Autonomous Vehicle Challenge

Project Report

June 4, 2021 Brandon Ru (rubran@ecs.vuw.ac.nz, 300562436)

Abstract

Building a small autonomous vehicle that utilises PID control theory within a team based environment. The use of PID controllers were mostly effective, with teamwork proving more difficult than expected. However, the experience was invaluable and was rich with learning.

1 Introduction

1.1 Aim

To create a vehicle that is able to pass the 4 part AVC Challenge track. The track consists of going through a networked gate (Q1), following a curved black line (Q2), navigating a line maze (Q3), and targeting coloured objects (Q4).

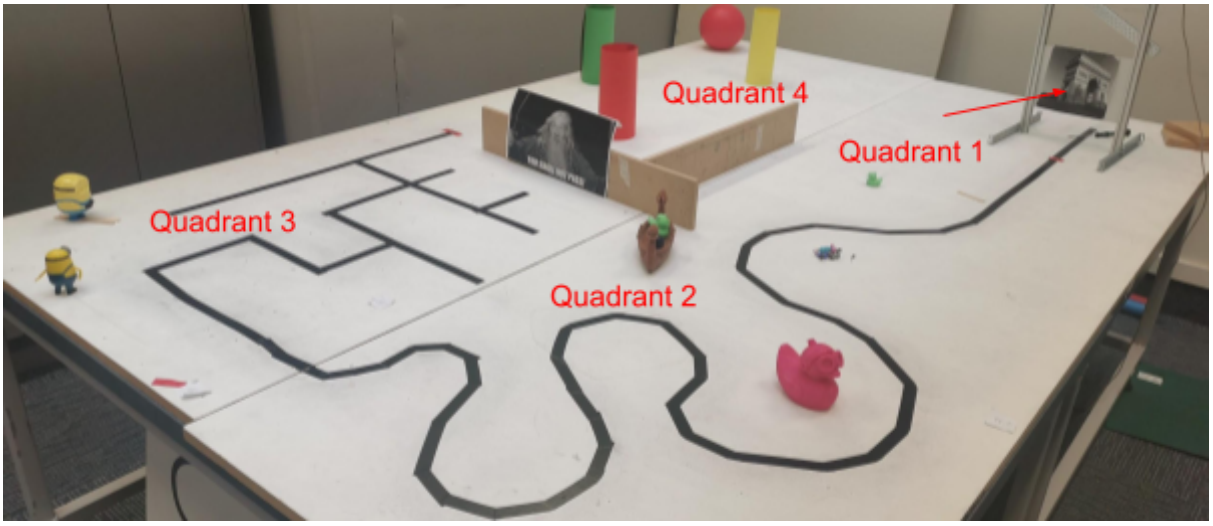


Figure 1. A photo of the AVC challenge track, taken by Brandon Ru

1.2 Motivation

A key motivation behind the AVC project is a better understanding of how to work in a team collaboratively towards a common goal. This is a key skill that we will need in the workforce, and the ENGR101 AVC project epitomizes this.

1.3 Scope

Compulsory Project constraints

4 weeks of lab time, non-negotiable.	Must be worked on only in the CO148 lab.
3D printers are shared amongst the cohort	Programming language is restricted to C++

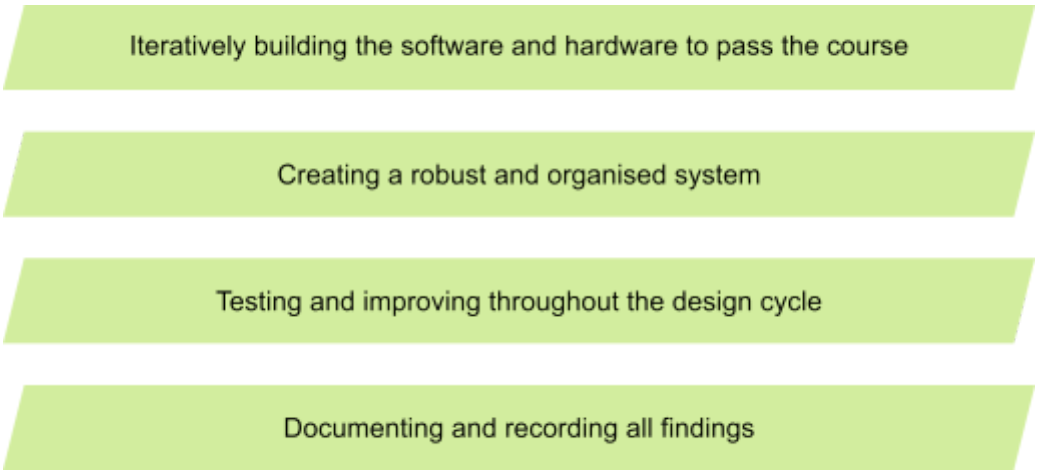
The scope includes:

- No starting code given
- No starting hardware given, apart from a Raspberry Pi.
- Team members are randomised, with differing skill levels.
- Code version control (GitLab) must be used.

The scope excludes:

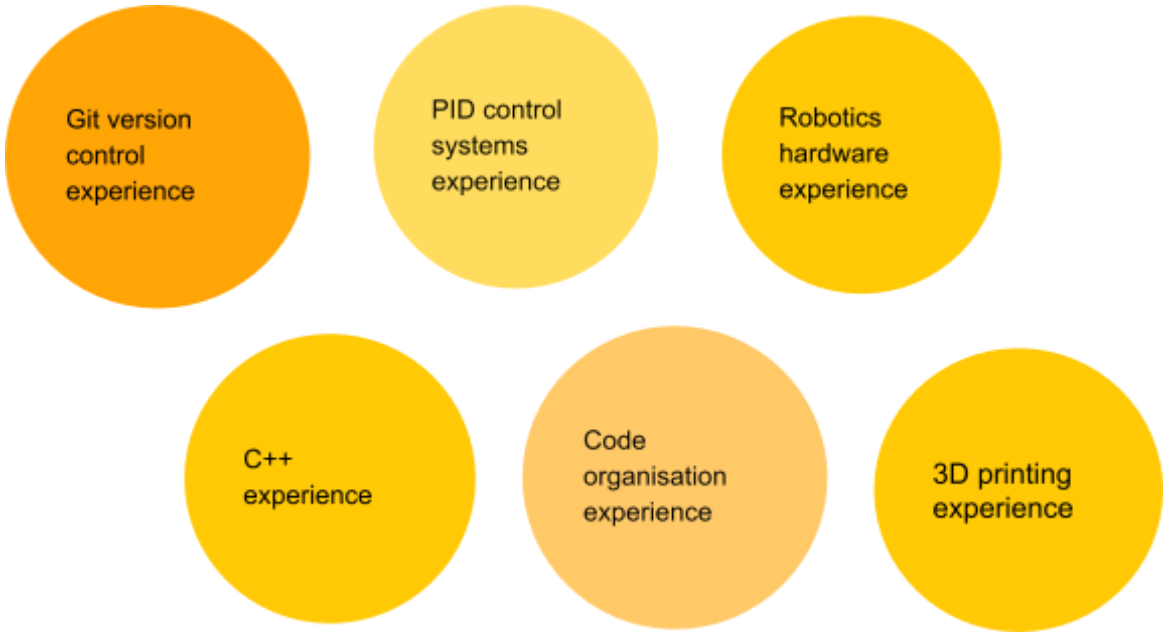
- Initialisation of the Raspberry Pi
- Low level interactions with the camera and motors, provided through the external “E101” library
- Gathering of hardware (screws, bolts, plastics, glues)

1.4 Objectives



1.5 Anticipated Benefits

When the project concludes, we would expect to gain experience with working in a team, more realistic expectations of teamwork, and learn how to effectively work together. More general experience can be summed up in the illustration below.



2 Background

2.1 Hardware

The critical hardware used in this project is

- Raspberry Pi Zero W, BCM2835 SoC
- Battery bank, with a micro USB port.
- Camera
- 3 servo motors: 2 blue and 1 black.

A Raspberry Pi is a limited processing power computer that runs Linux. It is powered by a micro-usb from a battery pack. The blue servo motors allow for a continuous rotational velocity to be set, whereas the black motor allows for a specific constant angle to be set. The camera is a low resolution colour camera, 320x240.

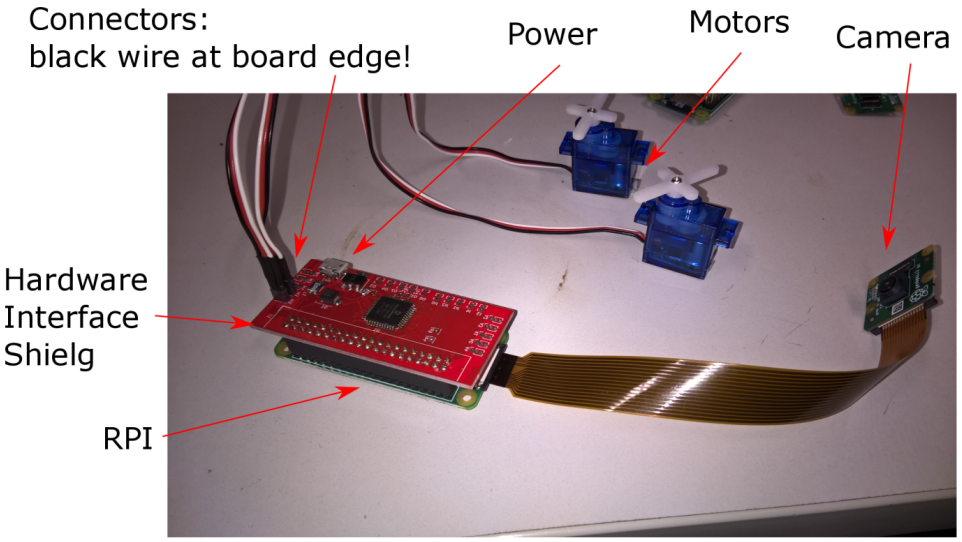


Figure 2. Raspberry PI with other components connected (taken from Lecture 14, Slide 14) [2]

In terms of constructing the chassis of the vehicle, this was primarily done with 3D printed parts constructed with TinkerCAD, and screwed together with nuts, bolts, and screws.

2.2 Software choices

The critical software choices for the project are:

- Use of the C++ language.
- Use of a shell script to build and run the program.
- Use of Git to version control our edits
- Code is remotely edited via SSH and command line editors

C++ is used so we can directly use the E101 library which is required to interact with the camera and motors. A fast compiled language is invaluable on a limited processing machine.

The shell script compiles and runs the program in one command and is easily run by any team member (only needing to remember one command), and is superior to a Makefile (two commands).
Git allows for automatic documentation, and an easy way to sync code between multiple parties working on code, and allows us to revert to the previous version if things go wrong. Editing via command line is significantly faster than via a graphical interface, or online, hence preferred. A graphical overhead is slow, and online editing requires constant Git syncing between the repository and the Pi.

2.3 Project Management

In order to facilitate an efficient and productive team, we also managed an element of project management throughout the 4 weeks. These included:

- A master project plan on GitLab, with the responsibilities of every member.
- A group discord server to provide updates and organise meetings.
- A weekly kan-ban board on Trello, to visualise and break down tasks effectively.
- Regular weekly meetings on Friday, to target what we are doing the next week, and what needs to be improved

3 Method

3.1 Team structure

The respective roles of the team members are as follows:

Brandon Ru	Project Manager, Software Lead Responsible for completing important documentation (such as the plan, and the progress report), organising the team, and designing the software structure / algorithms
Jack Hansen	Hardware Lead Responsible for the design, testing, and improvement of the hardware.
Ethan Windle	Software Developer Responsible for the development and testing of software.
Igor Hulse	Project Organiser, Software Developer Responsible for the development and testing of software, and organising the team.

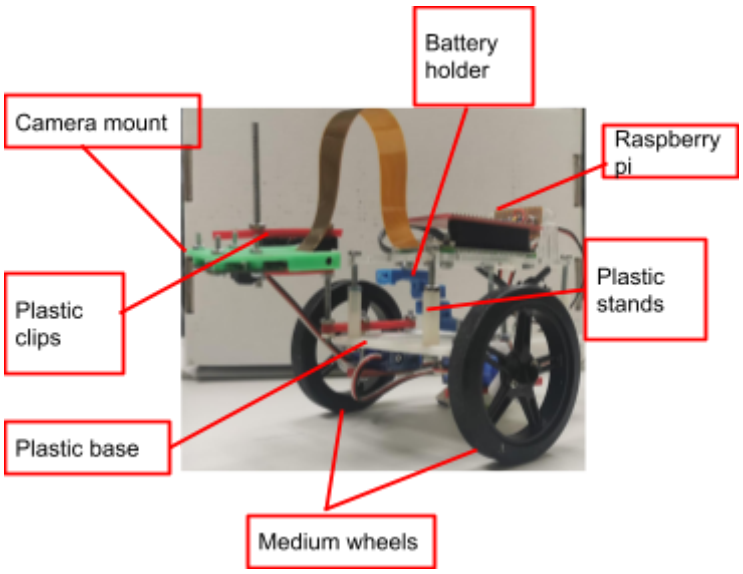
Although this was the initial plan, the roles eventually morphed into one and everyone began to contribute to other areas of the project - helping ease the bottlenecks.

3.2 How the vehicle was built

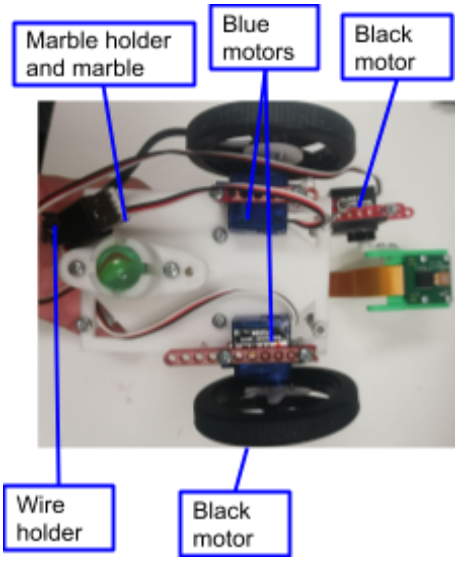
Components list:

- Raspberry Pi
- Camera
- x2 Blue motors
- x1 Black motors
- x2 Medium wheels
- x6 Plastic clips
- x1 Marble
- x1 3D printed marble holder
- x1 3D printed plastic base, same size as Pi
- x4 Plastic stilts
- x1 3D battery holder
- Battery
- USB to micro USB cable

Side on view



Bottom view

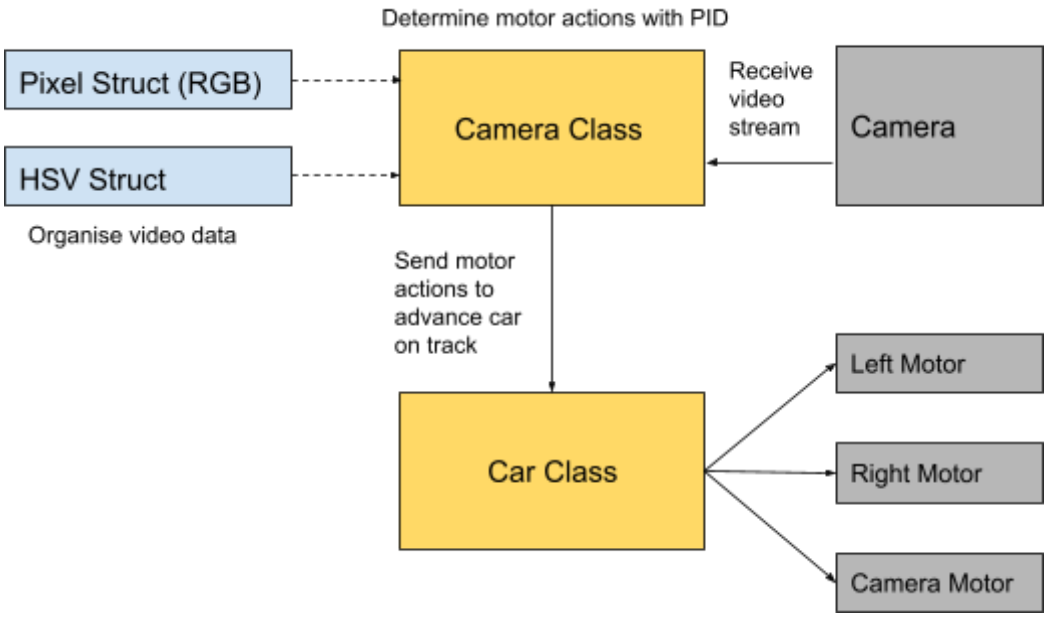


The final chassis design uses a more robust “sandwich” design, with a proper camera mount and reusable design (screws, bolts, and nuts).

Design philosophies

Feature	**Reasoning**
Camera position	The camera is raised so that it can detect more of the track, but is placed close to the wheels so that it does not sway too much when turning. Too much of a sway makes the robot hypersensitive.
Wheels	It has a front wheel drive, with two wheels in the front, while the rear is dragged along with an omni-directional marble at the back (in a 3D printed holder). The front wheels are double tyred, so that there is greater traction for turning. The motors are secured by clips, which are screwed tightly in place.
Battery placement	With a sandwich design, the battery can be slotted in nicely within the middle of the robot. Inside the middle, is also a 3D printed holder for the battery
Camera mount	The camera mount is similar to the motor mount. It is screwed in place tightly with clips and secured to the main chassis
Wires	The wiring of the car is placed underneath the car, so that it is not in the way. It neatly goes through the gaps in the 3D printed base of the vehicle and out the rear.
Weight distribution	The majority of the weight (the battery) lies in the middle of the vehicle, suspended about 5cm off the ground. The camera mount is lightweight, and does not affect the weight distribution much. The weight distribution is mostly even, and relatively close to the ground allowing for more stable turning (less risk of toppling).
Height	The height of the robot must be such that the camera is sufficiently off the ground to observe enough of the ground, but low enough that it does not topple during turns and stops. The optimal height found was 7.5 cm.
Plastic clips use	Plastic clips were used as they were good enough and fast to construct, as we were lagging behind in terms of hardware and were not on track for finishing.

3.3 Software Architecture



There are two helper classes within the program: the Car class and the Camera class. A class is a “package of code that allows for object oriented organising. A struct is an “abstract data type” that organises data together.

The Car class is responsible for directly interacting with the motor servos of the hardware, and maneuvering the robot. The Camera class is responsible for interacting with the video stream, and using algorithms to determine the direction to move the robot in. The Camera class is dependent on Car. The Pixel struct represents an image pixel with RGB attributes. The HSV struct represents an image pixel with HSV attributes.

An abstractly represented software architecture as above allows for cleaner and faster code development. Less time is wasted on code maintenance, and refactoring is kept to a minimum. Furthermore, this arrangement of code is easily unit tested, as classes allow for encapsulation of code.

PID controller [1]

A PID controller will be the main algorithm used throughout the project. This is a simple open loop algorithm that continuously reacts to an external system, and seeks to move the measured variable towards the initial set point, (with the difference known as the error) with an output signal. The proportional constant (KP) dictates how strongly the output signal is based on the error. The derivative constant (KD) controls how “predictive” the derivative component is, and counteracts the proportional component. The integral component deals with accumulative error.

3.5 Quadrant One

IPv4 address of gate	130.195.6.196
Port number	24
Message to send	“Please”

The server is sent an initial phrase of “please”, and replies with the secret password “123456”. This password is then sent back to the server, and the gate will open. To pass through the gate, we move the robot forwards at a medium speed (50% of max) for 2 seconds, passing through the gate.

3.6 Quadrant Two

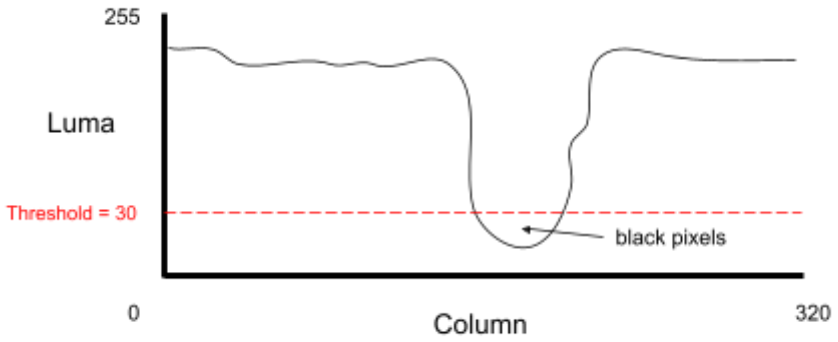
Final key parameters

KP = 1.0/500 KD = 1.0/1700 Maximum luminosity of a black pixel = 30

Quadrant two code will use a PID controller to follow the black line. An error will be calculated each frame, and the system will seek to move the error to 0. The error value is how centered the robot is on the black line.

Calculating the error

The algorithm first takes the luminosity of the central row of the video image, and thresholds it with an intensity of 30. Any value below this is considered black, and set as 1 in the array, otherwise it is set as 0.



Index	0	1	2	...	170	171	172	...	318	319
Value	0	0	0		1	1	1		0	0

Then, the error is given by the sum of each element in the binary array above multiplied by the signed distance from the center. If the error is positive, the line is on the right side. If the error is negative, the line is on the left side. If the error is zero, the line is centered. The magnitude of the error is proportional to how off center the robot is

Index	0	1	2	...	170	171	172	...	318	319
Value	0	0	0		1	1	1		0	0

Index	0	1	2	...	170	171	172	...	318	319
Value	-160	-159	-158		10	11	12		159	160

Error = 0 + 0 + 0 + … + 10 + 11 + 12 + .. + 0 + 0

Finally, the output signal is calculated with the error using the PID algorithm.

$$Output = K_P.e(t) + K_D.\frac{d}{dt}e(t)$$

In a vehicle context, the proportional constant (KP) will control how strongly the vehicle turns based on the error. The derivative constant (KD) will control how steady the vehicle turns, acting as “brakes” for the proportional constant. The derivative of the error is simply the current error subtracted with the last error known.

This output is then passed as a parameter into the robot's turning function, allowing it to center itself accordingly.

3.7 Quadrant Three

Key details

KP = 1.0/370
KD = 1.0/1700
Maximum luminosity of a black pixel = 30

Margin = 20
Right turn threshold = 0.3
Minimum number of black pixels for U Turn = 150
Turn time = 400 milliseconds

Quadrant three code will also implement a PID controller to follow the black line. An error will be calculated each frame, and the system will seek to minimise the error. However, this system will be slightly modified to account for the intersections of the maze.

Pseudocode for traversing the maze

This maze in particular, can be traversed in a very specific manner. Notably:

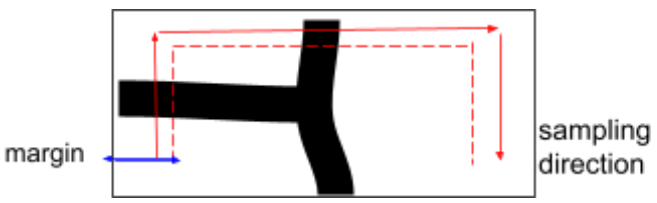
Pseudocode

TURN with the priority order LEFT, STRAIGHT, RIGHT.
IF no turns are possible, make a 180 degree turn

Calculating the error value

Instead of taking the luminosity values of the central row of the image, we take them from the boundary of the image. More specifically, the boundary of the image from the left to the right.

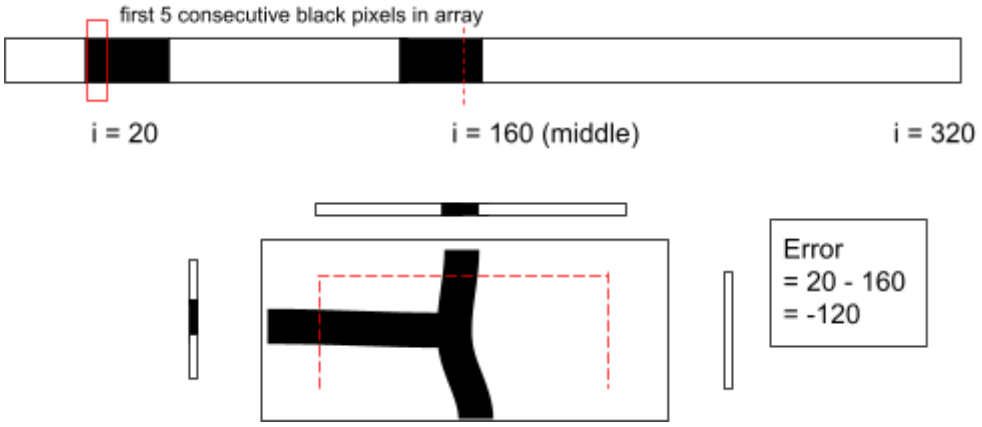
Illustration of luminosity sampling method



The luminosity array is the image sampled on the dotted red line in a clockwise direction. The sampling is done at a margin from the border of the image. Then the luminosity array is converted into a binary array, with the same method as above (thresholding each value against 30, and if it is above it is 1, otherwise 0).

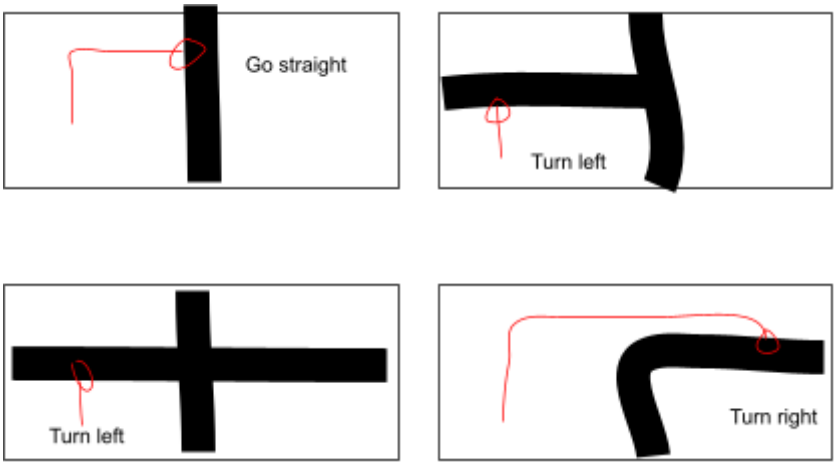
Then, the error of the system is equal to the index of the first element in the first consecutive occurrence of 5 black pixels minus 160 (the central index). Initially, it was the first black pixel but this gave false positives of the line occasionally (due to shadows). Making it the first 5 consecutive occurrences of black pixels reduces the false positives

Illustration of obtaining the error value



Turn logic

The reason this method works for quadrant 3 is because of the key fact it takes the **first** black pixel window from an array that is sampled **clockwise.** Therefore, it will prioritise left turns first, then straight actions, then finally right turns (as these are clockwise from each other).



Making 90 degree turns

A key difference within our algorithm to other teams is the presence of a delay in our turns, if the turn error is greater than a certain threshold. The delay would mean the car turns at a sharp angle for a constant amount of time, meaning it is less likely to lose the track while turning. This was a key addition that allowed us to pass quadrant 3 consistently. 90 degree turns are significantly harder as while completing the turn, our vehicle would lose the track when the camera swings out. The delay mitigates this.

U Turn Mechanisms

A U Turn is performed under the condition there are less than 150 black pixels on screen, and overrides all other turn conditions listed above. A black pixel deficit indicates a dead end. A U Turn on the vehicle is performed by spinning one wheel at max speed in the forwards direction, and spinning the other wheel at max speed in the backwards direction for a fixed amount of time.

3.8 Quadrant Four

Key details

KP = 1.0/500
KD = 1.0/1700

Condition for a red pixel = Red > 2*Green and Red > 2*Blue
Condition for a green pixel = Green > 2*Blue and Green > 2*Red
Condition for a blue pixel= | Hue(pixel) - 235 | < 3

The algorithm for quadrant 4 is very similar to quadrant 2, and yet again, implements a PID controller to navigate the track.

The vehicle repeats this all (red, green, blue) coloured cylinders

Spins around searching for colour of choice

Once found, it moves slowly towards cylinder with PID

Once close enough, it rams it and repeats.

3.7.1 Camera differences

The camera is now raised at a 90 degree angle and no longer faces the track. Instead, it faces forwards and is looking around.

3.7.2 Calculating the error

To calculate the error, the same method as quadrant 2 is used, where the dot product of the binary array is taken with the array of distances from the central column. However, the only difference is that the binary array is constructed differently. Instead of being thresholded with a luminosity value to determine if a pixel is black, it is checked against different conditions for if it is red, green or blue. These are listed above in the key details section.

The output is then calculated with the PID algorithm (with the same parameters as quadrant 2, surprisingly) and the robot now follows based on colour.

3.7.3 Avoiding false positives

To avoid false positives, we have made the vehicle enter a “searching state” to find the next targeted cylinder of choice if the number of coloured pixels it is searching for is below a certain threshold. Once it is above the threshold, it stops. This avoids false positives, and ensures the vehicle does eventually find the cylinder.

3.7.4 Knocking over the cylinders

As the robot moves closer to the cylinder, the number of coloured pixels on screen will increase. By setting a threshold in which it is greater than, and making it accelerate at full speed and then backwards immediately, it allows for it to knock over the cylinder.

3.7.4 Knocking over the red ball

In fact, as our algorithm already knocks over objects it targets, we can easily knock over the red ball by making the vehicle target red again.

4 Results

Initial results (week 1 and 2)

The initial results of the project were both promising and non-promising. The software was written so that it had a very good foundation. With a clean structure and scalable, reusable motor interaction functions that would set us up for success in the future. However, there was a real bottleneck in progress due to hardware. The lack of a prototype led us to be unable to be testing software properly until the 2nd week. Experimental software was written for quadrant 2. The hardware department was lost due to poor communication and design choices. Fortunately, a prototype was created by the second week and this aided in the project's progress.

Development (week 3)

Within this week, a lot more progress on the project was made. The final hardware body was built. The code for quadrant three was written, and tested fully in action. The team was finally on track with where it wanted to be. The camera mount was being built for the final week. Despite the progress however, project management still wasn't perfect and the team still often went off plan and did not use the Trello board provided.

Final week

The final week was where we tackled the final quadrant. The camera mount was quickly completed on Monday (as it was already being created the week before). Furthermore, the quadrant 4 code was very quickly written, finished completely on Tuesday. This is because quadrant 4 code is very similar to quadrant 2 code. In addition to this, our team has figured out a "productive groove", and progress is much faster.

Our vehicle was tested preliminarily by Arthur on the entire track, and despite moving at a slower pace than expected, the vehicle managed to navigate the entire track and we were awarded a grade of 100%.

Quadrant 2 - effects of changing parameters

Changing K_P		Changing K_D	
Value	**Effect**	**Value**	**Effect**
1/200	The robot turns too sharply and even on small bends, it will turn off the track.	1/1400	The vehicle's oscillation is smooth, but it no longer completes tight corners properly, turning too weakly.
1/500 (final)	The robot turns sharp enough to make tighter corners, but not too much so that it loses the track.	1/1700 (final)	The vehicle's motion is smoothed significantly, but still is able to complete the 2nd quadrant.
1/800	The robot turns too weakly, and does not complete the more tighter turns properly, and completely veers off the track	1/2000	No change in oscillation is observed. The robot is still swaying side to side

Changing the black pixel threshold luminosity

Value	**Effect**
10	Occasionally the vehicle cannot detect the line because of glare (which makes the black line brighter), making it go off course.
30 (final)	No shadows are detected.
70	Occasionally, the vehicle will detect shadows and veer off course.

Quadrant 3 - effects of changing parameters

Sampling margin

Value	Effect
5 pixels	The margin is too small. The camera detects the left tyre and consistently returns a very negative error (black pixels on the lower left)
20 pixels (final)	No abnormal behaviour occurs
40 pixels	The margin is too big, the intersections are detected far too late, and the turns are not completed successfully

Right turn threshold

Value	Effect
0.3(final)	The right angle turns are completed consistently
0.5	The right angles are turned correctly, but inconsistently
0.7	The robot does not complete right angle turns, but goes straight forwards.

Right turn time

Value	Effect
100ms	There is no noticeable difference in the vehicle's motion compared to a delay of 0.
400ms(final)	Right hand turns completed.
800ms	Right hand turns are far too overshot and the vehicle fails.

Lower bound of black pixels for a U Turn

Value	Effect
50 pixels	The robot does not even U Turn (even when off the track), due to shadows and stray black pixels.
150 pixels (final)	The robot U Turns at the proper position
200 pixels	The robot U turns early, when making 90 degree turns (as the number of black pixels decreases when this occurs).

Quadrant 4 - Effects of different algorithms for colour detection

Red and green do not have different results as the first attempt algorithms worked correctly. Uppercase colour indicates the RGB component of the pixel.

Algorithm	Effect	Possible reason why
BLUE > 2*RED and BLUE > 2*GREEN	The robot detects the green cylinder and goes towards it.	Debugging messages indicate that the blue cylinder has actually a minority of blue, therefore this method would fail. It has mostly red and green in similar amounts at around 200, and blue at 40.
|RED - GREEN| > 20 and RED > 200 and GREEN > 200 and BLUE < 50	The robot detects dark blue (one of the testers pants), and goes towards them	This is a very wide condition, and most likely would capture more than required. Including the dark blue.
|HUE - 235| = 10	The robot detects the yellow wall when spinning around, and moves towards that. However, when placed in front of the blue ball, it goes towards this too.	Possible reason could be because this is too large of an H range +- 10, and I will move it within closer, to isolate the exact color of the blue cylinder and not any external ones

5 Discussion

Team performance

Despite achieving the highest mark possible in the AVC challenge, the team performance as a whole was sub-par and inefficient. The workload was unevenly distributed, some team members lacked direction and motivation,

There should've been more regular meetings, as it appears that the meeting on Friday was not enough (and often members did not even show up to this one). The planning, although clear, was not utilised effectively.

In hindsight, this was partially due to the fact that as the team progressed, it would've meant many team members were now behind and no longer on the same page and possibly felt they couldn't contribute as much. This could've been solved by creating logging documents and more documentation for other team members to log into, to make sure everyone is up to speed.

Hardware

Again, whilst having a complete robot, our hardware again was relatively unsatisfactory. The parts were relatively inaccurately constructed together, and the design was non-sturdy. The motors would sometimes shift in position, the camera mount would tilt, or the battery would fall out. This is most likely due to excessive use of screws and nuts to secure the motors and camera, compared to 3D printing exact parts like some other teams have done, which provides more accuracy.

This would've been solved by focussing on the hardware as a team very early on, and getting it fully complete first, before focussing on software. The initial split focus most likely had strained the hardware lead, and caused a rush in development. Furthermore, an earlier approach to the hardware in the timeline would've meant greater access to 3D printers (as these are shared), and a more elegant design.

However, despite these difficulties the hardware was indeed viable, and allowed us to pass the track.

Software

The software of the vehicle on the other hand, was the strongest performing part of the vehicle. Part of this came down to the architecture of the codebase, which allowed for fast and clean development that was easily unit tested. However, there were still improvements that could be made.

The quadrant 3 code of the program was inconsistent and depended on far too many parameters. This makes the program less robust and reactive to changes in the track and the hardware. We could've fixed this by dedicating a greater design team to this quadrant, as at the time, the majority of the team was working on hardware.

The quadrant 2 code of the program is not perfectly smooth. This was ultimately not identified why, as we dedicated more time to quadrants 3 and 4. That is alright though, as it means that we can work on the future. It is suspected that it is due to lack of parameter accuracy, but we do not know.

The quadrant 4 code of detecting blue is very poor. This is a hardware issue. RGB testing of the BLUE cylinder was very shocking, as it revealed that the "blue" cylinder is actually yellow. This led us to use HSV , which allows us to isolate the H component of the colour, and solve this issue. However, this is still rather inconsistent. This is really only fixable by obtaining a higher quality camera.

6 Conclusion

The project output was very successful, and we managed to reach our goal of completing quadrant 4 eventually, with a few bottlenecks along the way. The team eventually began to work somewhat cohesively together, and began to understand the value of teamwork.

The collaboration, however, was not without fault. Lack of direction and motivation was a large problem in the team, but team members have been able to identify faults and work on them in the future.

In terms of hardware, the team learnt much about 3D printing and how a vehicle should be arranged. In terms of software, the team learnt much about PID controllers, code organisation, and computer vision.

While improvements could've been made, I would say the project was successful nonetheless.

7 Acknowledgement

The team would like to thank Arthur Roberts, Howard Lukefahr, and all of the engineering tutors for providing this opportunity to us, and aid in working on the project. Arthur, in particular for allowing some team members to stay especially late working on the robot, and we thank him for his time sacrifice.

8 References

[1] Peacock, F. (n.d.) *Idiots guide to PID Algorithms: PID for Dummies.* Control solutions Minnesota. Retrieved June 4, 2021 from https://www.csimn.com/CSI_pages/PIDforDummies.html

[2] Roberts, A. (2021, April 21) *ENGR101: Lecture 14 - Intro to AVC.* Victoria University ECS. Retrieved June 4, 2021 from https://ecs.wgtn.ac.nz/foswiki/pub/Courses/ENGR101_2021T1/LectureSchedule/ENGR101_Lecture14.pdf

[3] Roberts, A. (2021, April 23) *ENGR101: Lecture 15 - AVC: Image processing and movement control.* Victoria University ECS. Retrieved June 4, 2021 from https://ecs.wgtn.ac.nz/foswiki/pub/Courses/ENGR101_2021T1/LectureSchedule/ENGR101_Lecture15.pdf

[4] HSV colour in computer vision. (2013). Stack Overflow. https://dsp.stackexchange.com/questions/2687/why-do-we-use-the-hsv-colour-space-so-often-in-vision-and-image-processing

[5] Brian Douglas. (2012, December 14). PID Control - A brief introduction [Video]. YouTube. https://www.youtube.com/watch?v=UR0hOmjaHp0

If you can read this, I hope you have a nice day, and things go well in life sir/madam :)