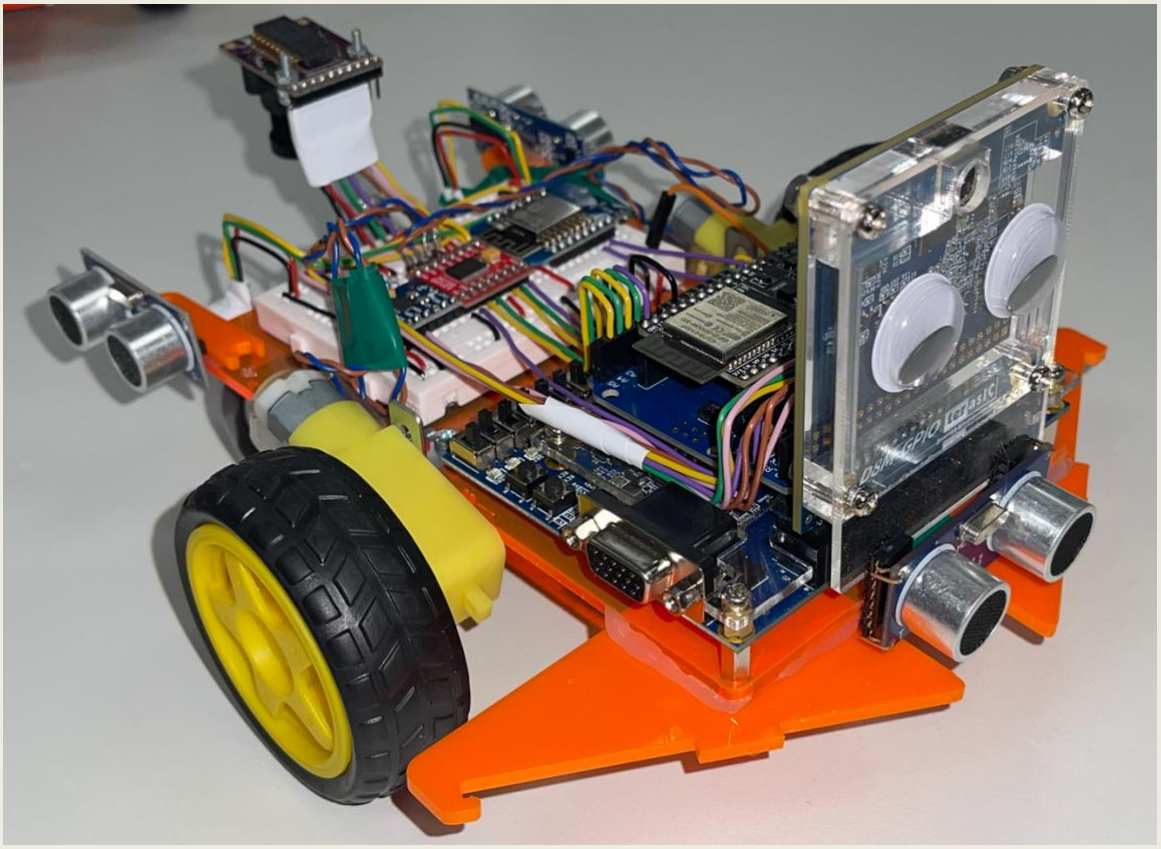
Word Count: 6668

Imperial College London

2nd Year Summer Project – ELEC50008

MARS ROVER



Group OSA2

Arthika Sivathasan (01921568)
Dilan Jayasena (01901114)
Savraj Sian (01847921)
Gian-Luca Fenocchi (01860614)
Kristina Filichenok (01902215)
Ifte Chowdury (01733579)
Aishwarya Anand (01852814)

1 TABLE OF CONTENTS

2 ABSTRACT

The project intends to create a rover that can map the position of six uniquely coloured aliens, striped buildings, and alien infrastructure, on a map visible to the user via a web app. The project incorporates both software and hardware integration. The drive, vision and radar modules use hardware to collect data that is then processed by the control and command module before being shown visually on the web app. To successfully actuate the rover, key design considerations had to be taken based on knowledge acquired from developing each sub-module. Throughout the timeline of the project, problems, limitations, and efficiency of the implementation were discovered and some resolution was made, all of which will be explored within this report.

3 INTRODUCTION

The aim of the project was to design and build a rover that can be either controlled manually, using a website or autonomously, through a self-guiding system (in some ways replicating one of the actual Mars rovers). The rover's task is to navigate its way through an arena and build a map of the different objects it encounters. The rover was to be solar-powered, utilising the solar-panels provided. The project was split into six different key subsystems, five of which were required to make the rover move and detect aliens, buildings, and fans on the arena. The Power Module was to be a separate substation used to charge a battery for the rover.

4 HIGH-LEVEL DESIGN

As a team, we had to design and implement six subsystems to control the rover. These subsystems had to effectively communicate with the central controller other to transmit data collected from the rover to the user and control the rover via inputs given by the user.

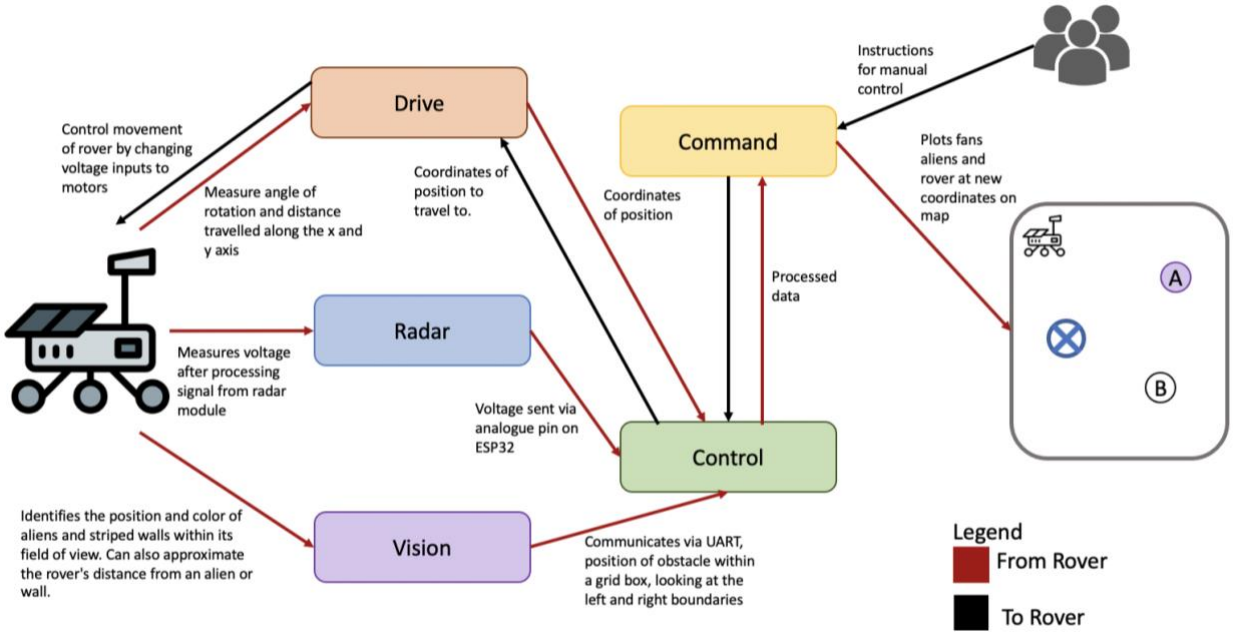


Figure 1: High-level design of rover and interaction of 6 subsystems

5 GROUP WORK AND PROJECT MANAGEMENT

5.1 GANTT CHART AND OVERVIEW OF KEY DEADLINES

The project was managed through regular meetings and the progress was tracked on a Gantt chart (See Appendix A). Table 1, below, shows the main milestones set by the team for the project and their delivery dates. A meeting was held before each milestone to discuss the work and review the deadlines and deliverables

The initial phase of the project focused on the design and development of the 6 subsystems (5 of which would be on-board the rover, with the sixth being a remote charging station). Each subsystem was allocated its own task in the project, with a team-member in charge. The second phase of the project dealt with the seventh task: the integration of the 6 key modules

Table 1: Project Milestones

Milestone	Deadline
Project Start	23/05/2022
Midway Point (Start Integration)	01/06/2022
Internal Report Deadline	20/06/2022
Final Report Deadline	22/06/2022
Internal Project Deadline	27/06/2022
Final Project Deadline	29/06/2022

5.2 Module (task) Allocation

The responsibility for each module was allocated based on the initial assessment of team members strengths and experiences. However, throughout the project, a lot of teamwork was established, with team members often collaborating to resolve issues and particularly during the integration process. Table 2 shows the initial allocation of tasks (modules) for the various subsystems

Table 2: Task (Module) Allocations

Subsystem	Team Member
Control	Arthika
Drive	Dilan
Command	Client - Luca, Server - Savraj
Vision	Ifte
Radar	Kristina
Power	Aishwarya
Integration	All

Since there were two members allocated to Command, this module was completed relatively quickly. Following this, Luca and Savraj moved on to aid the team in the integration of the various modules.

5.3 Meetings

Within each meeting, a set structure was followed to ensure that all key details were discussed, and issues addressed. A meeting structure was developed (see Appendix B), which worked well initially, but which necessarily changed somewhat during the integration phase.

6 Control Module

6.1 Overview:

The Control Module is responsible for receiving and processing data from all other submodules, using the ESP32. It redistributes this data to other submodules for use. The Control Module must communicate with a server (Command module) wirelessly and therefore, needs to establish a connection to Wi-Fi. A database is established to store any data, which the server can then query.

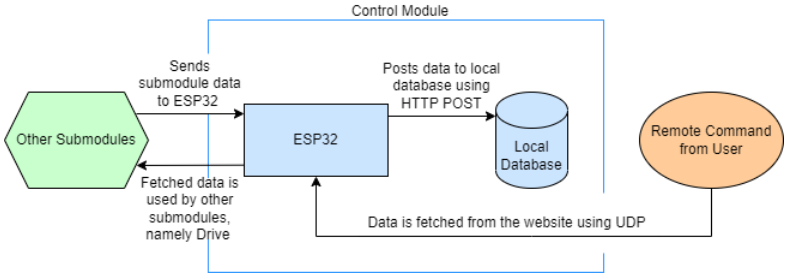


Figure 2: ESP32 interaction with other submodules

6.2 INITIAL DESIGN DECISIONS AND IMPLEMENTATION:

To connect the ESP32 to a Wi-Fi signal, the inbuilt *WiFi.h* C library was used. . This provided the appropriate functions that allowed utilisation of the ESP32's Wi-Fi capabilities, after providing the suitable Wi-Fi login details. It is important to note that the ESP32 has three different Wi-Fi modes [1]:

- **Station Mode**: In this mode, the ESP32 can connect to other networks and the router assigns an IP address to the ESP32. The ESP32 can talk to other devices connected to the same network.
- **Access Mode**: ESP32 acts as an access point and stations connect to the ESP32.
- Both Access mode and Station mode.

For this project, the ESP32 was set in Station Mode. The database implemented in the Control Module used MySQL because the team was most comfortable with MySQL syntax in contrast to other languages.

The Control module initially used a TCP connection to send and receive data to Command. This was because TCP is connection-oriented, and as a result more secure and reliable. Furthermore, TCP allowed for a database in which we could queue instructions during manual control of the rover, as TCP ensures in-order arrival of datagrams. However, after integrating it became apparent there were issues with delays and switched to UDP to receive commands during manual control, so that data could be fed instantaneously to the Drive module. However, any data needed by the Command module was stored in a database first and so still utilised HTTP and TCP.

The database was hosted locally as opposed to remotely on an Amazon Web Server. The intention was to change this at a later date to be remote. Getting operations working locally was important to initial testing since moving to a remote server would introduce another factor of latency. However, using a remote server would have been beneficial as it would have allowed access to the data from anywhere, at any time. In the final design, a hybrid solution is used where the Control module hosts and processes data on a local server and the Command module uses a remote database. This was to prevent any potential network bottlenecks or delays.

To send data to the local database, HTTP protocols were used. They work as a request-response procedure between a client and a server, meaning that data could be requested only when needed. The other advantage of using HTTP is that it returns response codes, after each interaction with the database. This was useful for troubleshooting as it was easy to see whether each request had been successful or had failed. For example, a HTTP response code of 200, would indicate 'no-fault', whereas a HTTP response code in the 500s would indicate an error within the server. HTTP POST was used to post data to the local database. A future consideration would be to use HTTPS, for added security. Along with the main code, PHP scripts were used to interact with the functionality of the server. These scripts contained commands to connect to the server and SQL queries.

6.3 PROBLEMS AND TROUBLESHOOTING:

Problems with the HTTP protocol often arose; however, they were often easy to debug since the HTTP response codes helped identify the problem at hand. Similarly, there were problems with ensuring things were working on the server as expected. However, the database in XAMPP was visible in PhpMyAdmin to see if entries were updated as expected. This was especially useful when testing the ESP32's wireless capabilities as using the Arduino's serial monitor was not possible, since this

required a wired connection, however evidence of what was happening to the database through the PHP scripts was observable.

7 DRIVE MODULE

7.1 HARDWARE INTEGRATION

The drive module hardware comprises the ESP32, motor controller TB6612FNG, and the Optical Flow Sensor (OFS) ADNS3080. The ESP32 transmits and receives serial data from the database that is transferred to and from the control inputs and outputs of the motor controller and the optical flow sensor. The TB6612FNG is a dual-channel H bridge motor controller which operates the DC drive motors, through changing the polarity for directional control and PWM control of the applied voltage for speed control. Finally, the ADNS3080 is a device that uses motion estimation which involves analysing moving objects where the optical flow sensor takes pictures at a given frequency and analyses the correlation between the images in order to identify a change in position.

7.2 PROPORTIONAL CONTROLLER

A proportional controller was utilised to enable the rover to drive in a straight line. The controller uses a negative feedback loop when the OFS detects a change in the x value, as the rover travels along x=0. During the correction process, each wheel adjusts its angular velocity to correct the position of the rover to keep it on the desired trajectory. The power of the wheel closest to the imaginary straight line was calculated by multiplying a negative square of the error by a constant gain of ten, whilst the wheel furthest from the line will have a positive square of the error multiplied by the gain. This results in the closest wheel decreasing in velocity and the furthest wheel increasing in velocity. To quantify the error, the relationship between the speed of the motor against distance off course was mapped. A square relationship was used as it allows the rover to react swiftly to larger displacements off the valid route, while small displacements require a smaller change in velocity between the wheels. This approach was proven effective when put into practice with the rover being able to visibly amend its vector and stay on track.

7.3 PHYSICAL UPGRADES

To enhance stability and traction during rotation movements, a ball bearing was connected to the rover. This substituted the nylon end-nut, which was observed as generating excessive friction, resulting in a loss of mobility. When observing rotation with the implemented ball bearing, there was a clear improvement in the smoothness of rotation due to the lower friction.

7.4 HARDWARE UPGRADES

When calibrating and operating the ADNS3080, it was evident the OFS had limitations where occasionally it struggled to measure accurate changes of displacement in motion. This compromised the reliability and the accuracy of the rover to travel in a straight line, rotate at fixed angles and record accurate distances for the coordinate system. During practice, *Blu Tack* was affixed to the threading of the lens to prevent the focal length from altering from its optimal position. After research on improving OFS performance and accuracy, it was recommended to append a focused, high-intensity light source. A bright focused LED with a low viewing angle was therefore installed. As a result, a clear improvement in precision was observed when measuring distance travelled and displacement off track.

However, all the enforced improvements were insufficient as the impact of distance and rotations of the rover presented accumulating uncertainty and unreliability. In response, a solution was produced

in the form of a gyroscope and accelerometer (MPU 6050) [2]. This device provided more accurate measurements of the angle and helped accelerate the rover on the XY plane. The device was enforced in all manoeuvring programs as an input. Its function was to correct the OFS when required. When effectuated, the rover during the exercise demonstrated more precise rotations, and enhanced capability to execute pinpoint movements. Therefore, the device helped quantify the power necessary to rectify the orientation of the rover to maintain a straight course. See Appendix C for a flow chart that summarises the complete module in detail.

8 COMMAND

The command module is split into two sections: client and server.

8.1 CLIENT

The client-side of the Command module has the job of visualising data sent by the rover through the backend *NodeJS* server. The web app aims to display a map that illustrates the position of the rover, aliens, obstacles, and fan. This map will have to be updated regularly to display accurate positions. Furthermore, where manual control of the rover is required, a message must be sent to the server, containing manual control data.

The client was implemented using *ReactJS* (See Appendix D.1), being robust and modular. This allowed the application to be accessible from most devices (with a modern browser). The client/server split allows the front-end to send an HTTP GET request, which returns the data and position of all entities. Using the recently retrieved data, a map was drawn using HTML's canvas function. Each entity was drawn as a separate shape and had a unique colour.

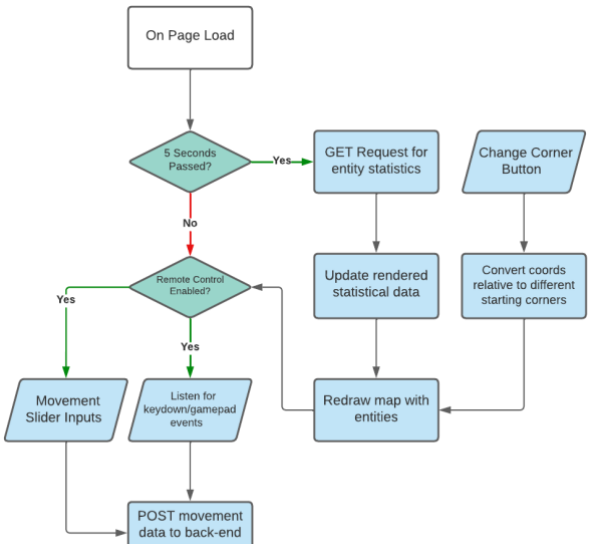


Figure 3: Client flow diagram

To manually control the rover, the user would have to click an enable button, unlocking the controls to move the rover. This button enables the switching of the operating mode between autonomous driving and manual control. The web application was designed to allow up to 3 methods of control input. The user could choose between using their keyboard keys, a wireless/wired gamepad controller or by manually selecting the distance, angle, and power through a slider form. The message sent, containing the movement data, consists of three values: distance, angle, and power. These three variables allow precise control of the rover.

8.2 SERVER

The role of the server is to retrieve data from the MySQL database used by the rover, store relevant information in a *DynamoDB* database and send that data back to the client. It is also able to send remote control commands to the rover. The decision to store data offsite was taken, as in the scenario that the rover was to go offline or be interrupted, data could still be retrieved, and since *DynamoDB* is a managed service, security is ensured, and regular backups are possible.

The server was implemented using *Node.js* since it interacts well with *React*, is flexible, and has many libraries to facilitate making it. The client's HTTP GET requests are to various URL paths hosted by the server to get information, for example, */getaliens* to get colour and coordinate information on all found aliens. These were made using the *Express* middleware since it enables quicker and easier development. When a request is made, the MySQL database is queried for the relevant information using the *mysql-await* library and the *DynamoDB* database is then queried for the same information using the *aws-sdk* library. The newer *DocumentClient* version of the *DynamoDB* API was used as native JavaScript types and objects can be used, resulting in a more natural integration with NodeJS. Both queries are made asynchronously as the information is not guaranteed to be returned immediately, so the server should wait. This is achieved using the *await* function and *promises*, which act as placeholders for values yet to become available. Once the responses arrive, their items are pushed into respective arrays, which are compared using a function in the *lodash* library, and any objects that appear in the MySQL database, but not in the *DynamoDB* one, are added to it and this newest version is pushed to the client for use. Without the asynchronous queries, it would be likely that the comparison fails as at least one of the arrays would probably contain 'undefined' data since a response would not have been received in time. The flowchart portrays the actions of the server based on the two possible input triggers it can receive.

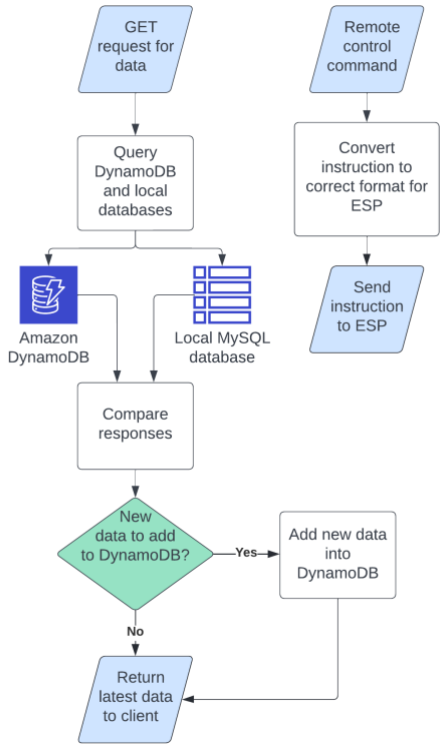


Figure 4: Server flow diagram

All of this happens in the background from the client's point of view, and it receives the data needed quickly; an average time delay (excluding outliers) between request and response of 104 ms was measured (See Appendix D.2), which is significantly faster than the five-second gap between requests from the client. The decision was taken to only have the most recent rover information in the *DynamoDB* database to avoid it expanding too large, so when new rover data is requested, the old data is deleted when the new data is added. However, this deletion only occurs once the new data arrives from the MySQL database, to avoid erasing the only available data should the rover become unreachable. The remote control works by sending UDP datagrams directly to the ESP32, which contain movement instructions sent to the server by the client through another endpoint. This is achieved using *Node.js'* built-in UDP socket functionality.

9 VISION

The vision module is designed to identify aliens and alien buildings within the camera's field of view and communicate the objects' coordinates to the ESP32. This is achieved with a modular design and the EEE_IMGPROC module is key to accomplishing the alien detection.

9.1 IMPLEMENTATION

The camera sends the image in its view as a Bayer pattern which is then interpolated to RGB values. Initially, the weighted average of a 5x5 square around each pixel was used to blur the image and

remove erroneous distortion based on Gaussian blurring, however, this approach required a large buffer array that utilised too much RAM. This was solved with a 1-dimensional, 5-element weighted average since it only requires 4 pixels to be buffered and succeeds in reducing error since the rover only needs the variation in horizontal RGB values in order to rotate to an appropriate angle. The weightings use discrete approximations of Gaussian blurring [3] so that the FPGA isn't slowed by too many calculations, although this does reduce the effectiveness of the blur. The final averaged pixel value is normalised by right shifting instead of dividing as it takes fewer cycles.

Once the blurred image RGB values are saved, they are converted to HSV (hue, saturation, value) in order to make colour masking easier since it removes the need to implement inequalities with multiple variables and is easier to understand the colours from a human perspective. The HSV conversion is scaled up with a coefficient of 170 so that the FPGA can do fixed-point arithmetic without truncating essential data, and the resultant values vary from 0 to 1019. Ternary operations and the modulo operator are used instead of trigonometric functions to get faster and more accurate results. A MATLAB tool called 'Colour Thresholder' is used to find the ideal threshold HSV values for masking each colour of the aliens by manually adjusting the hue, saturation, and value until only the alien is visible. This step is crucial because the ambient lighting has a significant impact on the accuracy of the colour detection, so calibration of the mask thresholds is only done in labs where lighting is less likely to vary. Detected alien pixels are highlighted with their corresponding colours whilst all other pixels are set to greyscale. In order to further remove random noise, a pixel is not highlighted unless a pixel to the left and right of it also meets the HSV thresholds. The minimum and maximum coordinates of these highlighted pixels for each colour are used to create bounding boxes which are latched and drawn on the frame in the following cycle. Alien buildings are detected in a similar way, except the FPGA checks for at least 3 black pixels followed by 3 white pixels, or vice versa. It was decided that precise edge detection of buildings was not necessary since as long as the direction was determined, the ultrasound sensors would work to avoid any collisions.

In order to communicate the alien locations to the control module, the FPGA loops through an FSM (Finite State Machine) and sends bounding box coordinates to the FIFO (First In First Out) message buffer. From this, they will be transmitted along the Arduino receiving pin D9, which is connected to UART port 2 on the ESP32. The message buffer is necessary as it allows the FPGA to queue all the data before sending it as a serial transmission, reducing the frequency of CPU interruptions.

9.2 TESTING

The image processing code is tested by compiling it on Imperial's remote desktops, taking at least 5 minutes, and then flashing the FPGA with the .sof file locally. If any erroneous highlighted pixels exist not in the proximity of the aliens, the bounding boxes become unusually large and the coordinates output to the ESP32 are inaccurate as a result. In most cases, this is due to the ineffectiveness of the Gaussian blurring or loose thresholds on the colour filter.

10 RADAR

10.1 PURPOSE OF THE MODULE:

The HB100 Doppler radar is used to detect a fan in the arena, while the rover is moving.

10.2 Testing the Radar with the Fan:

The signal at the IF port of the radar when the fan is under the arena and the radar is mounted underneath the rover located over the fan is a 14mV p-p triangular wave.

10.3 Design and Implementation Process:

The radar was mounted underneath the rover to minimize its distance from the fan to be detected. The IF port picks up the signal reflected from the fan (AC triangular wave of 366Hz frequency), and the surrounding clutter (DC offset of 210mV plus noise). On its own, the radar outputs a very weak signal, so for sufficient analysis of the data, the signal is amplified. A 2-stage bandpass filter with a 250-450 Hz passband is then used. It is easier to analyze DC values when using ESP32 for the purpose of locating the fan. After filtering, the signal is rectified and peak-detected to obtain a DC output, which is proportional to the amplitude of the received signal at the IF terminal of the radar. See Appendix E for the circuit diagram for the reference voltage.

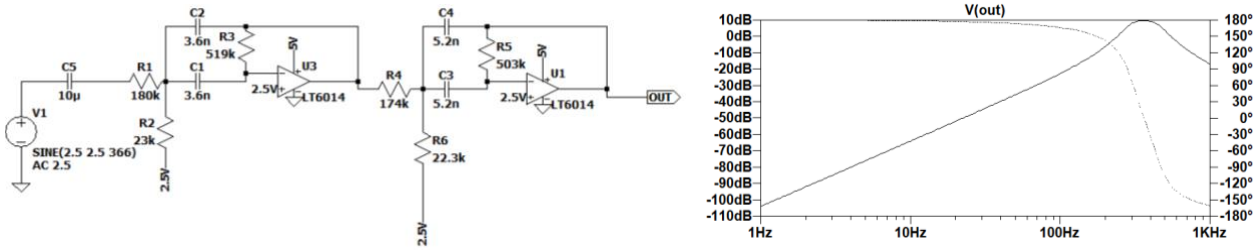


Figure 5: (Left) Bandpass Filter Diagram, (Right) Bode Plot graph

10.4 Filter Circuit Testing

It is seen that the gain of the filter at the passband differs from the simulation by 10db. This is due to op-amps in the simulation being different to the ones finally implemented (MCP6002), as well as the values of accompanying resistors and capacitors being a little different. This does not pose a big issue, since this can be compensated by the amplifying stage. The important result is that the passband frequencies range matches quite well.

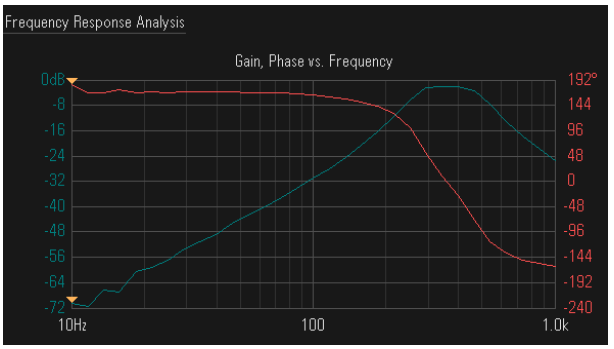


Figure 6: Frequency Response Analysis of Filter

10.5 LTSPICE Simulation

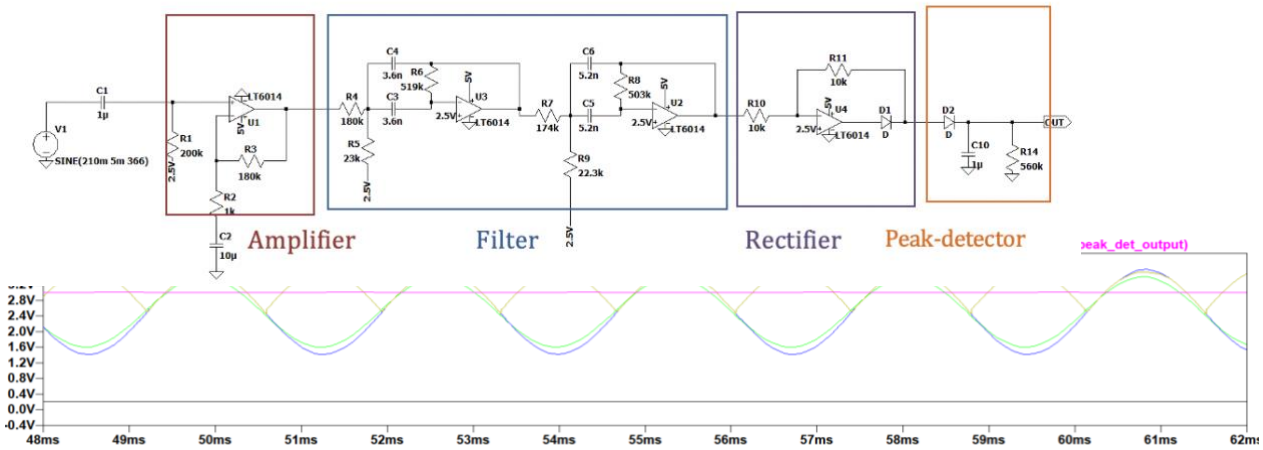
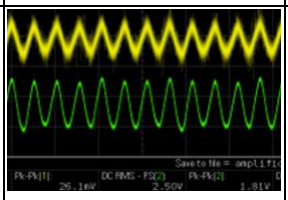
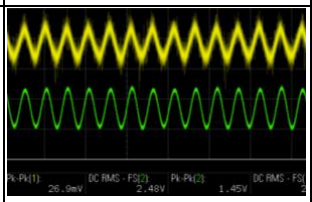
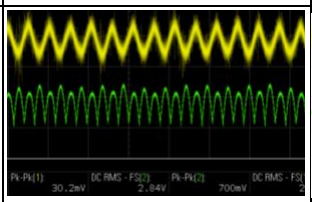
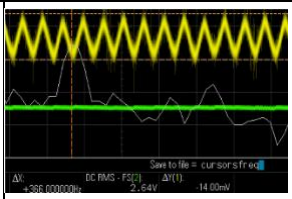
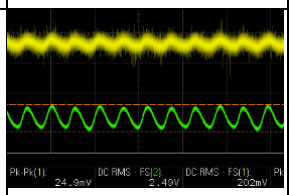
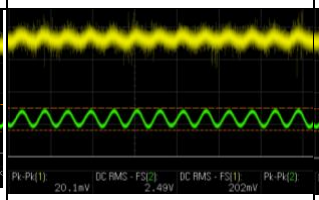
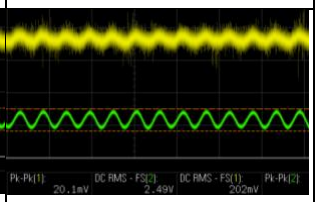
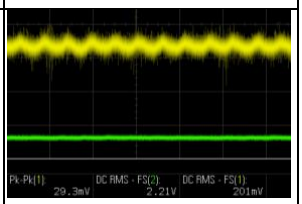


Figure 7: The diagram above shows the expected behaviour at each of the nodes. The output DC voltage level changes from 2V when there is no signal in, to 3V when the signal is maximum(7mV).

10.6 Testing Full Circuit With The Radar And The Fan

In the table below the yellow waveform is the wave coming from the IF terminal of the radar. Green is the output wave of the corresponding block.

Block	Amplify	Filter	Rectify	Peak-Detect
Radar Over the Fan				
	IF Terminal(mV): 200(DC RMS), 26.1 (Peak to Peak) Output(V): 2.5 (DC RMS), 1.81(Peak to Peak)	IF Terminal(mV): 200(DC RMS), 26.9 (Peak to Peak) Output(V): 2.48 (DC RMS), 1.45(Peak to Peak)	IF Terminal(mV): 200(DC RMS), 30.2 (Peak to Peak) Output(V): 2.5 (DC RMS), 1.81(Peak to Peak)	IF Terminal ΔY (mV): 14 Output(V): 2.64 (DC RMS)
Radar for 5cm Away				
	IF Terminal(mV): 202(DC RMS), 24.9 (Peak to Peak) Output: 2.49V (DC RMS), 460mV (Peak to Peak)	IF Terminal(mV): 202(DC RMS), 20.1 (Peak to Peak) Output: 2.49V (DC RMS), 340mV (Peak to Peak)	IF Terminal(mV): 201(DC RMS), 23.7 (Peak to Peak) Output: 2.54V (DC RMS), 200mV (Peak to Peak)	IF Terminal(mV): 201(DC RMS), 29.3 (Peak to Peak) Output: 2.21V (DC RMS)

Distance Away	>7cm (fan out of scope)	0cm – 5cm
Output	2.15V	2.70V – 2.20V

Figure 8: The results from the oscilloscope for each of the blocks match considerably close with the simulation. The lower table shows varying output voltages for varying distances.

11 POWER

11.1 AIM OF MODULE

The aim was to create a power station that will charge a 10W-rated battery pack efficiently.

11.2 CHARACTERISING PV PANELS AND DECISIONS TAKEN ABOUT ARCHITECTURE

The PV panels were initially characterised under lab lighting conditions before proceeding to take measurements under direct sunlight. An arrangement that provided a high current was desired so that the switch-mode power supply (SMPS) would not draw more current than the panels can provide even in poor lighting conditions, as this would drop the input voltage drastically. After testing indoors, an agreed arrangement that fulfilled the requirements was 4 panels in parallel which would give a maximum power point (MPP) at around 4.1V and input currents in the range of 50-60mA. Outdoors on a sunny day, it is expected the input currents would be far higher, around the range of 700-800mA for an input voltage close to 5V. The solar panels provide a maximum of 5W altogether, which is enough to charge the battery as shown below.

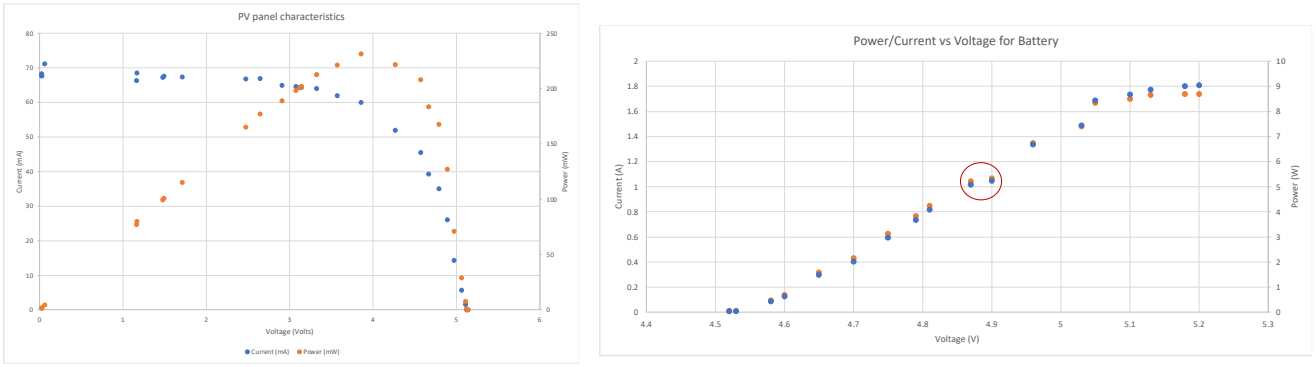


Figure 9: (Left) IV and PV characteristics of 4 solar panels in parallel, (Right) Characterising battery while charging.

11.3 OVERALL ARCHITECTURE OF MODULE

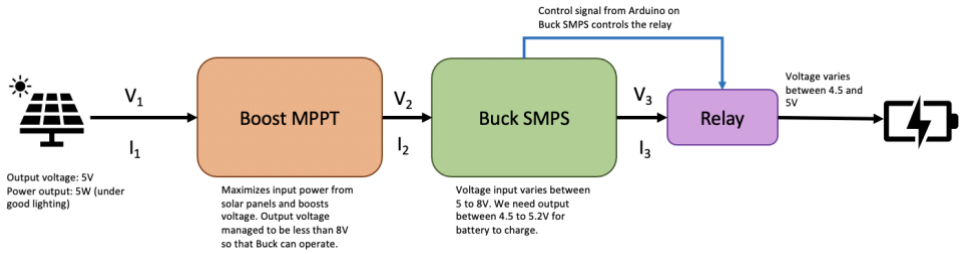


Figure 10: Topology of charging substation

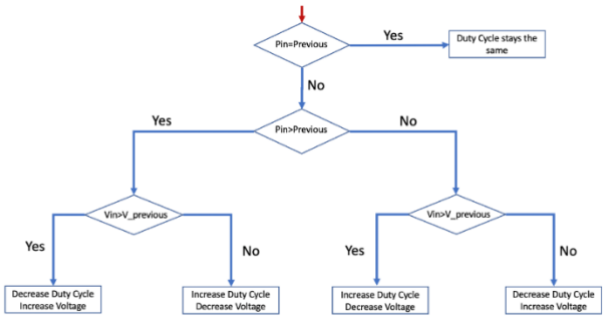
11.4 DESIGN CONSIDERATIONS – MPPT

The maximum power point tracking (MPPT) of 4 panels in parallel gives input voltage at around 4.1V. The battery needs voltages in the range of 4.5V to 5.2V to be able to charge (refer to Fig 9. right).

The voltage in the MPPT must be boosted and then bucked to the desired value. Also, from research [4], the efficiency of the boost converter for the MPPT is far higher than the buck converter for most duty cycles. After the boost, a buck voltage regulator regulates the output voltage to keep it between

4.5 and 5.2V, ideally around 5V. An approach to do this in an iterative loop was taken since it is approximately 1000 times faster than the MPPT to guarantee that the output voltage will always be suitable for the battery (frequency separation approach).

There are two main algorithms used in the function of the Boost and the Buck. The MPPT algorithm tries to find the power point by increasing or decreasing the voltage to move right or left, respectively, along the PV curve.



The duty cycle increments are dependent on the absolute value of the error between the current power and the previous power reading. When this error value falls below a certain value, the duty cycle increments are decreased such that the next power and voltage values will be closer to the maximum power point. Changing the increments of the duty cycle takes into consideration the error, preventing the system from oscillating around the MPP and helps the overall tracking capability of the system.

11.5 Testing Of MPPT Algorithm Independently

The Boost MPPT was tested both indoors and outdoors with the 4-parallel arrangement. Due to the varying sunlight level, it was more difficult to justify whether the MPPT was working outside since the powers were prone to change slightly. These measurements were taken with a constant load of 10 Ω at the output of the MPPT. It is important that the duty cycle of the boost is saturated at 10% otherwise the algorithm takes a long time to reach the MPP, and at very low duty cycles, the current is very low, so efficiency is poor (see Appendix F). At very low duty cycles the boost can be modelled as an inductor in series with a diode which gives a 0.7V voltage drop and bucks the voltage.

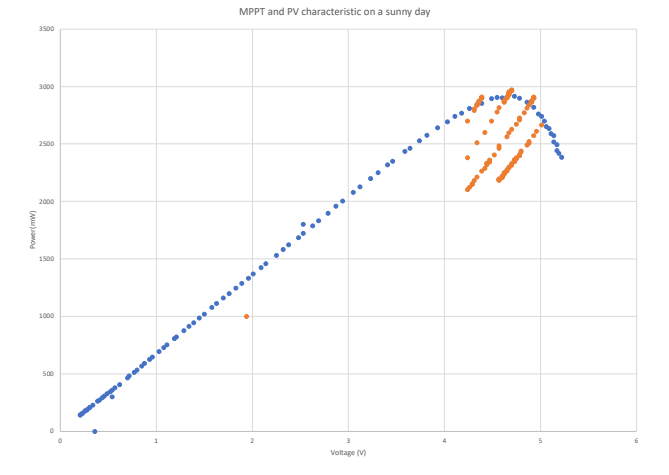


Figure 12:MPPT on a sunny day

11.6 Design Considerations – Buck Output Regulator

To balance the output voltage, the duty cycle is changed accordingly. Once the output voltage is fixed at the desired 5V, a signal is sent to the relay to close the circuit and charge the battery. A normally open (NO) configuration was selected, as it is crucial to carefully regulate the voltage input to the battery before we start charging, to protect the battery.

When the output voltage is exceeded, or is undervoltage, the relay opens resulting which causes the circuit current to drop drastically. The boost code detects this sudden drop in the current by looking at the error value between the previous and latest current value and sets boost to its initial duty cycle and the system restarts.

A 1kΩ resistor is added in parallel with the battery as this allows the capacitor to have a discharge

path when the relay turns on and the output voltage initially spikes for a small amount of time. This prevents the relay from oscillating in a short span of time

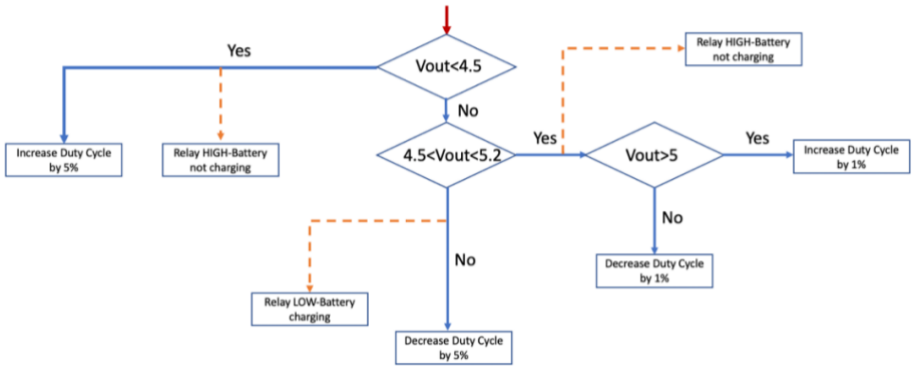


Figure 13: Algorithm for buck regulator

11.7 Limitations

The entire circuit operates poorly in dim lighting conditions when the current is below 100mA. To overcome this, bypass diodes were initially used, however, since there are no solar panels in series, the bypass diodes would be redundant and negatively affect the overall efficiency of the system. Hence, the system works well under good lighting conditions, but not in partial shade. A future consideration would be to create a system that is able to choose whether to operate as a buck or boost depending on the voltage and current constraints imposed by the solar panels.

11.8 Efficiency

The system has an overall efficiency of 70-80%. An 80% efficiency was calculated on a bright day and corresponds to an input power maximised at 3W (input voltage of 4.6V and current around 700mA).

11.9 How long does the battery take to fully charge?

The rated capacity of the battery is 5000mAh. The substation provides 500mA of current (on the same sunny day), therefore it would take 10 hours (estimated) for a battery to charge fully from a drained condition. However, in reality, it takes 14 hours to charge from a USB charging source. This is likely due to the battery charging at constant voltage and not drawing the same amount of current at all charge states; the battery was partially charged (50% charge) when the test was conducted.

12 Autonomous Drive

12.1 Vision

The Vision module is responsible for sending data on where objects in the arena are. It does this by sending the Control module information on the coordinates of the bounding box and the type of object it has identified. Using the coordinates of the bounding box, the Control module is able to calculate the distance and the angle of the object with respect to the rover. This is because further away objects have smaller bounding boxes so we can calibrate the distance from the rover with the size of the bounding box. Once the distance and the angle of the object have been found, with the respect to the rover, we are able to calculate the position of the object on the map. This is useful as the autonomous drive uses this information to avoid objects when driving around the arena.

12.2 Drive

When the rover operates autonomously, it drives on a strategic route (See Appendix G) in the arena to locate and plot the coordinates of the aliens, buildings, walls, and the fan, simultaneously, avoiding contact with them. The rover incorporates three ultrasound sensors and one camera as sources of input for detection and identification. By utilizing the vision module, the camera can identify and differentiate objects while quantifying their orientation and distance. The three ultrasound sensors

are strategically distributed across the rover increasing the overall scope of detection. A sensor is affixed to the front of the rover to detect objects in close proximity and aid the main camera to verify the distances of large non-coloured objects, such as walls. Conjoined with the two rear sensors, they provide reinforcement monitoring potential obstructions when manoeuvring.

All three inputs transmit data to the drive module via the control module located in the ESP32 in the form of coordinates and angles. The drive module fetches and receives the data to derive an angle of rotation and a distance to travel to reach an obstacle. When an item is obstructing the rover's trajectory, it will manoeuvre around it. In order to keep the rover's positional data accurate, upon detection of an arena wall or corner, the position of the rover is adjusted. This adjustment allows the rover to recalibrate itself regularly with respect to the wall and corners. This will improve the accuracy of the rover and reduce its chance of 'getting lost'. Upon completion of mapping the arena, the rover will hug the arena walls and navigate to the starting position.

13 Integration With Other Modules

The key aim when integrating the sub-modules together was to ensure that modularity between the submodules and the subsections, within each module, was kept. This aided tremendously when troubleshooting errors once two submodules were integrated together. Modularity meant that errors were more easily locatable as there was no need to look through the entire integrated code as they were localised.

13.1 Command and Control

The key objective in integrating the Command and Control submodules was linking the local database (used by the Control module) with the remote database (used by the Command module). This was done by the server sending MySQL queries to the local database, as mentioned in the command module section. Once the databases were linked, checks had to be carried out that the data was being correctly queried, and the appropriate information was being returned to the server.

Another key issue was faced in attempting to send remote commands to the ESP32 since it runs off a local network. This had to be resolved by connecting the server to the same network to enable this communication. The ESP32 listens for incoming UDP packets (from the Command server) and extracts the distance, angle and power information needed to carry out the remote instruction sent by the client. This information is then passed onto the drive module.

13.2 Control and Drive

Drive and control modules were integrated systematically, segregating the code. For instance, driving the rover straight was implemented first, then enforcing the rotating function. This approach allowed us to upkeep modularity within the code for the purpose of debugging and helped construct a substantial platform to program the autonomous algorithms.

One key issue that had to be dealt with once the two submodules were integrated together, was that the optical flow sensors stopped being detected within the code, resulting in undefined behaviour. The problem here was a small repeat of code that reinitialised the optical flow sensor, which caused it to malfunction. This highlighted the importance of checking through the altered parts of code thoroughly to ensure all variables are initialised to a given value and there are no repeats in code, unless necessary.

13.3 Command, Control and Drive

Once the Command, Control and Drive modules had been integrated, it was essential to ensure that the data traversed correctly from Drive to Control and finally to Command and similarly from

Command, through Control and back to Drive. This was key to guaranteeing that the mapping functionality and the manual control functionality of the rover would work well. When testing the manual control, it was vital to make sure that the rover correctly and consistently carried out the commands it was fed. An issue faced, was that once the Drive module was integrated with the Command and Control modules, the wheels stopped spinning at the same rates, which caused the rover to turn when travelling straight. Similarly, the rover would not rotate at precise angles given to it by the Command module and travel the exact distances either.

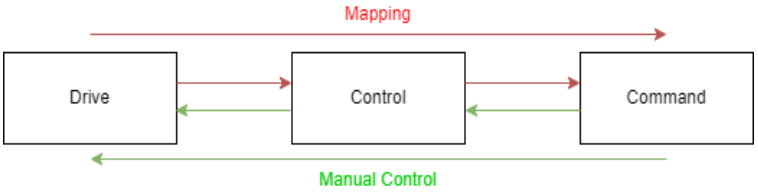


Figure 14: Integration of Drive, Command and Control

As mentioned in the Control module breakdown, the problem was solved by switching to a UDP connection when carrying out the manual control functionality of the rover. This would allow the rover to constantly listen to data that the command server would send, based on the different button presses made by the user on the remote controller. Therefore, the UDP connection meant that the rover would not travel and rotate precise angles and distances, but it instead could carry out commands until it stopped being fed the same button presses or no button presses.

13.4 Control and Radar

Integrating with the radar was fairly easy and not many issues were faced. The radar circuit was wired to an analogue pin on the ESP32 (which was also an ADC pin) so that the voltages outputted by the Radar module could be read. However, this introduced the first main problem: By using an ADC pin, any voltages (between 0V to 3.3V) read would be assigned a number between 0 -4095 that the ESP32 would read [5]. This relationship was mainly linear but would deteriorate at the extremities of the range (See Appendix H). Once determining the relationship, a simple algorithm was written that would indicate when the fan was present, ie. if the voltage surpassed a specific cut-off voltage. A cut-off voltage was 2.25V was chosen, which corresponded to a number of 2840.

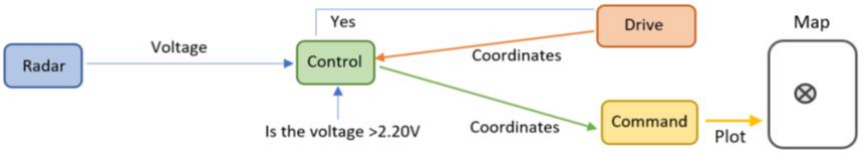


Figure 15: Algorithm for detection of fan

13.5 Control and Vision

Two serial channels were needed to establish communication between the Control module and the Vision Module. One channel was for reading the data transmitted by the FPGA, using UART. The other channel was used to output these values on a Serial Monitor to allow observation of what was happening.

A key design feature implemented during the integration was the formatting of the message sent from the FPGA(Vision) to ESP32 (Command). Originally, the plan was to send two 32-bit long messages that sent information about the four coordinates of the bounding box and the colour of the alien encountered. However, it soon became apparent that we could combine the two messages by only sending the x-coordinates of the bounding box and by encoding each colour of the alien, using three

bits. This meant it was possible to send all the information required by the Control module, within just one message, as opposed to two.

Another issue was that the Hex values transmitted across by the FPGA were being read as ASCII

values, and therefore were outputted as symbols on the Serial Monitor. Originally, the issue was thought to be the baud rate, however, later saw that it was because the *readString()* function was used in the EPS32 code, instead of the *readBytes()* function, which caused the code to be interpreted as ASCII codes rather than hex values.

13.6 TROUBLESHOOTING

The general approach to troubleshooting was to use "Print" commands to check outputs and where the code may get stuck. The modularity of the design made it significantly easier to localise errors since it was possible to test each individual module before joining them together and testing again. With these two principles in use, the debugging process ran smoother since it was easier to spot errors.

Another valuable process was to keep code history, so that going back to initial designs and attempting a different approach was a feasible technique. This allowed alterations within individual modules before retrying them in the integrated code, to ensure systems were working correctly on all levels. For example, when debugging the manual control functionality of the rover, any minor changes made to the Drive section could be tested in isolation, before checking the overall functionality within the integrated code.

14 CONCLUSION AND EVALUATION

The project was to design and build a rover that could be controlled both manually, using a website and autonomously on an arena, where the rover will navigate and identify aliens, buildings, and a fan infrastructure. The rover monitors its surroundings using the camera (Vision), the OFS and ultrasound (Drive), and the Doppler Radar (Radar). The vision module checks whether pixels meet a certain colour threshold and filters and blurs the image captured by the camera to remove random noise. By utilising the vision module that gives an effective distance from an object, and an ultrasound fixed at the front of the rover, the drive subsystem can detect walls and avoid obstacles autonomously. The drive subsystem has three main hardware components, the ultrasound sensors, OFS (obtains distance travelled and coordinates) and gyroscope (obtains angles). A proportional controller is implemented in the drive module that tries to minimise small errors that arise due to friction or uneven surfaces, to keep the rover in the desired trajectory of motion. By measuring angles of rotation and distance, the drive module helps to map the arena and locate the rover. The radar module helps detect fans under the arena by using an external circuit that processes the IF signal from the Doppler radar. The data from the vision, drive and radar subsystems are processed by the ESP32 and then transferred to a database which is then read by the command module that presents data visually on the web app. The power substation can charge the battery using a 2 SMPS topology, at an efficiency of 70-80% under good lighting conditions. Overall, the team believes that the project objectives have been successfully met and implemented in a rover capable of being manually remote-controlled, as well as incorporating an efficient autonomous drive system and algorithm for the identification and detection of aliens, buildings, and a fan.

15 REFERENCES

[1] R. Santos, “ESP32 Useful Wi-Fi Library Functions,” [Online]. Available: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/#1 . [Accessed May 2022].

[2] R. Santos and S. Santos, “ESP32 with MPU-6050 Accelerometer, Gyroscope and Temperature Sensor,” [Online]. Available: https://randomnerdtutorials.com/esp32-mpu-6050-accelerometer-gyroscope-arduino/. [Accessed June 2022].

[3] L. Day, “WHAT EXACTLY IS A GAUSSIAN BLUR?,” 21 July 2021. [Online]. Available: https://hackaday.com/2021/07/21/what-exactly-is-a-gaussian-blur/. [Accessed June 2022].

[4] I. Glasner and J. Appelbaum, "Advantage of boost vs. buck topology for maximum power point tracker in photovoltaic systems," *Proceedings of 19th Convention of Electrical and Electronics Engineers in Israel*, 1996, pp. 355-358, doi: 10.1109/EEIS.1996.566988.

[5] R. Santos and S. Santos, “ESP32 ADC – Read Analog Values with Arduino IDE,” [Online]. Available: https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/. [Accessed May, June 2022].

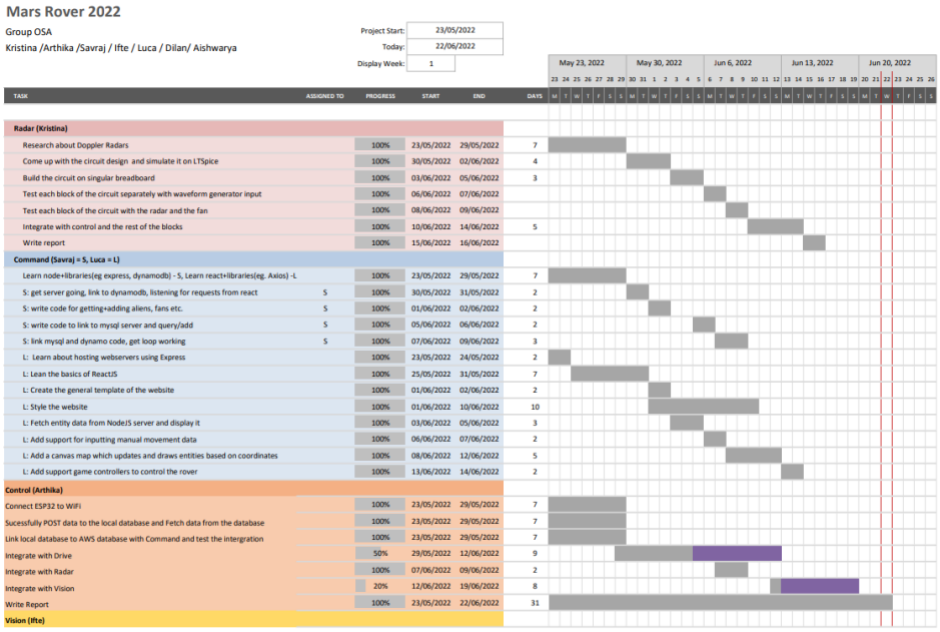
16 APPENDIX

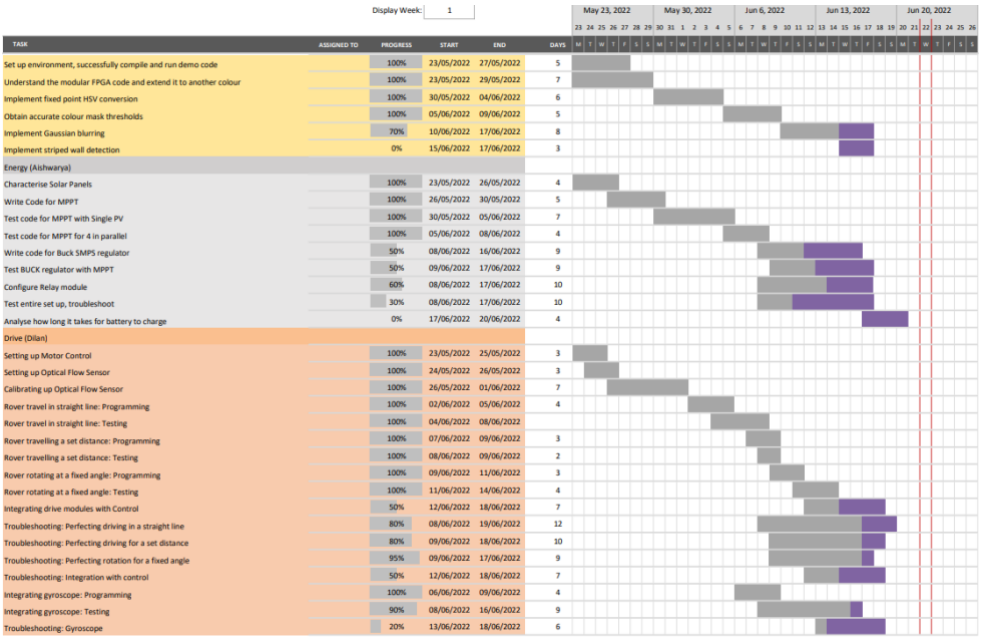
Appendix A: Gantt Chart

Link to Gantt Chart

https://imperiallondon.sharepoint.com/:x:/r/sites/MarsRoverProject-GroupOSA-EE/_layouts/15/Doc.aspx?sourcedoc=%7B07445E4C-7F3E-44B2-A7C2-8246913A312B%7D&file=Project_Management_Mars%20Rover%202022.xlsx&wdOrigin=OFFICECOM-WEB.MAIN.REC&ct=1655901248864&action=default&mobileredirect=true

Screenshots of Gantt Chart

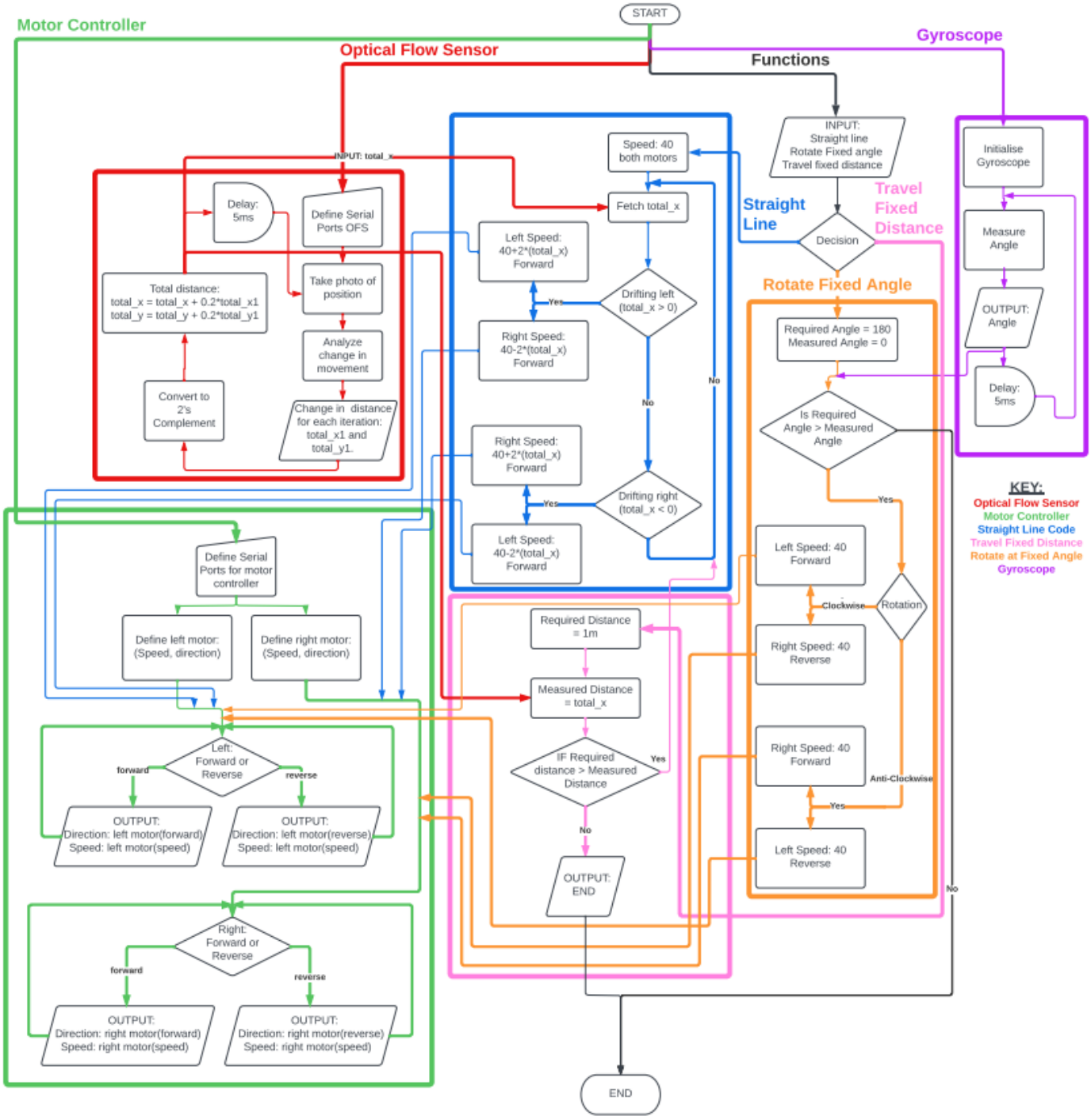




Appendix B: Meeting Schedule

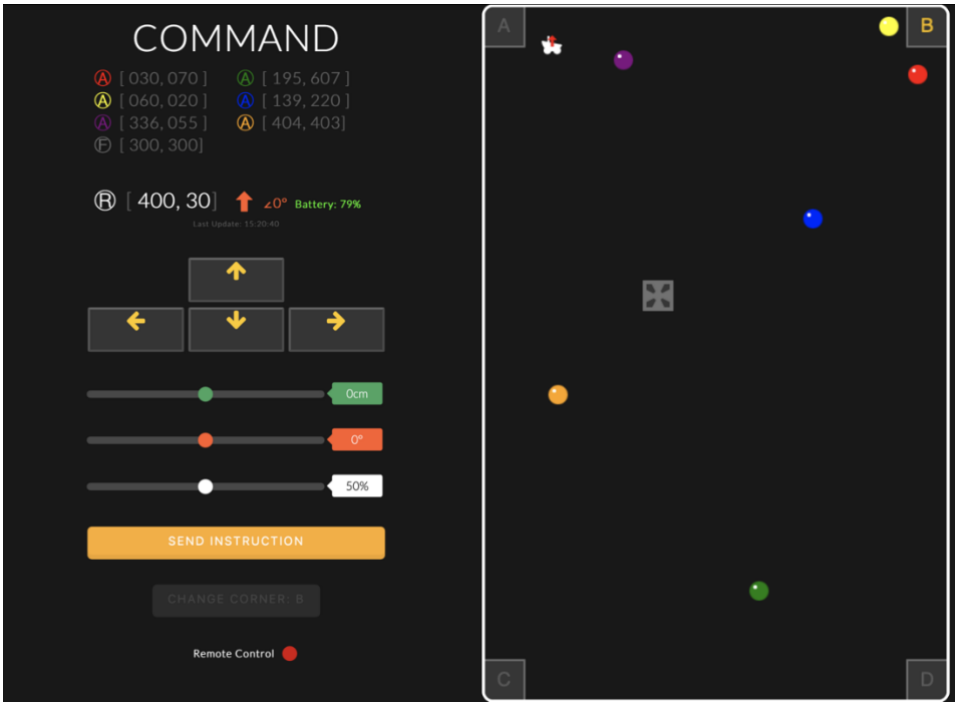
Topic	Minutes Allocated
Brief update on progress made within each submodule	30 minutes
Questions for each submodule	10 minutes
Deadlines for each submodule	10 minutes
Discussion on organising the next meeting and any meetings with lecturers	10 minutes

Appendix C: Drive Module Flow Diagram



Appendix D : Command Module

Appendix D.1: Client Web App :

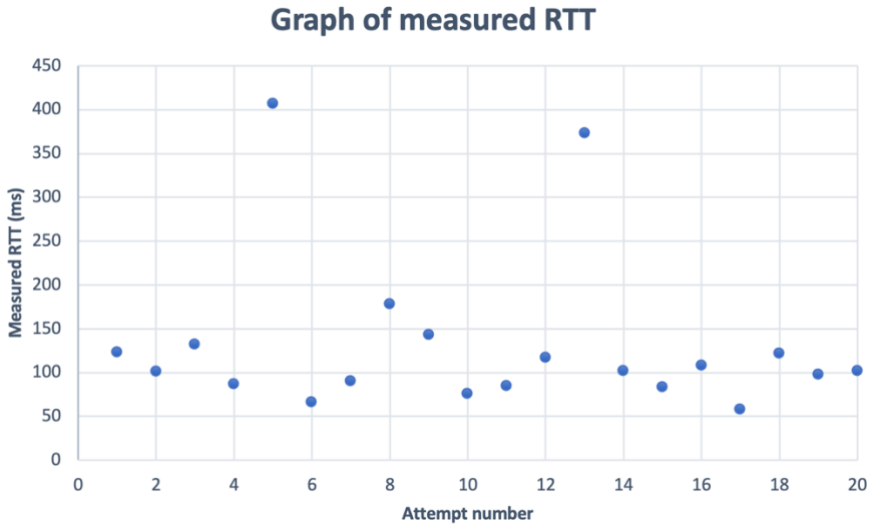


Appendix D.2: Measured RTT Values

Table for measured RTT between client and server:

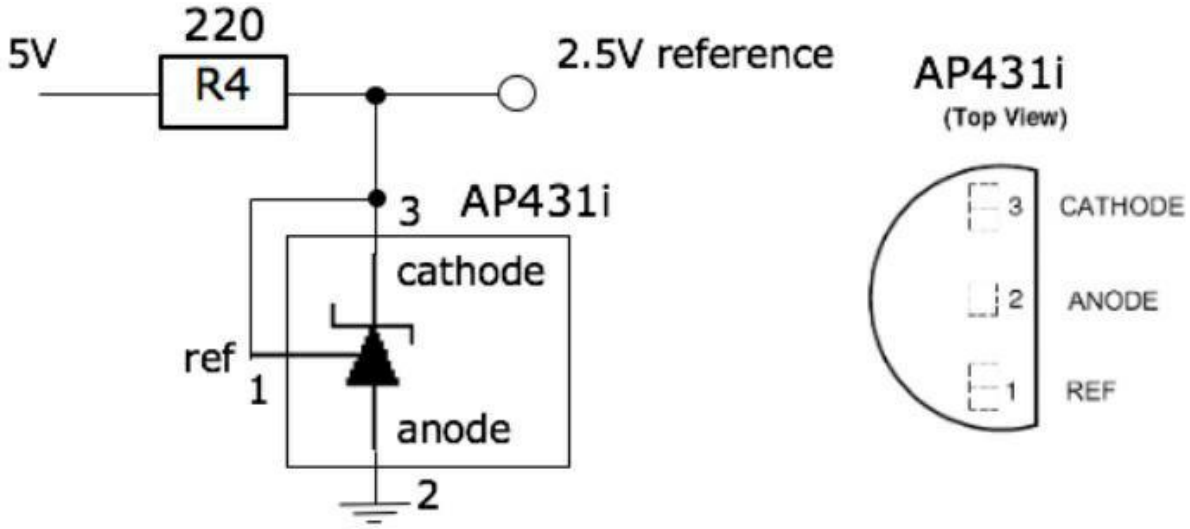
ATTEMPT NUMBER	MEASURED RTT (MS)	AVERAGE RTT (MS)
1	123	104
2	101	
3	132	
4	87	
5	407	
6	66	
7	90	
8	178	
9	143	
10	76	
11	85	
12	117	
13	373	
14	102	
15	83	
16	108	
17	58	
18	122	
19	98	
20	102	

The measurements of 407ms and 373ms were taken to be outliers in the average RTT calculation. These are most likely the result of rerouted/lost UDP packets. Therefore, this average could be described as an amortised average since the time taken was in a reasonable range for the majority of the values.



Appendix E: Voltage Reference Circuit

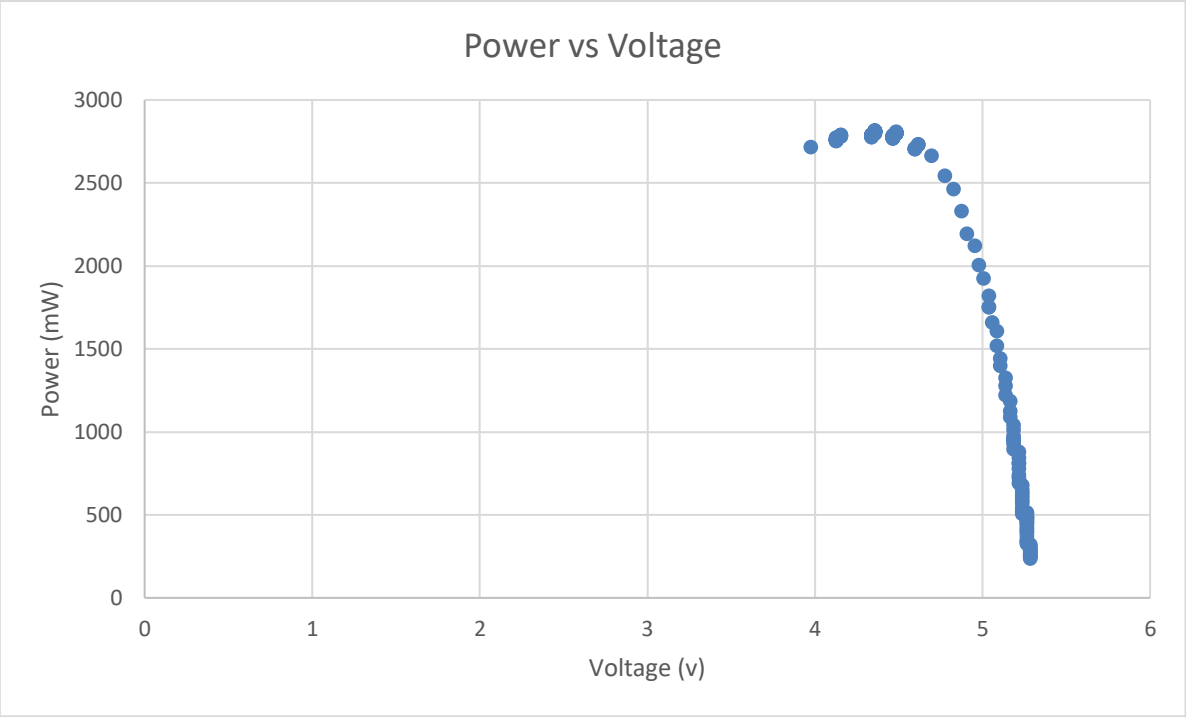
The following circuit was implemented to create a voltage reference used to centre the wave halfway between the power rails of op-amps (Ground and 5V)



Appendix F:

Power Graphs

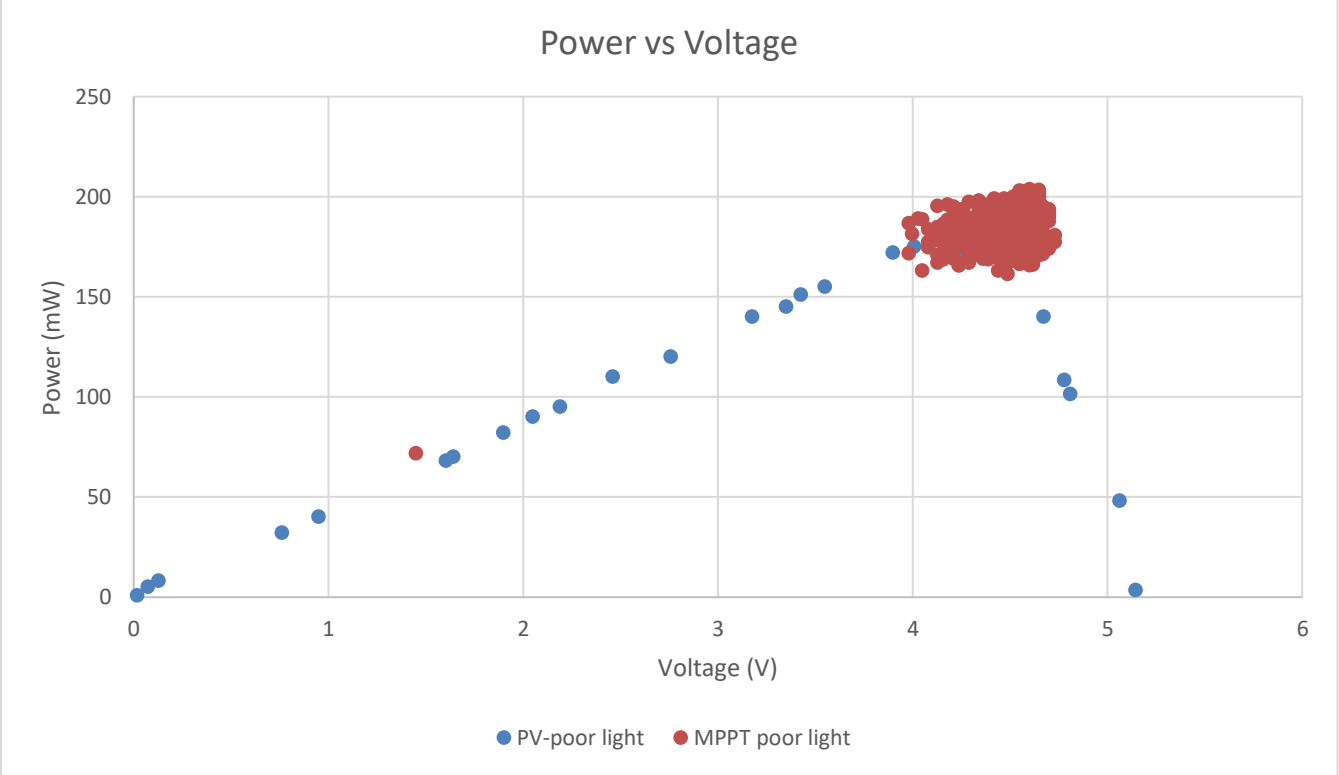
MPPT for Good Lighting



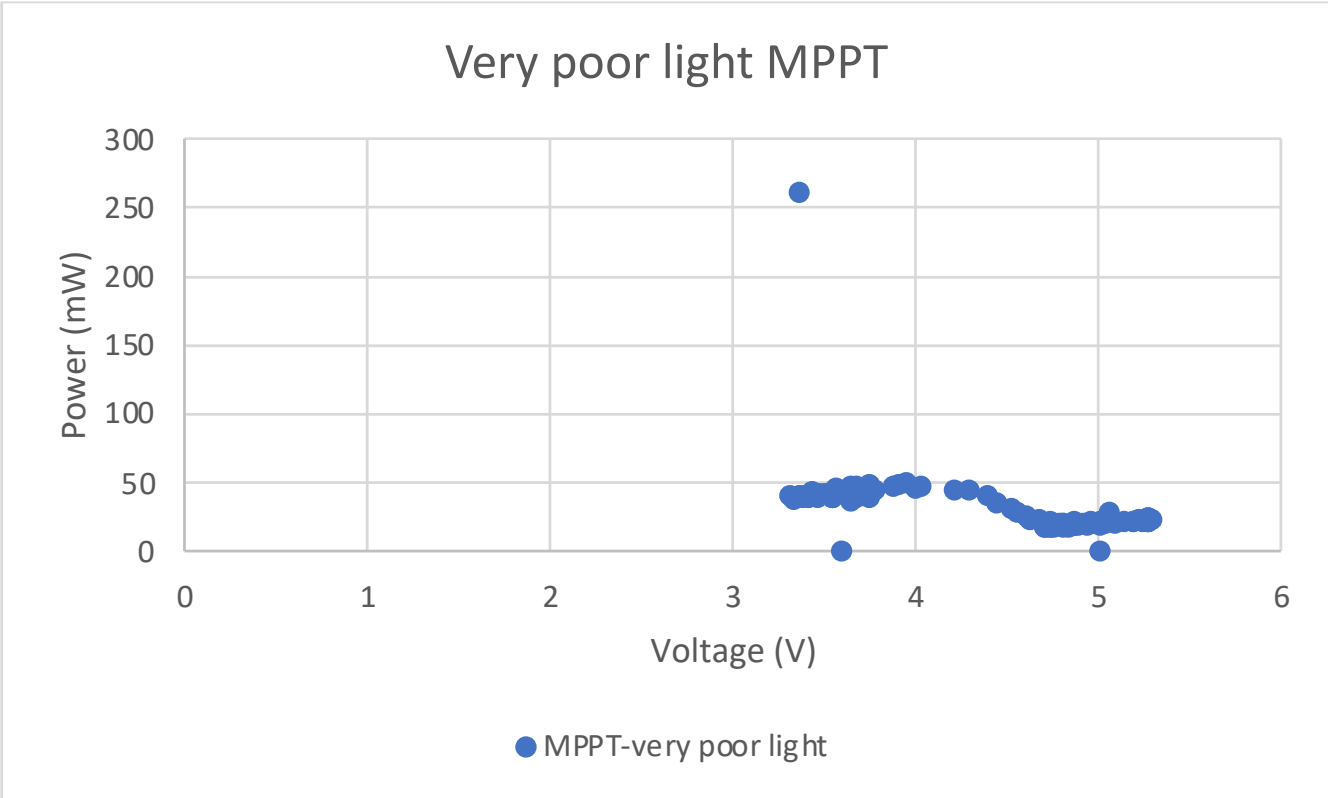
MPPT for boost with a saturation at 0.01 duty cycle - MPPT takes a long time to reach desired value

MPPT for Poor Lighting

MPPT indoors in poor lighting conditions

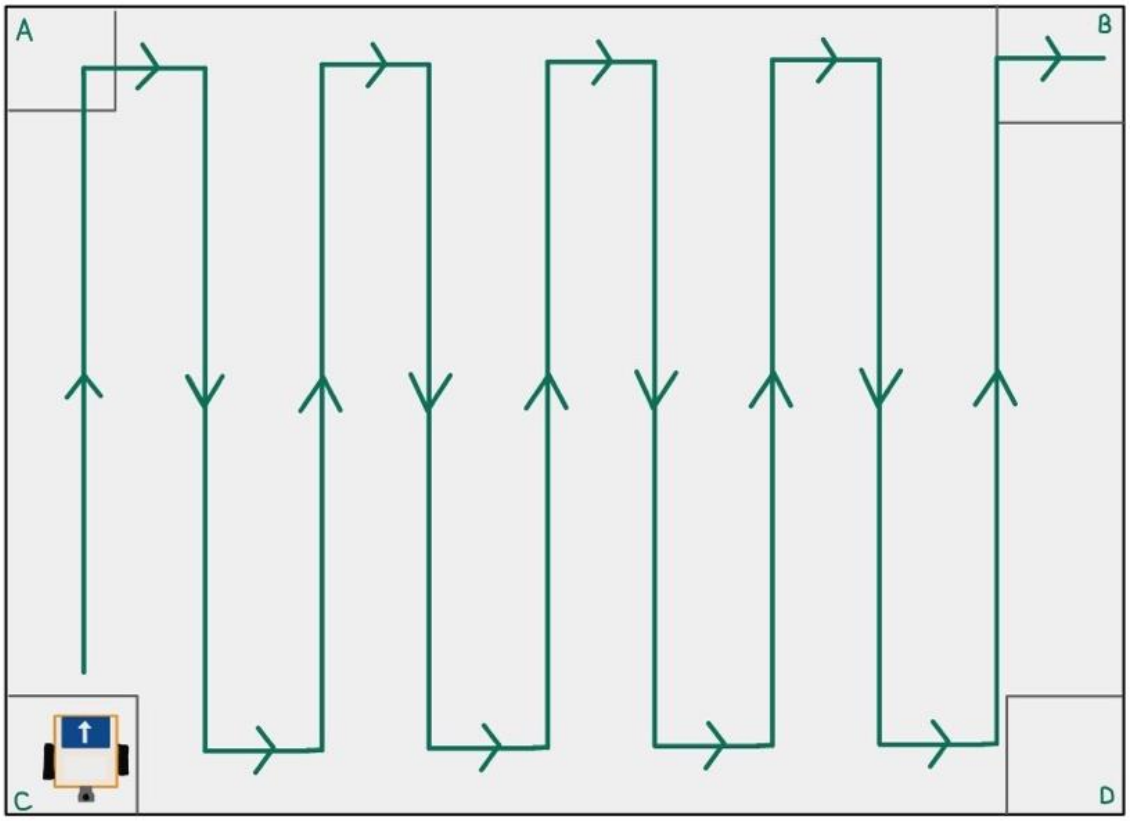


MPPT for Very Poor Lighting

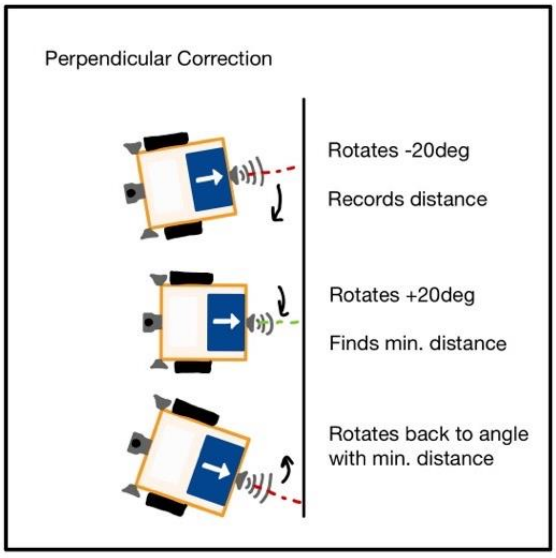


Appendix G: Diagrams for Autonomous Drive

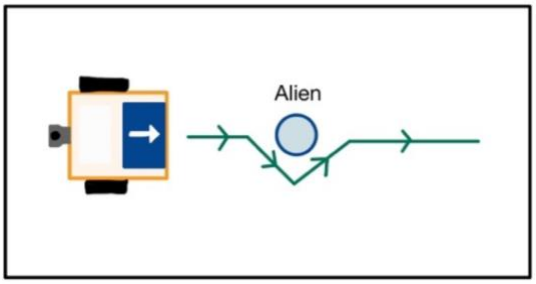
Autonomous Drive Route



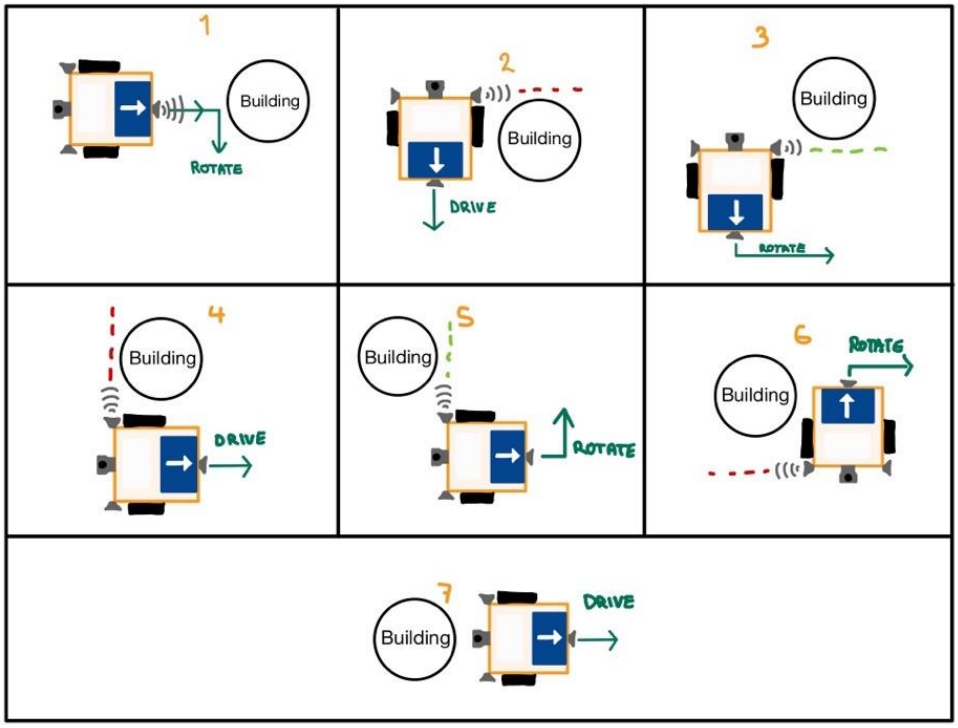
Angle Correction at Walls



Alien Avoidance



Building Avoidance



Appendix H: Analogue Voltage to ADC Voltage

Below is a graph showing the approximately linear relationship between the analogue voltage and the ADC voltage. As you can see the analogue voltage varies between 0V to 3.3V and the ADC voltage varies between 0 to 4095.