AVC REPORT 2023

Name: Michael Visser

Student ID: 300652084

ENGR 101

Lecturer(s): Howard Lukefahr, Arthur Roberts

Report on the 2023 AVC robot project for ENGR121

Team 26

Abstract:

The Victoria University Engineering course contains a well-known challenge called the Autonomous Vehicle Challenge (AVC) which is where multiple groups get together to create a robot to complete a set course. The course contains sharp turns, dead ends, and several obstacles that the robot must complete with no human input during its run of the course. The teams have very little experience with the hardware and software as well as working with people that they may not know, making this a difficult task that spans several weeks to complete. This report belongs to Team 26, known as “Low Riders”. On Tuesday in Lab A, we managed to complete quadrant 1, 2, and 3 smoothly, but in quadrant 4, Arthur had to slightly interfere with the robot to get it to complete the course.

1. Introduction:

1.1 Scope:

The scope of this report covers the process, execution and result of the AVC project. It covers the content learnt in previous labs and lectures from the ENGR 101 course. Each team is made up of 4-5 random students put together to build and program a robot to complete the set course. For this project 5 weeks are given with specific lab times as well as open facilities at other times as well. All materials and equipment for the robot are given like continuous and non-continuous servos, Raspberry PI 0W, parts and equipment for robot construction like 3D printers, a custom PCB, a Raspberry Pi camera, and a battery to power the robot.

1.2 Motivation:

The problem to be solved was to create a robot that will open a gate via Wi-Fi, follow a line, complete a maze, and drive towards specific objects. During this we learnt how to create parts using FreeCAD and how to 3d print the created parts, how to troubleshoot a 3d printer, and ultimately how to code the robot to complete the tasks given.

1.3 Aim:

The robot needs to be able to navigate autonomously through the set course. The course consists of 4 quadrants. Each of these quadrants have a different challenge for the robot with a different approach required. The first quadrant consists of a gate that the robot needs to wirelessly communicate to for the gate to open. Quadrant 2 consists of a black line that the robot needs to follow using the camera. Quadrant 3 consists of multiple intersections with dead ends which the robot must navigate in order to complete the quadrant. Quadrant 4 consists of 3 different colored towers that the robot must navigate to in order and then push over an obstacle to complete the course.

1.4 Objective:

Construct a robot that is autonomous using the given equipment. Use the set course to create a program that completes the set course. Create a program using the C++ language using the given E101 Library to autonomously move the robot from start to end. Present a finished project that can complete the course by Thursday 1st June.

2. Background:

2.1 Hardware:

- **Raspberry Pi 0W:** The Raspberry Pi 0W is the main processor and controller for the robot. It consists of a single board computer programed using C++ to process the input from the camera. Commands are then given to the custom PCB which gives information to turn the motors.
- **Servos:** Two different types of servos are used depending on what task is to be completed. Two continuous servos serve as the main motors as they do not stop turning unlike regular servos, making them great motors. One non-continuous servo is used to move the camera angle depending on what quadrant we are in.
- **Custom PCB:** The custom PCB has a 10-bit ADC, an 8-channel digital input/output used to control the servos. It also powers the Raspberry PI using 5V DC power and 3.3V to the servos.
- **Camera:** The camera is used to send images to the Raspberry Pi. This is then processed on the Raspberry Pi with the code running on it.
- **Battery:** The battery is used to power the Raspberry Pi, and everything connected to it. The battery was connected via a USB to Micro-USB cable delivering 5V to the Raspberry Pi.

2.2 Software:

We had to use C++ to create the software for the robot. This would be run on the Raspberry Pi on the robot, meaning it could access all the hardware attached to it like cameras and servos to create what we need to complete the course.

2.2.1 Following the line:

In quadrant 2 the robot is required to follow a line that is windy and hard to follow. In a report written by Aarav G [1], he explains to use “Two infrared proximity sensors [Not available for this project but solutions available] to detect the line”. Using sensors, the robot can move forward when both the left and right sensors don’t detect black, turn left when the left sensor detects black and the right does not, and turn right when the right sensor detects black and the left does not.

The same sort of sensor can be used for quadrant 4 as well. By splitting up the image processing, we can check the left and the right to see if we are on course.

2.2.2 Navigating a maze:

Quadrant 3 requires the robot to navigate a sort of maze on the ground. With black lines creating multiple intersections with dead ends, the robot must navigate to the end to move on to the final stage. An article written by Md Mobshshir Nayeem [2], He explains something called the “LSRB algorithm”, which simply states, turn left if it is an option, go straight if left is not an option and straight is, turn right if left and straight aren’t an option and right is, and do a U-turn if left, straight, or right aren’t options. This algorithm guarantees that the robot will make it to the end of the maze even though it may not be the most efficient path.

3. Method:

3.1 Hardware:

All 3D models uploaded to a group GitLab account with the models stored on the branch “main”. (Alexander Heffernan, Michael Visser, Cara Lill, 2023-06-13).

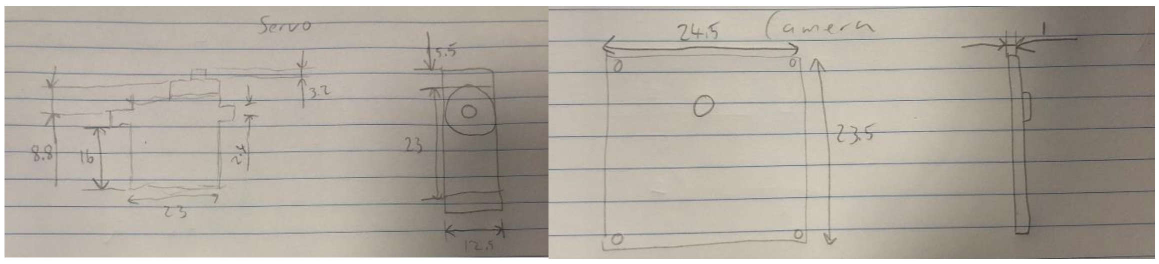
3.1.1 Components Used:

- **Raspberry Pi 0W**
- **Custom PCB**
- **Raspberry Pi Camera**
- **Battery**
- **USB to Micro-USB cable**
- **2x Continuous Micro Servo Motors (FS90R)**
- **1x Micro Servo Motor (MG90S)**
- **2x small wheels**
- **1x acrylic plate**
- **1x 3D printed baseplate**

- **4x 3D printed poles**
- **1x marble**
- **1x 3D printed marble holder**
- **3x wheel mounts**
- **1x 3D printed camera mount**
- **1x 3D printed camera servo holder**
- **4x 40mm M3 bolts and nuts**
- **4x M3 screws**
- **2x 10mm M2 bolts and nuts**
- **2x M2 screws**

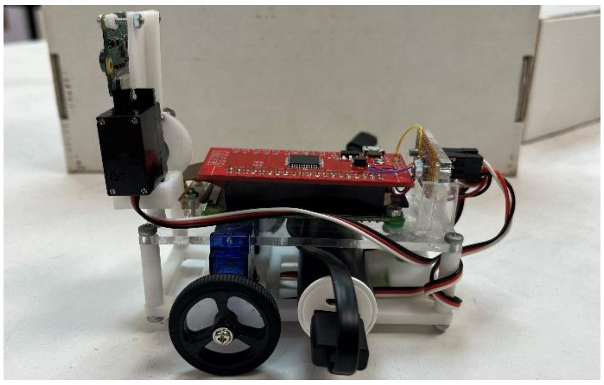
3.1.2 Construction:

- **Design:** Diagrams of parts were created so we knew the exact sizes of each individual part. This was used to aid in the creation of 3D printed parts so that each part could fit precisely. The diagrams were sketches of each part given like servos, camera, marble and the main acrylic base plate.



- **3D Printing:** Most of the extra structural parts used on the robot were made using a 3D printer. A program using FreeCAD was used to design the parts from scratch. Measurements taken from the parts aided in the design of each part and we made sure to double check our measurements and design so that no unnecessary waste was created. Some parts failed but a simple recalibration of the printer fixed most of our issues. The camera servo mount broke when we tried to attach it to the acrylic plate on the robot, but we managed to force the screw in and still use the part as intended.
- **Robot:** There are two main plates on the robot that everything connected to. One was an acrylic plate that the Raspberry Pi, custom PCB, camera servo, and the camera connected to. The other plate was the biggest 3D printed part on the robot which the marble, wheel servos, wheels and the battery connected to. The 3D printed plate was below the acrylic plate joined together with 4 3D printed poles which made a raised platform. This space was also the perfect size for the motors to be stuck between the plates so they wouldn't move so that no extra mounting hardware had to be used. The marble was near the back of the robot in a cutout in the 3D printed plate. This was used so that the robot could slide at the back when it needed to turn. It was secured with an additional plate that would clamp it down so it couldn't fall out. The battery was placed in between the motors and the marble. This was secured by having the battery cable above the battery, so it was stuck

between the two plates. This also meant that the removal of the battery was easy, and no extra mounts were needed. It also put the needed weight at the bottom of the robot so that it had grip but wouldn't fall over to one side. The camera servo was on a raised 3D printed mount at the front of the robot so that it had a good view of the line but also in front of the robot when the camera was rotated. The camera was mounted to the servo using a 3D printed mount that screwed into the back of the camera. This meant that the camera could be rotated when needed for each quadrant. The Raspberry Pi and the custom PCB were mounted on the acrylic plate so that they could be easily accessed at the top of the robot when needing to plug in cables.



The robot was designed around what tasks it was needed to complete. The robot had to be small enough to fit through a raised gate at the start of the course. It also had to have a good weight distribution for it to stay upright. The camera needed to be able to pivot up and down so it could transition from following a line to seeing straight ahead. This meant we needed to put it in the center. All parts needed to be easy to access and strong enough or easy to repair in case something happened.

3.2 Software:

The software was uploaded to a group GitLab account with the code stored on the branch "main". (Alexander Heffernan, Michael Visser, Cara Lill, 2023-06-13).

The software has 4 sections for each quadrant in the course. At the start of the code, the camera is tilted down to the correct position and is turned on.

The software finishes only when all 4 quadrants have finished.

3.2.1 Quadrant 1:

In quadrant 1, the robot has to communicate with the gate over Wi-Fi. This is done by first requesting to connect to the server at 130.195.3.91, port 1024, with "Please". The server then sends back a verification password which the robot then sends back to the server to verify connection. The gate is then opened and the motors on the robot move forward for 2.5 seconds. This is the end of quadrant 1 and the robot moves onto quadrant 2.

3.2.2 Quadrant 2:

To follow the line, the robot uses an on-off loop. This is done by turning right whenever the error gets too high, and turning left when the error is too low. If the error is close enough to 0, the robot will drive in a straight line until the error changes. We made the line following fault tolerant by detecting if no black pixels are on the screen. When this happens, the error will be equal to 0. The robot will reverse if the error is equal to 0 until it finds the black line again. The error is calculated along the middle row of the image output. The software then adds the pixel number of each of the black pixels to the error. Pixel number 0 is in the middle of the camera.

3.2.3 Quadrant 3:

By using the LSRB algorithm previously discussed, the robot makes its way through the maze. This is done by scanning the image the camera takes and calculating top. Left, and right errors. This calculates the next move the robot will make deciding where it will go.

3.2.4 Quadrant 4:

Using the on-off system described before, the robot navigates this quadrant by finding the colors red, green, and blue. The robot approaches the cylinder by turning until it is visible. Then the robot will approach the cylinder until the pixel count of the cylinder becomes high which means it is close enough to the cylinder to go to the next one. Once it has done this to all 3 cylinders, it will then locate the ball and move close to the ball. The robot will only stop until the ball is gone which means that it has pushed it off the edge. Once the ball is gone then the robot will stop the code, completing the course.

4. Results:

4.1 Hardware:

The hardware for the robot was exactly what we were striving to do. That being small, light, and minimalistic. There were some strengths and weaknesses that were present though. These are outlined below:

Strengths:

- **Small:** The robot was small which was important in order to make it through the first quadrant
- **Light:** The robot was light which meant combined with the size it was very maneuverable throughout quadrant 2 and 3.
- **Custom Parts:** These custom parts made it possible for the robot to have precise fitting parts. This meant that we didn’t have to modify anything as it all fir perfectly.

- **Cable management:** The robot had good cable management meaning that the wires would not get stuck at all in any moving parts. This meant nothing would accidentally disconnect during a run of the course.

Weaknesses:

- **Camera Placement:** Because of the height of the robot, this meant that the camera was very low to the ground. If the camera was up higher, we would be able to detect the line more accurately as it was taking up most of the image output in the position it was in.
- **Marble Holder:** This holder was originally designed so that the ball could move freely inside of the compartment it was in. In the end this was not the case as the marble holder was too tight for it to move freely, so it was just dragging at the back of the robot, but it was still enough for the back to freely slide.
- **Low Design:** If the surface of the course was not flat then this would have caused issues but since the set course was flat there no issues with this weakness.
- **Inaccurate Motors:** The provided motors were inaccurate and required constant adjustment for the movement to be consistent. If the robot was left for too long, then it would not be able to drive in a straight line or in reverse.

4.2 Software:

The code in C++ for the robot was successful. It managed to complete the entire course with only a little bit of interference near the end in order for it to complete the course.

4.2.1 Quadrant 1:

During testing, we figured out that the robot was not fast enough to get through the gate before it closed. This meant we had to implement in quadrant one a piece of code that would make the robot drive in a straight line for 2.5 seconds. This also meant that the robot had to be placed in the correct direction or it would lose sight of the line before quadrant 2 could be activated.

4.2.2 Quadrant 2:

While the robot did make it through this quadrant, it was oscillating back and forth making it slow to get through it. This was caused by the robot trying to align itself with the line but overshooting and then trying to align again. The robot, however, still made it through with plenty of time left on the clock.

4.2.3 Quadrant 3:

After a lot of testing, the robot made its way through quadrant 3 very accurately. The low hanging camera on the robot means that a lot of refinement was needed to get the robot to accurately make its way through this quadrant. This was because it was struggling to distinguish between different pathways making it choose the wrong one. Constant testing and refinement though made the robot able to detect these intersections accurately. The decision on which intersection to take was followed using the “LSRB algorithm” [2].

4.2.4 Quadrant 4:

Quadrant 4 did not go as planned in the final marking stage. Some slight interference from the marker was needed for the robot to reach the end. This was caused by the robot mistaking the red ball at the end of the course as the red cylinder and as such drove straight to the end of the course. The marker moved the red ball to the side and the robot was able to complete the course. The time spent on quadrant 3 took up most of our testing time and as such there was not much time to refine quadrant 4. With more time this fault would have been easy to resolve.

4.3 Overall:

Overall, the robot performed well through quadrant 1, 2, and 3. Some improvements could have been made throughout quadrant 2 and 4. The slight interference needed from the marker took off 2% of our grade but the robot managed to make it to the end of the course in time.

5. Discussion:

5.1 Hardware:

While the hardware designed was effective, there could have been some improvements to aid in the development of the software. Below are some improvements that we could have made to aid in the development of the software:

- **Camera Placement:** Due to the low camera, the robot had very sensitive line detection. Raising the height of the camera would have improved the accuracy of line detection as we would get a bigger picture. We could have achieved this using either bigger wheels or a higher mount.
- **Marble Holder:** Making the inside of the marble holder smoother and not as tight would have improved the control of the robot as it wouldn’t be dragging across the ground.
- **Low design:** The low design on the robot meant that it was more likely that the robot could catch on to something. Bigger wheels would have solved this issue.

5.2 Software:

The software was able to guide the robot across the whole course, but some improvements could have been made to fix issues like oscillating and efficiency.

5.2.1 PID Controller:

The Proportional-Integral-Derivative (PID) controller is used to prevent the robot from making disproportionate or oscillatory changes to the black line it is following. This controller is capable of providing an exact and timely adjustment to the control function, therefore enabling the robot to “take different types of curves at different curve radius without getting off the line” [3]. 3 of the 4 quadrants would benefit massively from this controller as the robot needed to take sharp turns and make small adjustments on straight lines.

5.2.2 Motor Speeds:

The speed of the robot could have been greatly improved with testing of different motor speeds. In the early stages we stuck with speeds that we knew worked well. Instead of this we should have found the fastest speed we could use through testing without making the robot inaccurate or unreliable.

6. Conclusion:

In conclusion, the project was a success as the robot was able to navigate itself autonomously around the set course. With some refinement and testing, the robot was able to complete the course quite successfully. Though even with success, there are areas where mistakes were made and can be improved upon for further iterations of the robot.

When we first started the project, our main goal was to build a robot that could autonomously traverse the course given. We had to rapidly learn how to use the materials and software given to build and control the robot. With this project, we achieved our goal of having the robot autonomously traverse the course. We also learned an abundance of skills and tips on how to control and use the hardware and software given.

References:

[1] Aarav G, “how to make a line follower robot using Arduino”, Autodesk Instructables, 2018. [Online]. Available: https://www.instructables.com/Line-Follower-Robot-Using-Arduino-2/. [Accessed Jun. 11, 2023].

[2] M. Nayeem, “Coding a line following robot for maze using LSRB algorithm and finding it’s shortest path”, towardinfinity.medium.com, Jul. 7, 2019. [Online]. Available: https://towardinfinity.medium.com/coding-a-line-following-robot-using-lsrb-and-finding-the-shortest-path-d906ffec71d. [Accessed Jun. 11, 2023].

[3] B. Reboot “Line follower robot (with PID controller)”, huckster.io, Aug. 11, 2020. [Online]. Available: https://www.hackster.io/anova9347/line-follower-robot-with-pid-controller-cdedbd/. [Accessed Jun. 11, 2023].