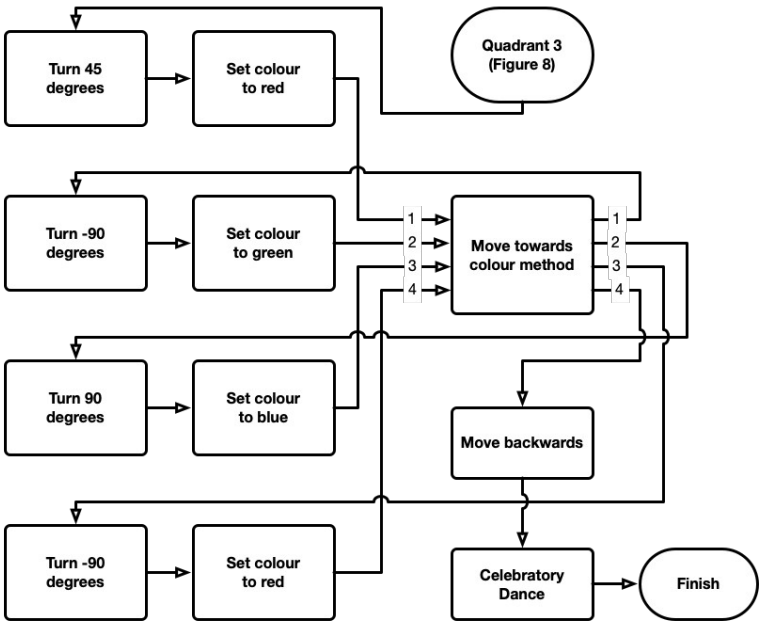


Figure 9: Quadrant 4

Then once the target is identified and its direction is determined (refer to section 5.10), the robot utilizes a similar approach employed in previous quadrants. Using proportional control in a loop to navigate towards the cylinder while constantly correcting its course. Concurrently, the robot keeps track of the number of coloured pixels visible in its field of view. This information proves valuable, as the count increases as the robot approaches the target, with the cylinder occupying more space in its vision.

When the count exceeds 5500 pixels, the robot halts its movement, rotates towards the next target, and repeats this process three times. The sequence is followed for the green, blue, and red cylinders. However, during the final iteration, the robot ceases its movement only when the count of red pixels falls below 500. This indicates that the red cylinder has fallen off the table. Upon detecting the absence of the red cylinder, the robot reverses its motion and executes a celebratory dance before concluding the program.

5.6 moveDistance Method & turnAngle Method

These two methods employ a similar process to control the robot's movement based on a desired outcome. Both methods take the desired outcome value as input and execute the necessary commands to move the robot accordingly. The process involves configuring the motors to turn in specific directions for turns or in the same direction for movement. Then, a predefined amount of time is waited until the motors have rotated the desired amount, after which they are stopped.

In the case of a negative inputed value, the robot either turns left or moves backwards. Conversely, a positive inputed value causes the robot to turn right or move forward.

To determine the appropriate wait time, the following formula is used:
Time = (|Input| * Scalar) / Speed. It is worth noting that different scalar values have been determined through repeated trial and error for each method (600 for turnAngle, and 3400 for moveDistance). Additionally, the absolute value of the input is utilized because the required time remains the same regardless of the movement direction.

5.7 findPosition Method

One of most critical tools used in this program, this method takes takes a desired row or column of pixels as input and determines the relative position to the centre of a black line, if present.

Initially, the brightest and darkest pixels in the given data set are identified, these two values are then used in the following formula to establish a threshold for black pixels:

Threshold = DarkestValue + (BrightestValue – DarkestValue) * 0.3

This equation calculates a brightness value that lies 30% between the darkest and brightest values. It serves as a reference point to determine whether each pixel in the desired row or column is black or white. Pixels above the threshold are considered white, while those below are considered black. This approach adapts dynamically to changes in lighting conditions and disregards shadows.

Next, by utilizing two nested for-loops (one for rows and one for columns), the program adds either a 0 or a 1 to a vector list for each white or black pixel, respectively.

Afterwards, using another for loop, the program calculates the sum of all the positions for each black pixel in the list. This is achieved through the following formula:

Sum += (0 or 1) * (Vector index – half the vector)

Multiplying by 1 contributes a value to the sum, while multiplying by 0 does not. Consequently, the average position of all black pixels is determined using the formula:

position = sum / black pixel count.

Finally, if the number of black pixels falls within the range of 25 to 100, indicating the presence of a black line, the method returns the position value. However, if there is no line or if there is an excessive amount of line, it returns an error code (e.g. 999).

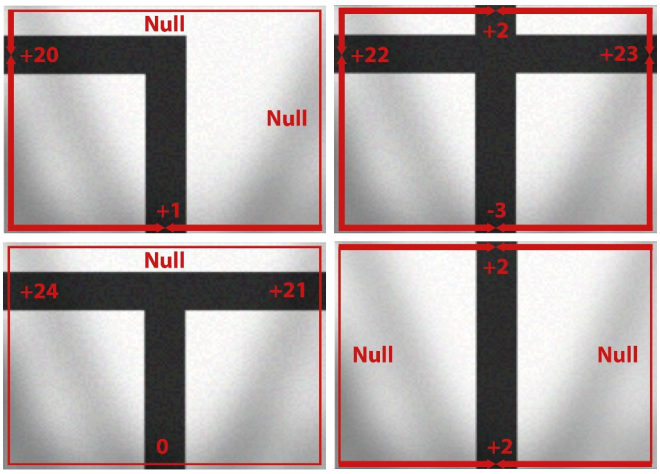


Figure 10: Intersection view

5.8 Straighten Method

This method tries to rotate the robot so that it is parallel with the line, this is done by comparing the bottom and top position value, and turning until they match, or are within a certain range of each other. Using the turn angle method (refer to section 5.6).

5.9 **TurnCorner Method**

This set of manoeuvrers is specifically designed to navigate sharp corners consistently. It starts by moving backwards 5 cm (see section 5.6), followed by straightening out (see section 5.8). Next, it turns 30 degrees in the desired direction and advances 5 cm forward. Subsequently, the robot proceeds to move forward slowly until the line is positioned at the centre of its field of view, utilizing the findPosition method (refer to section 5.7), which scans the centre column. Afterward, the robot readjusts its orientation until it becomes parallel to the line once again, effectively cutting the corner.

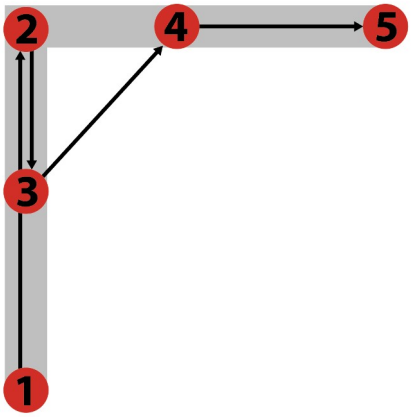


Figure 11: Corner turn sequence

5.10 **Coloured Cylinder Method**

When the coloured cylinder method is called, it scans rows 45 to 75 of pixels using two nested for loops, similar to the findPosition method described in section 5.7. However, instead of solely considering intensity, it compares the red, green, and blue values according to the following ratios:

1. Red cylinder: Red must be 1.8 x greater then blue and 1.8 x greater then green.
2. Green cylinder: Green must be 1.57 x greater then blue and 1.25 x greater then red.
3. Blue cylinder: Blue must be 1.3 x greater then red and 1.2 x greater then green.
4. Red Ball: Red must be 2 x greater then blue and 2 x greater then green.

6 Discussion

6.1 Results

Team 7 successfully completed the mission in a remarkably short time, thanks to the exceptional teamwork exhibited from the beginning. This collaborative effort allowed us to make significant progress towards achieving the mission milestones and effectively tackle the challenges we encountered, ensuring that we either met or exceeded our scheduled targets.

Our innovative and unique choice of hardware, design, material and manufacturing techniques, coupled with our swift construction of a robust and dependable platform for software testing, proved to be invaluable. As a result, Team 7 quickly surpassed other teams in terms of progress.

Regarding software development, our strategy of dividing the project into four distinct quadrants, each responsible for specific mission tasks, with a central module overseeing program control in a modular structure, proved highly effective. This approach facilitated efficient development and testing of individual components, while ensuring seamless integration of the entire system through regression testing that encompassed real-world scenarios and potential edge cases. Consequently, we created a fault-tolerant system capable of withstanding various challenges.

Although the project encountered obstacles and setbacks along the way, our team persisted, dedicating numerous late nights and countless hours to gain valuable insights.
In the end, we achieved commendable results. The autonomous vehicle exhibited reliable performance, successfully completing all assigned tasks with excellence, thereby highlighting

the effectiveness of our design choices and implementation strategies.

6.2 Future Development

Moving ahead, we propose implementing several enhancements to the autonomous vehicle system to further improve its performance. Firstly, we suggest refining the algorithms responsible for decision-making, aiming for better accuracy and faster response times. Additionally, integrating additional sensors like angular velocity sensors into the system would enhance its perception capabilities and provide feedback of movement.

Another area where improvements can be made is in the computer vision system. By utilizing OpenCV or similar technologies, we can achieve more precise classification of targets, such as the black line or other coloured targets.

In terms of hardware, upgrading to higher quality servo motors or stepper motors would significantly enhance the system's reliability and repeatability. Furthermore, introducing a third wheel with a lower drag coefficient would provide substantial assistance.

By continuously iterating and innovating, we can push the boundaries of autonomous vehicle technology. This progress will contribute to the advancement of this growing field, ultimately leading to safer roads. The proliferation of self-driving vehicles also has the potential to reduce travel times, road traffic but also to minimize carbon emissions, thus benefiting the environment.

7 Conclusion

The AVC Project has indeed been a remarkable learning experience, providing valuable hands-on experience in autonomous vehicle development. The project has allowed us to bridge the gap between theoretical concepts and practical implementation, enabling us to apply our knowledge in a real-world setting. Throughout the project, we have gained valuable insights into various aspects of autonomous vehicle development, such as perception systems, motion planning and control algorithms, equipping us with the necessary skills to tackle future challenges in this field.

The project has also highlighted the importance of effective teamwork and collaboration. Developing an autonomous vehicle requires a multidisciplinary approach, involving expertise from various domains such as software engineering, robotics, and electrical engineering. The project has also allowed us to work together as a team, leveraging our individual strengths and expertise to overcome obstacles and achieve our goals.

Moreover, the project has fostered a culture of innovation and problem-solving. Throughout the project, we have encountered numerous challenges and obstacles that have tested our perseverance as developing an autonomous vehicle is a complex task that requires creative thinking and the ability to adapt to changing circumstances.

In summary, the AVC Project has been an exceptional learning experience, showcasing our dedication, teamwork, and technical expertise. We are proud of the accomplishments we have achieved, and we are excited about the possibilities that lie ahead in the field of autonomous vehicle development. The project has equipped us with the necessary skills and knowledge to contribute to the advancement of this technology, and we eagerly look forward to future opportunities in this dynamic and transformative field.

References

[1] E. Bolter, J. Perera, R. Godakanda, and G. Green, “AVC.cpp,” GitLab. https://gitlab.ecs.vuw.ac.nz/course-work/engr101/2023/project3/t7/avc-project-plan (accessed Jun. 13, 2023).

[2] A. Roberts, “AVC Reference Material,” ecs.wgtn.ac.nz, Apr. 28, 2023. https://ecs.wgtn.ac.nz/Courses/ENGR101_2023T1/AVC_manuals (accessed Jun. 13, 2023).

[3] A. Roberts, “AVC project video processing and movement control,” ecs.wgtn.ac.nz, May 01, 2023. https://ecs.wgtn.ac.nz/foswiki/pub/Courses/ENGR101_2023T1/LectureSchedule/ENGR101_Lecture15.pdf (accessed Jun. 13, 2023).