13/06/2023

ENGR101 - PROJECT 3

AVC FINAL REPORT

Lecturers: Howard Lukefahr and Arthur Roberts

Victoria University Wellington

Abstract

The Autonomous Vehicle Challenge (AVC) at Victoria University brings engineering students together to construct and program autonomous vehicles for navigating a path independently. Despite time constraints and limited hardware and software experience, Team 5 successfully built a functional autonomous vehicle. However, further development is possible as the result only covers 80% of the course. While the hardware design was flawless, coding presented challenges. Our team's experiences with time management and teamwork offer valuable insights for future projects, highlighting areas for improvement and providing lessons on successful project execution.

Introduction

Scope

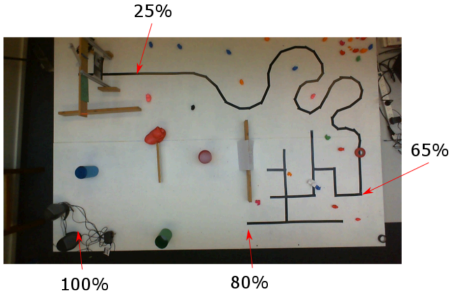
This report covers the process of working towards completing Victoria University's Engineering 101 Autonomous Vehicle Challenge (AVC). Each team of four to five random students is given four weeks to build and program a robot to finish a course. All materials and equipment, including motors, power banks, the Raspberry Pi, robot building elements, wheels, 3D printers, and the Raspberry Pi Camera, are given.

Motivation

This project's goal is to design, construct, and build a robot with a team of assigned members. It must negotiate a course with multiple portions that become more difficult as the robot progresses, testing the team's teamwork skills in order to problem solve and use each member's knowledge in real-world settings. The project seeks to discover the simplest, most efficient, and best solution to the problem.

Aim

The robot must be able to navigate through the course. The course consists of four different connected environments with different settings and challenges to test the robot. The first quadrant requires the robot to open a gate by exchanging data with the server over WiFi. The second quadrant tests the robot's ability to follow the curved black line. The third quadrant



requires the robot to make sharp turns and turn in the right direction in order to reach the end of the black line. The fourth (and final) quadrant requires precision in order to locate and maneuver towards three colored cylinders one after the other without hitting the cylinders, and finally drive towards the red ball to push it off the table without the robot itself falling.

Objective

Build a vehicle that is completely autonomous using the provided equipment and knowledge about the course. Construct a C++ program that uses the given E101 library to correctly navigate the vehicle from the beginning to the end of the course. Present a completed project that can complete the course by June 1st.

Background

Hardware

Raspberry Pi

The Raspberry Pi is a single board computer used as the main processor and controller for the robot. It is programmed using C++ in order to process the input from the attached camera. It provides functions such as controlling the motors, sensors, and camera.

Printed Circuit Board (PCB)

The PCB has a 10-bit ADC, which converts information in analog and digital signals into 1s and 0s so that it is readable by the computer. It also calculates the error for line detection. An 8 channel digital input and output are used to communicate with the servos.

Servo

There are two different types of servos used. The continuous servo is used to control the wheels as they allow continuous 360 degree rotation. The micro servo is used to control the angle of the camera from a position that we set ourselves.

Software

C++ terms

- Unsigned int: positive integer
- Int: positive and negative integer
- Array: a list of set size
- Method: a grouping of code that can be used when the method is called
- Function: a block of code that performs some operation when it is called

Proportional Integral Derivative (PID)

PID is a mathematical equation used to smoothly adjust and control the direction of the robot based on the middle black line that is being followed. The change is calculated by scanning a horizontal row of pixels and getting the pixels that are black. If the pixel is black, it will return 1, and if the pixel is white, it

will return 0. It is then applied to the PID algorithm to make an adjustment for the motors. The error is then calculated by multiplying the 0s and 1s with the array of indexes (getting the position), and adding all the numbers together to get the total error. The average error is then calculated as $error = error/bpix$ where bpix is the number of black pixels. This is then plugged into the PID algorithm to get the required adjustment, which could be calculated by using the formula: $adjustment = Kp \times error + Kd \times de/dt$. In the applied algorithm that is used in the team's code, the adjustment value is only obtained by multiplying the error with the proportional coefficient (Kp), resulting in a simplified version of the PID algorithm ($adjustment = Kp \times error$). The adjusted motor speeds are calculated based on the error and the adjustment value.

Close Loop

Loop that receives continuous input (algorithm used for the PID).

RGB Value

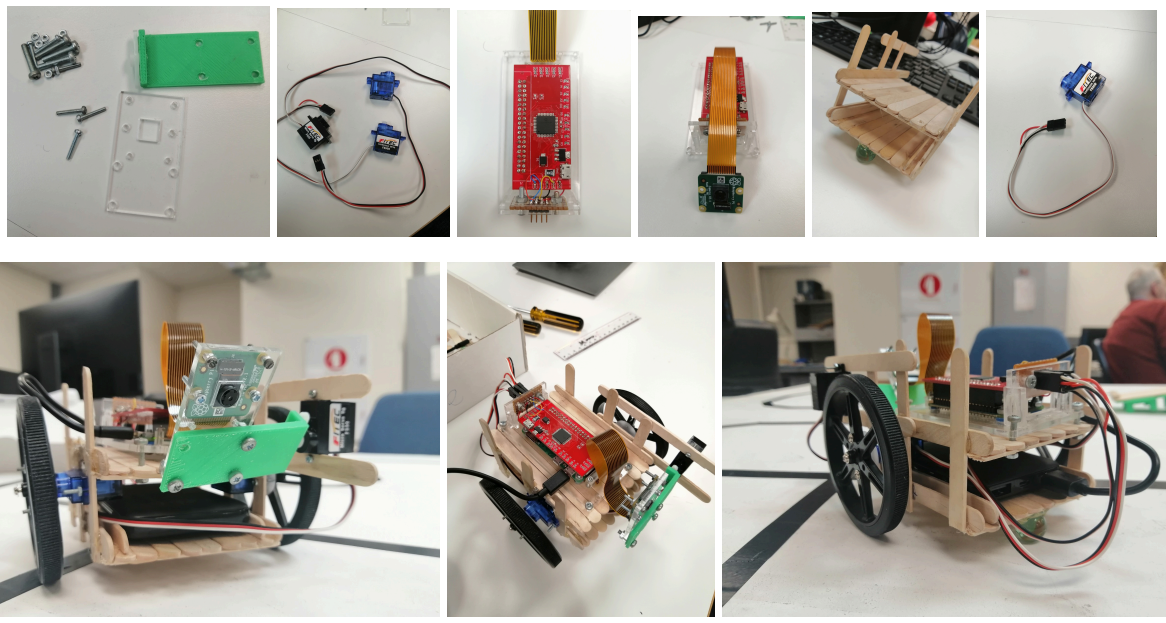
The RGB values (Red, Green, and Blue) are represented by distinct channel values R = 0, G = 1, B = 2. Throughout the course, the robot must follow a black line where the RGB values are used to compare and calculate for the robot to do specific movements or actions. Furthermore, the fourth channel, alpha channel, is used to discern the existence of black and white pixels by comparing the strength of each, with black pixels having a considerably lower intensity than white pixels.

Method

Hardware

Components used

Raspberry Pi 0W, Raspberry Pi camera, 2 large wheels, power bank, popsicle sticks, 2 continuous servo motors for wheels, 1 micro servo motor, 1 marble, PCB, 3D printed components (brackets), screws



(prototype robot)

Building the vehicle

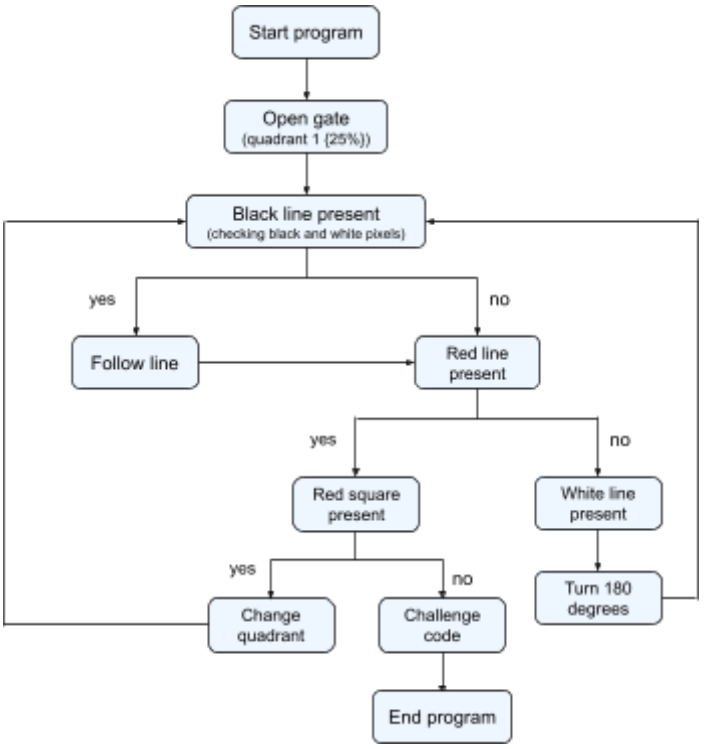
The robot's body is made of wooden popsicle sticks and hot glue instead of 3D printed parts. It consists of two tiers: the lower level holds the power bank and connects the wheels to the servos, while the top level houses the Raspberry Pi and PCB. The Raspberry Pi, micro servo motor, and camera are attached using 3D printed brackets and screws. A marble at the bottom helps balance the vehicle. The robot needs to be short enough to fit through a gate and complete the first quarter of a course. The wheels are positioned close to the camera for accurate readings and decision-making. Wires are hidden neatly, and all sections are easily accessible for repairs or component replacements.

Software

Uploaded to a group GitLab account.

Overview

The code is split into five different files, each responsible for separate parts of the course. Every file includes the needed libraries and header files in order to connect all the quadrants into 1 main file.



(program overview)

Description of each file and what it does:

➢ **Main file**

Algorithm:

1. int error = init(0) ← Initialize the system and check for any errors during initialization
2. Open_screen_stream ← open the camera functionality
3. Boolean a = true ← indicates that the system is running
4. Quad = 0 ← set the starting quadrant to 0
5. While a is true ← while system is running
6. Use a switch case based on the value of quad (what quadrant the robot is in) to execute different algorithms
7. Case 0 ← calls the function for opening the gate

8. Case 1 ← execute the code for quadrant 2
9. Case 2 ← execute the code for quadrant 3
10. Case 3 ← execute the code for quadrant 4
11. After each case, update the value of 'quad' to proceed the next quadrant
12. stoph() ← stops the system after the while loop
13. Return 0 ← exit the program if program is executed successfully

Controls and calls each quadrant within a while loop. To keep things simple, instead of having one giant file that does all the quadrants, we break it into multiple subfiles and call them in the main file. The programme executes in a main loop that continues until it is broken. The main file begins by calling the file to open the gate in quadrants one, two, three, and four. The main file contains a switch case that allows us to switch between algorithms.

➢ **Gate (Quadrant 1)**

Algorithm:

1. connect_to_server(server address, port) ← establish a connection with the gate server
2. Send_to_server("please") ← send 'please' to server
3. receive_from_server(password) ← receive the password from the server
4. send_to_server(password) ← send password back to server

The gate file involves opening a gate by communicating with it over the network. In order to communicate with the network and send a message to open the gate, we use 3 functions in E101 library which is about moving around arrays of chars and waiting for the server. To open the gate, we use the connect_to_server function to connect to the server's IP address, 130.195.3.91, and then send the message "please" using the send_to_server network function. They will receive the password-containing message from the server (receive_from_server function) and then send the password back to the server. The vehicle will remain stopped until the gate opens. When the gate opens, the vehicle will go to the next quadrant.

➢ **Quadrant 2**

Algorithm:

1. Initialize the servo position ← put the servo camera down
2. Set initial parameters and variables (arrays to store pixels, threshold value, kp value)
3. While q2 is true ← enter a loop for quadrant 2
4. open_screen_stream(), take_picture(), update_screen() ← Capture the camera image and process it
5. For loop to calculate the average index of black pixels in each column
6. If pixel is less than threshold, pixel is black
7. If pixel is bigger than threshold, pixel is white
8. Calculate the error as the difference between the average index and the middle point (middle point is 120 since half of 240)
9. Adjust the motor speeds based on the error using a proportional constant
10. Adjustment = kp * error ← set adjustment value by multiplying kp and error values (PID)
11. set_motors(left_motor speed + adjustment) ← Set motor speeds and add it with the adjustment val if black pixels are detected
12. set_motors(left_motor speed) ← Set default motor speeds if no black pixels are detected
13. set_motors(), hardware_exchange() ← Update the motor speeds and perform hardware exchange
14. quad_change() ← change the code to the next quadrant if condition is met

In Quadrant 2, the robot follows a black line using image processing and proportional motor control. It starts by initializing variables for threshold, proportional coefficient, and middle point. Within a loop, the robot continuously takes pictures, adjusts its position based on the calculated error, and modifies motor speeds accordingly. The error is determined by analyzing black and white pixels in the middle row of the image. Black pixels are identified by being below the threshold and assigned a value, while white pixels are assigned a value of zero. The motors are adjusted using the proportional

coefficient (with the formula $adjustment = kp \times error$ and the calculated error. If a red pixel is detected, indicating the presence of a red square, the robot switches to the next quadrant.

➢ **Quadrant 3**

Algorithm:

1. While q3 is true ← enter a loop for quadrant 3
2. Calculate error values for different areas of the images, such as the left column, horizontal line, and right column using same formula for calculating errors in quadrant 2
3. error_horiz = error_horiz/bpix_horiz; ← calculate average error position for the horizontal line
4. Check for intersections based on the number of black pixels and the range of errors in the horizontal line.
5. If intersection is detected:
6. Determine the turning direction based on a count variable.
7. Use a "center" function to align the robot to the center of the black line.
8. Update the count variable.
9. If no intersection is detected:
10. Adjustment = kp * error ← calculate adjustment value by multiplying kp and error in the horizontal line (PID)
11. Adjust the motor speeds.
12. quad_change() ← change the code to the next quadrant if condition is met

Similar to Quadrant 2, the robot uses the "center" function to align itself with the center of the black line by calculating the error value and determining whether pixels are black or white. If there are no black pixels, the robot turns right or left based on the error direction. This quadrant starts by defining arrays for storing pixels in different areas of the image. The main function captures the image and calculates error values for each area. It searches for black pixels and their positions, then calculates the average error position. Intersections are detected based on the number of black pixels and the range of errors in the horizontal line. The count variable keeps track of the intersections, and the robot turns in the desired direction based on the count. When no intersection is detected, the robot moves forward along the black line. The loop continues until red pixels are detected, indicating the presence of a red square and the need to switch to the next quadrant.

➢ **Quadrant 4**

The completion of quadrant 4 is not possible due to the short timeframe. However, if given additional time, our proposed technique would involve adapting the code used in quadrant 2. The robot would be designed to recognise and approach red pixels rather than black pixels. The number of red pixels sensed increases as the robot approaches a red cylinder. If the number exceeds a specified threshold, the robot will come to a halt, reverse, and seek for the next cylinder. This technique would be repeated for the three red, green, and blue cylinder towers.

The robot would scan the center row of pixels to determine the color of the cylinder for precise detection and navigation. If the center pixel matches the center of the screen, the robot comes to a halt and approaches the cylinder. If this is not the case, the robot will keep turning until alignment is established. This method ensures precise motions. This procedure is carried out for each cylinder tower. Similarly, when a red ball is identified, the robot moves forwards until no more red pixels are seen, indicating that the ball has been successfully displaced.

➢ **Quad Change**

Algorithm:

1. Initialize variables to store the red, green, blue values and store the number of red pixels
2. Start a for loop ← to iterate through each column of pixels
3. totRed = (int)get_pixel(row, col, 0) ← Get the red, green, and blue color values of the pixel (example provided gets the red value of the pixel. Repeat code for the blue and green but change the channel number).
4. redness = (double)totRed/((double)totRed + (double)totBlue + (double)totGreen); ← Calculate redness, greenness, and blueness ratios. (example provided gets the redness ratio)
5. if((redness > 2*greenness)) ← Compare the pixel color using ratios. If the redness is bigger than 2*greenness,
6. numberofredpixels += 1 ← increment the number of red pixels.
7. Outside of the for loop, check if the number of red pixels exceeds a certain threshold to determine whether a red rectangle (indicating a quadrant change) is present
8. If the threshold is exceeded, print "Switching to next quadrant" and return false

When used in the main function, it changes the quadrant code. When the robot is following the black line, there will be red squares on the map to indicate that it is time to swap quadrants. When the robot recognises a particular number of red pixels, it calls the quad_change() function. We determine the number of red pixels by counting the number of red, green, and blue pixels and comparing their redness, greenness, and blueness. If the ratio of redness to greenness and blueness is greater than a specific threshold (in the detected pixel), the number of red pixels is increased by one. When the number of red pixels is bigger than 1000, the robot will switch its algorithm and use the next quadrant's algorithm.

Testing

Hardware

Not much testing is required for the hardware except for the fact that the finished product must be short enough to fit through the gate and complete the first 25% of the course.

Software

- Opening the gate (Quadrant 1)

  In order to test this particular section, the robot is positioned behind the gate prior to executing the 'gate_open' function found in the 'gate.hpp' header file. If the gate opens then it has worked

- Quadrant 2

  In this quadrant, the robot is required to follow a black line. To test this, the robot is placed on the white mat with the black line. When the robot runs on the black line and adjusts itself based on the position of the line, the code and PID are both successful. The Kp number will be modified depending on the smoothness of the robot's movement.

- Quadrant 3

  The testing methodology employed in quadrant 3 is similar to the one utilized in quadrant 2. However, the presence of intersections introduces additional complexities that necessitate further testing. In this scenario, the robot must effectively identify the existence of intersections and determine the appropriate direction to turn at each intersection. When the robot successfully

completes quadrant 3, the code and intersection detection are successful. This means that the robot can determine which way to turn to complete the quadrant.

- Quadrant 4

  Several parts of this section require testing due to the differences with the other section. Even though it implements a similar method to quadrant 2, the color detection and the movement of the servos (turn, approach, stop, reverse) needs to be tested. The effectiveness of the code implementation is dependent on the robot's ability to carry out the intended operations and eventually push the ball off the table.

- Quadrant change

  Because the robot will detect a red square when the map moves into the next quadrant, the detection of red pixels should be checked by printing 'switching to next quadrant' when it sees a red square, indicating that the code is successful and the map will move to the next quadrant.

Results

Hardware

The prototype of the robot was mostly successful on the first attempt unless for the fact that there are slight measurement errors in the 3D printed brackets which is slowing the testing down. Other than that, the robot's camera is properly working as intended, and it fits through the gate.

Software

- Quadrant One (Opening the gate)

  The only issue found is not sending the password back into the server. After a few tests, the function was fixed and ran without errors.

- Quadrant Two

  Detecting the black and white pixels in order for the robot to follow the line works successfully and runs without errors. The Kp and adjustment value for controlling the motor needs to be more accurate. The test is calibrated through trial and error.

Kp Value	Results
0.1	Overreacts. Unable to stay on the line.
0.08	Still overreacts & stops on the curves.
0.05	Stops on the curves due to the turns (still overreacts)
0.02	Follows the line & goes through the curves smoothly & accurately

- Quadrant Three

This part employs counts to determine the number of intersections and which way to turn during each intersection in order to navigate through them. For example, turning left when the intersection count is one. During the testing, it was discovered that turning left at every intersection is more effective, except for the last one where it should turn right. A delay factor was implemented to be more exact with the count intersections so that the robot can precisely define if there is an intersection or not, as well as to make the robot's turns sharper and more precise. During testing, the sleep value of 50 is used, with the left motor being 51 and the right motor being 45, to allow the robot to move slowly in order to obtain a more accurate turn and direction.

- Quadrant Four (Challenge)
  This quadrant was never fully developed. In the tests that were run, the robot could identify that it is switching to the fourth quadrant. However, due to time constraints, the robot only managed to approach the red cylinder without stopping (it kept pushing the tower).

Discussion

Team

The success of the vehicle project relied on the team's organization and communication. Initially, roles were assigned based on individual interests and knowledge, but adjustments were made as needed. However, towards the end, having only two coders limited testing due to technological constraints. To improve overall performance, better planning and a more inclusive coding strategy were needed. By involving all team members and adopting a collaborative coding approach, they could have utilized everyone's strengths and saved time. Assigning one team member to each quadrant would have been a more efficient strategy. Other successful teams followed similar approaches, valuing input from all members. In summary, effective organization, communication, and an inclusive coding strategy would have enhanced the team's performance, leading to greater success in the project.

Hardware

Our robot's hardware design comprises a layered approach that provides various advantages in terms of organization and functioning. It was carried out exactly as expected and successfully completed the course with few difficulties. The camera is positioned on the top layer due to the robot's elevated height, offering a clear view of the colors of the white and black line while minimizing visual input from other portions of the course. This design choice simplifies algorithm implementation and testing, allowing the vehicle to recognise the black line and intersections accurately. By raising the camera, we improve the accuracy and clarity of the black line acquired by the sensor, hence improving the robot's overall performance.

Software

The software played a critical role in the project, although the final code was not fully completed or optimized. The code was organized into separate files for each quadrant, allowing easy testing and improving code readability. However, there were areas that could have been enhanced in each quadrant.

In quadrant one, the robot successfully passed through the gate but lacked redundancy in checking if the gate was fully open. Quadrant two implemented line following using a PID algorithm, but only the proportional term was utilized. Incorporating the derivative term could have improved error detection. Quadrant three had success, but intersection counting and handling dead ends could have been improved. Quadrant four's code was incomplete due to time constraints. Overall, while the software had its successes, there were missed opportunities for optimization and improvements throughout all the quadrants.

Future Work

Despite having met 80% of the project's objectives, there is still 20% that needs to be explored further. This involves addressing quadrant 4, which was unable to be finished within the timeframe provided. Adapting the code used in quadrant 2 to recognise and approach red pixels rather than black pixels is one of the proposed solutions. Furthermore, it is critical to improve the detection and navigation algorithms for exact alignment. By committing further resources and time to investigate these alternatives, the project aims to achieve its full potential and fulfill the remaining goals.

Conclusion

Overall, our robot did not meet all of the required criteria because it only completed 80% of the route, which is different from the initial expectation of completing the entire course. In terms of the challenge, potential utilization of new methods and approaches was considered. However, limitations in time and external factors, such as system downtime, impeded the pace of testing, ultimately preventing the vehicle from finishing the course due to insufficient time. Despite the absence of a 100% completion as originally planned, the true success of this project lies in the lessons acquired throughout the process and the insights gained from project failures. A failure allowed the team to think about reasons behind shortcomings and recognize the significance of effective planning and time management. Communication and teamwork are required for a team to function effectively and without undue stress.

References

Small, J., Pho, R., Salim, E., Ran, J. (2023-06-13). Avc project.
Retrieved June 13, 2023, from:
https://gitlab.ecs.vuw.ac.nz/course-work/engr101/2023/project3/t5/avc-team-5.git

Roberts, A. [Arthur]. ENGR101 t1 2023 - Engineering technology Lecture 14 slides
Retrieved June 13, 2023, from:
https://ecs.wgtn.ac.nz/foswiki/pub/Courses/ENGR101_2023T1/LectureSchedule/ENGR101_Lecture14.pdf

Roberts, A. [Arthur]. ENGR101 t1 2023 - Engineering technology Lecture 15 slides
Retrieved June 13, 2023, from:
https://ecs.wgtn.ac.nz/foswiki/pub/Courses/ENGR101_2023T1/LectureSchedule/ENGR101_Lecture15.pdf