AVC Report

ENGR 101, 2017 T1

Name: Niels Clayton
Student Number: 300437590
Lab day & time: [Friday, 11:00am – 1:00pm]
GitHub: https://github.com/Matt-Rothwell/avc-2018/

Abstract

We have been challenged to design and code a robot that can autonomously complete a maze. We have been paired with random people from out ENGR101 course and must work coherently together to achieve our goal of completing the maze. We have been given four weeks to complete this project, and have been supplied with a Raspberry Pi, a PCB which acts as an analogue to digital converter, two motors and wheels and a camera. Additionally, we were given the opportunity to buy more parts from the 'parts bazar' such as sensors, with the allotted virtual currency we have been given. We also have access to 3D printers, where we are allotted ten 3D prints to help in the construction of our robot. From this project we will gain hands on experience in designing and planning a project in a group environment, and communication skill to help us work well together as a team. These skills will be applicable to real world situations and will be very valuable.

Introduction

Scope

This challenge is designed to introduce us to the fundamental concepts of electronic engineering along with the basics of robot design. Some of these skills will have been learnt in previous lectures and labs and will enable us to create an autonomous vehicle to complete the maze. We will also be introduced to 3D modelling for the creation of parts used in the construction of the robot, as well as learning how to work in a team of 5 people to meet a deadline while keeping within a budget.

Aim

The goal of our project is to work together with a randomly selected group of people to build and program a robot which is autonomously able to complete the challenge that we have been set. Our challenge is a maze comprised of 4 quadrants.

Quadrant 1:

The robot must connect to the server controlling the gate remotely, ask for a password by transmitting the string "please" to the server, receive the password, and then transmit it back.

Quadrant 2:

The robot will need to be able to follow a line. The line will have curves in it and the robot should be able to successfully follow without losing track of the line.

Quadrant 3:
The robot will once again follow a line, but this line will have right-angle turns rather than curves and will have multiple paths that the robot will have to choose between. Some of the paths will lead to the next quadrant, and some will be dead ends.

Quadrant 4:
The robot will enter a physical maze with no line to follow anymore. Here it must rely on sensors which we are able to purchase 'parts bazaar'. This section of the maze also has a gate the opens and closes on a timer, so the robot must be able to recognise and react to the opening and closing, and time when it can pass through.

Objectives and motivation
The motivation for completing this challenge is that the aspects of this task are all key for any engineering degree. Because of this, by completing this course we will have gained valuable skill that will not only allow us to continue within our engineering degree but will also be applicable within the industry. The most obvious of these is the group work experience. In the field of engineering collaboration and working with others affectively to follow a brief while under monetary and time constraints will be key. The experience to use 3D modelling and printing equipment will also be useful for future engineering assignments and will allow us to place another skill under our belts for the future.

- Our objectives are to understand the basic functionality of the PID correction system, and how it works.
- Work affectively as a team to meet a deadline, to produce an aesthetically pleasing, under budget autonomous vehicle that can follow a line, communicate with servers, and navigate the maze.
- Develop the vehicle within the monetary and time constraints place on us.

Background

Our task is to design and build an autonomous vehicle to traverse and complete a four-quadrant maze. The first quadrant requires the opening of a gate controlled by a remote server. The second quadrant consist of a white line placed on a black background with curves and turns in it. The third quadrant consists of the same white line but rather than curves it now has right-angle turns and T-junctions that implement choice into the direction which the robot can take. Finally, the forth quadrant consists of 100mm high walls which the robot must traverse through in order to reach the finish. The robot must also be aesthetically pleasing to a consumer (not be a mess of wires, and parts), and it must be recyclable (the hardware must be reusable so no modifications to supplied parts such as sensors, and no glue as this would stop recycling). On the final challenge day, we will be given 15 minutes and five attempts to prove to our customers that we meet each of these specifications. Once the robot has started the course it may not be touched, although it can receive a small nudge from a tutor of it gets stuck, however each nudge will deduct one point from our final mark. A full top down view of the course can be seen in figure 1, including a labelling of the grade that will be awarded for completing the course to specific points.

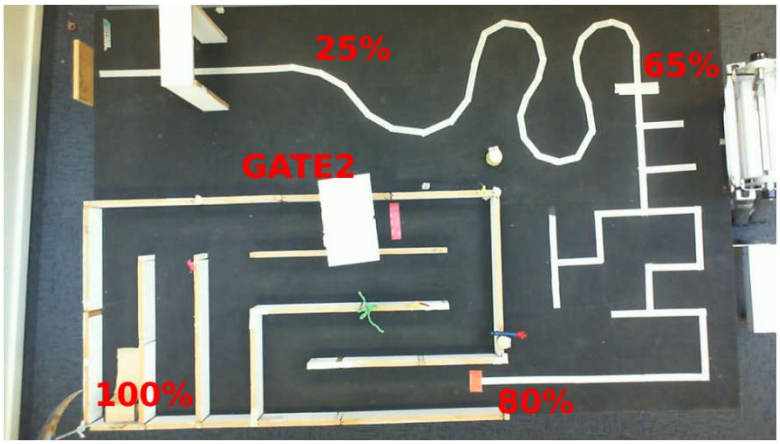


Figure 1:

Top down view of the course that our robot will need to navigate

The robot will be constructed from several different components. A Raspberry Pi connected to an analogue to digital converting PCB, to which we will attach our DC motors, a camera, and either short or long-range infrared sensors.

A raspberry pi is what is knows as a SoC (system on a chip) and is a small Linux based computer with all the basic requirements of a computer all on the same board. Raspberry pi's have been developed to be a cheap educational tool for students and for anyone that wants to use them and are easy to purchase online. We will be using the raspberry pi as the 'brains' of our robot and will be doing the majority of our coding on it.
An IR sensor (infrared sensor), emits infrared light, and then measures the percentage of light reflected back to measure the distance to an object. For this AVC project we will be given access to both short and long-range sensors. The short-range sensors return a digital reading (either 1 for if there is something close, or a 0 for if there isn't). The long range sensors return an analogue signal that represents a voltage, which will be greater the closer an object is to the sensor. Determining the actual distance to an object is from the sensors is difficult, this will be discussed in the method. We also found that our sensors did not return the same values at set distances, meaning that this bias would need to be accounted for in the code.
The DC motors we are using are have only two settings, either on or off. Because of this in order to control the speed of the motors pulse width modulation (PWM) is used. This is done using a square waveform (either a high voltage or no voltage), and then altering the duty cycle of the wave. This means that if the duty cycle of the wave is 75% the motor will be on for 75% of the time, and off for 25% of the time. This modulation simulates the affect of the motor being on 75% speed.

Finally, in order to control the robot we are using a range of different control systems. The one we have been using is the PID control system. This stands for proportional, integral, derivative controller. This controller calculates the overall 'error' within the system based either on where the line is under the robot, or where the walls are relative to the robot. This controller first calculates the proportional error of the system, this is the direct error that the system measures initially. It then calculates the derivative error, or the rate of change of error, this is done by using stored values, and allows you to mitigate the oscillation between a positive and negative error. Finally the integral error calculates the total change in error over time, to see if the previous steps have decreased the overall error compared to a previous state. These all work in conjunction to output a final error signal that is then used to alter the state of the system (in the case the position of the robot).

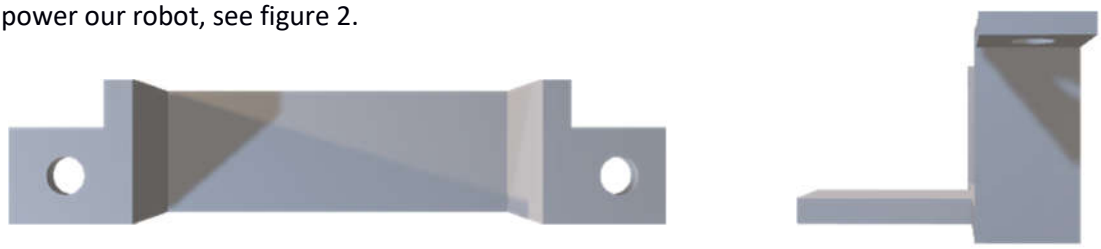
Method

As a basis for our robot on which to build, we have been given:

- Raspberry Pi
- PCB including an H-bridge, and a 10-bit ADC
- 2 motors (each with a wheel)
- Wireless dongle

We have also been given access to 'parts bazaar' where we have up to $100 to spend on sensors we may believe help us complete the challenge we have been given.
Our team has opted to purchase three long range IR sensors. These three sensors will be places on either side of out robot, and one in front of the robot to allow for maximum sensor coverage.

Our robot has been designed trying to keep it as compact as possible. This has been done to ensure that our robot remains as manoeuvrable as possible and will be able to travers as many possible locations as possible. To achieve this, we have chosen to use the smaller radius wheels rather than the larger ones. Although this may cost our robot speed, as one rotation of the motor will not be able to cover as much distance, it allows out to place the wheels directly under the robot without collision with the chassis. With this motor placement we have achieved a very tight turning circle and kept the motor frame height to a minimum. This increases the overall manoeuvrability of our robot greatly, allowing for sharper and more precise turns in quadrant C.
On the underside of our robot we have 3D printed a brace in which to hold a slim battery to power our robot, see figure 2.



We have placed the battery in this position to avoid increasing the height of our robot. Since out robot has a very small wheel base, if it becomes top heavy it will become very easy for it to topple over when making sharp turns. By placing our battery on the bottom, we shift the centre of mass closer to the wheel base increasing stability and manoeuvrability.
As a part of this battery brace we have also incorporated two smooth rounded back supports which the robot will drag behind it to keep the frame lifted off the ground. We have chosen to use this design as the marble holder we previously printed out was too bulky, no longer allowing us to store our battery on the bottom of our robot.

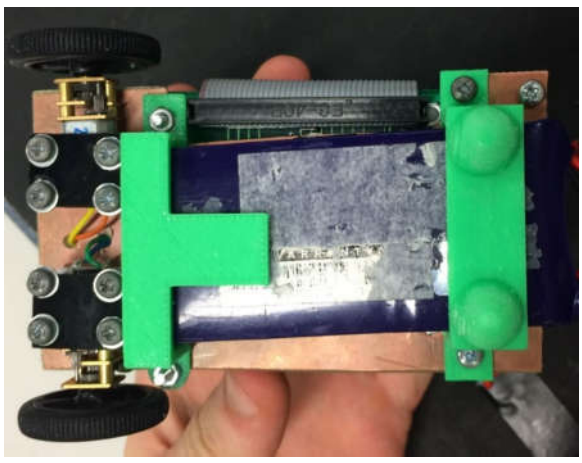


Figure 2:
Base mounted battery holder.

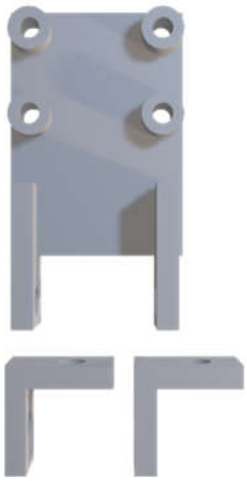


Figure 3:
3D model of camera mount

We have mounted our IR sensors on either side of our robot, and one underneath the raspberry pi mounted on the from. These placements will allow us to measure the distance on either side of the robot, and weather there is a wall in front. These sensors have been mounted using small 3D printed brackets that hang over the edge of the robot, reducing their overall impact in the width or height of the robot. The camera mount seen in model 3 was mounted into of the robot at an angle of 30 degrees looking down. to provide a large field of view, and to allow us to see what is more than directly in front of the robot, allowing it to react in time for the turns. This placement is seen in figure 4.

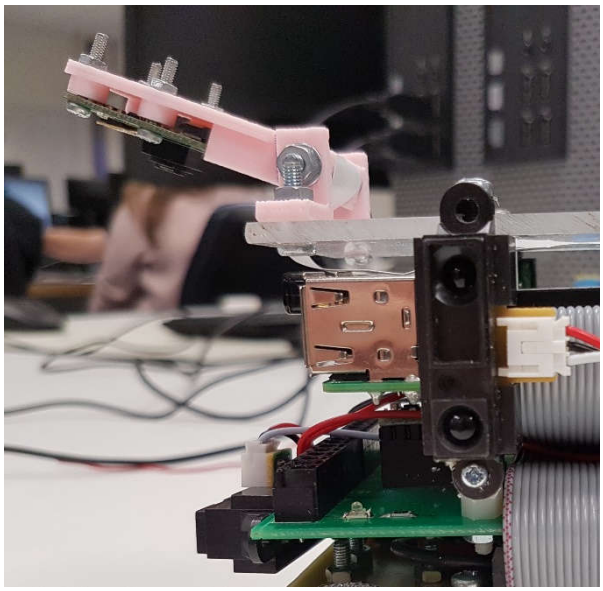


Figure 4

In order to meet the aesthetic requirements we have also included a physical switch located on the top of the robot in order to turn it on or off. This will improve the user experience over all and will make operation much easier. We have also attempted to shorten any cables on the robot.

In our code we have separated each step of the process into separate methods that are called on, this has structured our code and made it much simpler to understand and replicate. For the calculation of the error we have implemented the PID control system. This first begins with the taking of a picture which is then passed through to the calculate error method, see figure 5. The calculate error then take this picture and passes it to the get_colour_threshold method. In this method a for loop retrieves each of the black level values in row 120 of the picture and stores the minimum and maximum values for the picture. These values are then summed and divided by two to return the threshold. This threshold is then passed through to the get_white_pixels method, along with an array and the picture. The get_white_pixels methods then runs through the picture checking the black levels of each pixel, if the black level is below the threshold then it will add a 0 to the array, and if it is above the threshold it will add a 1 to the array. This array is then passed back to the calculate_error method where each white pixel is weighted depending on its positions from the middle of the array, see figure 5. It will then divide this error by the total number of white pixels and return it. This error is then used to set the motor speed, and then stored in the previous error field. This then allows us to pass through the previous error and the current error to the calculate_pid method which calculates the derivative of the error and takes it away from the current error to give the final error to be applied to the motors. This error calculation system allows us to follow the white line without losing it, but in case we do lose it we have implemented that the robot will reverse till a line is detected again, and then it will begin to follow the line again.

In quadrant 3 the code to follow the line is the same however we now also read the pixels down the columns 100, and 220. We use the same error detection method as we did with the horizontal row, but only implement it when the robot no longer senses a while line. When this is the case the robot switches of to reading the columns, and will turn to the side which it reads the greatest white pixel value. Because of this it will turn in the direction of

the right-angle turn until it recognises the horizontal white line again and then it will follow that.
In sector 4 the error signal was much simpler, we called the maze_error method that took reading from both left and right sensors and subtracted them, multiplied them by the value of kp, and then returned it. This allowed the robot to move down a corridor and stay centred. To make a turn in the maze we call the object_infront method, when this method returns true the robot will stop, turn in the direction of the greatest error till the wall in front is no longer sensed (the direction where there is no longer a wall), and then go forward again till it senses the wall again.

Results

During our ENGR101 lab we were given fifteen minutes to prove that our robot could navigate the maze to our customers. During this time we were able to pick up the robot and restart it at the beginning of the course if it were to get stuck or break down. We were also allowed to ask the tutors for a push if the robot were to become stuck, but each push would subtract one from the final mark.
Our first attempt on the course was no very successful as the robot only opened the gate, but then did not move. We determined this to be because of a flat battery, and asked for a replacement. Our next attempt was more successful, making it to the 75% mark before losing the line and falling of the edge of the table. On out final attempt our robot successfully made it through quadrants 1, 2, and 3 with no interference, and then in the maze it completed the first turn before becoming lodged in a corner and no longer progressing. This left us with a final total mark of 82%.

Discussion

Overall I believe that the project was successful, although we did not complete the course fully we made good used of the resources handed to us, and produced an aesthetically pleasing robot within the price range allotted. Our team did had issues with time management, as the final design of the robot was not yet complete till two and a half weeks in. because of this we had much less time to test our code than we would have liked. Our team also had issues with finding time to work on the robot together, with one team member failing to show up altogether. We found that the only day that we could all consistently meet was the day of our lab session. In order to counter this we created croup discord servers, google docs, and messenger chats to communicate with each other what it was we were working on. We also had a group GitHub on which we stored all of the iterations and changes to our code. This allowed us to all work on the code at home and then share our changed and vet which changes should be included.
Based on the final result it is clear that quadrant 3 is still requiring some small changes to the coding since it had an issue with a turn sometimes. This is because when reading the vertical arrays of pixels we only returned if there was a white pixel there, rather than an error signal. This would mean that sometimes a random white fleck on the black background might trigger the robot to turn left rather than right. This could be solved by creating an error signal, or even by returning the amount of white pixels on either side and making the robot turn in the direction of the side with the greater number.
We also had issues with quadrant 4. This is because we places the IR sensors symmetrically on either side of the robot. This meant that when it came to taking turns if it were to stop sensing the wall in front wile one sensors was pointing into the corner, and one sensor was pointing into the end of the turning corner. This would mean that the error value would be zero overall and the robot would no longer turn. This could be fixed by placing the sensors in

different positions on either side that that they are never able to read the same distance on either side.

Conclusion

To conclude, this project has been a learning experience in which we have had the opportunity to apply skills that we have acquired in lectures and labs to a real world problem. It has also given us the opportunity to experience what it is like to work with other on a group project and develop successful communication skills to be able to work coherently. This project has also demonstrated the difficulties of producing something under time constraints and money constrains to a standard that is usable to the customer. It does this wile giving us the freedom to time manage ourselves, giving us the opportunity to develop planning skills and management skill.

Overall this project has been an interactive learning experience that has introduced many new concepts, wats of thinking, and new technologies and a fun and testing way.

References

Foundation, R. p. (2017). *Raspberry Pi*. Retrieved from https://www.raspberrypi.org/

Wellington, V. U. (2017). *Intro to AVC, SSH and Compiling.* Retrieved from ecs Victoria: https://ecs.victoria.ac.nz/Courses/ENGR101_2018T1/LectureSchedule

Appendix

Figure 5

```
int calculate_error(int orientation)
{
    int error = 0;
    int wp = 0; // number of white pixels
    int pixels[(orientation == 0)?320:240];
    if(orientation == 0)
    {
        get_white_pixels(get_color_threshold(0),pixels,0); //initialise array

        for(int i = 0; i < 320; i++)
        {
            error += (i-160)*pixels[i];
            if(pixels[i] == 1)
            {
                wp++;
            }
        }

        if(wp == 320){
            return 20000; // case for if sensor only senses white
        }else if(wp == 0){
            return 10000; // case for if sensor only senses black
        }else{
            return error/wp;
        }
    }
    else
    {
        get_white_pixels(get_color_threshold(orientation),pixels,orientation); //initialise array

        for(int i = 0; i < 240; i++)
        {
            if(pixels[i] == 1)
            {
                wp++;
            }
        }
        return ((wp > 0) ? 30000 : 0);
    }
}
```

Figure 6:

```
int calculate_pid(int currentError, int previousError, long elapsed)
{

    /* Set kp and kd */
    double kp = 0.95;
    double kd = -0.35;
    double proportionalSignal = 0.0;
    double derivativeSignal = 0.0;

    /* Calculate Signals */
    proportionalSignal = (double)currentError * kp;

    derivativeSignal = (((double)currentError - (double)previousError) * kd)/elapsed;

    return proportionalSignal + derivativeSignal; // return final signal

}
```

Figure 7:

```
void get_white_pixels(int threshold, int pixels[], int orientation)
{
    if(orientation == 0)
    {
        for(int i = 0; i < 320; i++)
        {
            pixels[i] = 0; // pixel is black

            int pixel = get_pixel(120, i, 3);
            if(pixel > threshold)
            {
                pixels[i] = 1; // pixel is white
            }
        }
    }
    else
    {
        int col = (orientation == 1)?220:100; //check the left side first
        for(int row = 0; row < 240; row++)
        {
            pixels[row] = 0; // pixel is black

            int pixel = get_pixel(row, col, 3);
            if(pixel > threshold)
            {
                pixels[row] = 1; // pixel is white
            }
        }
    }
}
```

Figure 8: