Imperial College London

2022 Mars Rover Report

ELEC50008 - Engineering Design Project 2

Unity Mars Rover

Authors:
Charmaine Louie, CID: 01711892, 2nd Year EIE
Himanish Joshi, CID: 01867208, 2nd Year EIE
James Ong, CID: 01848230, 2nd Year EIE
Matthew Setiawan, CID: 01917068, 2nd Year EIE
Amy Yau, CID: 01868783, 2nd Year EEE
Hao Jian Mang, CID: 01852853, 2nd Year EEE
Milan Paczai, CID: 01873004, 2nd Year EEE

Submitted to:
Dr. Bouchaala, Adam

June 22, 2022

Contents

1 Abstract

This project aimed to design an autonomous rover system capable of exploring an alien colony on Mars. The system was required to navigate a test arena while building a map to show the locations of aliens and their underground infrastructure. The rover was also required to navigate without direct, real-time control from the user and to also avoid colliding with aliens or their structures. Furthermore, an additional, separate energy system was designed to recharge rover batteries from solar energy.

2 Introduction and Background

A Systems Engineering approach was taken to build a rover that meets the requirements delineated in subsection 2.1. This separates the rover into the different subsystems and enabled a large challenging problem to be decomposed into smaller, more manageable ones. The following subsystems were identified in Figure 1:

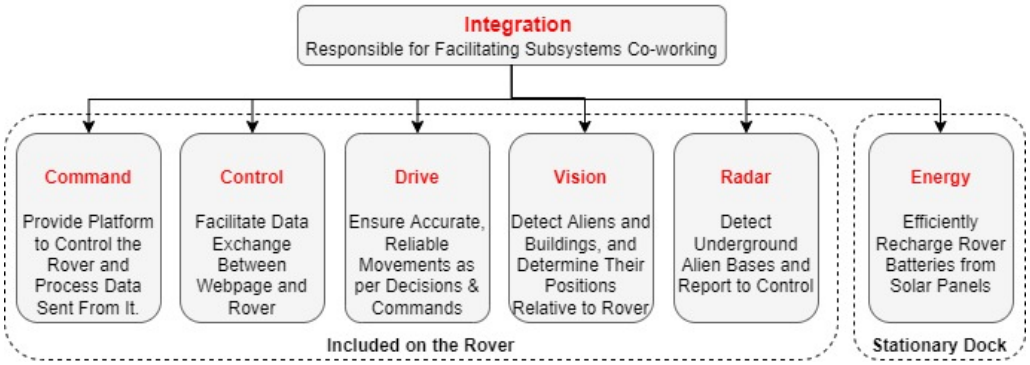

Figure 1: A block diagram of Unity Mars Rover

2.1 Rover Requirements

The requirements identified for the rover are shown in Figure 2 below:

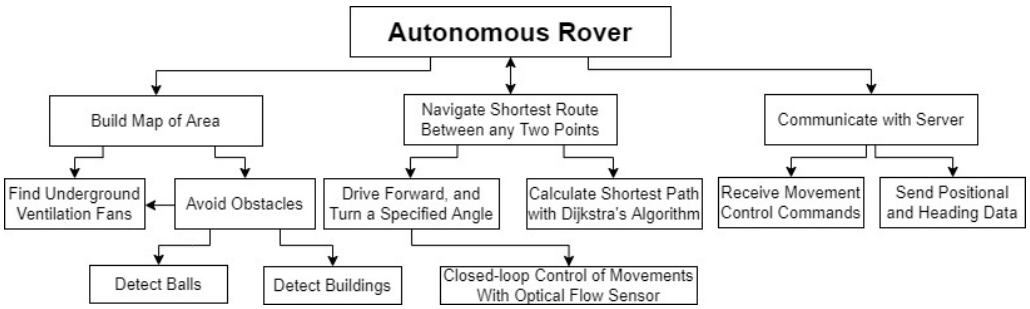

Figure 2: Block diagram of requirements for Unity Mars Rover

In the table below, the requirements of the rover are discussed:

Subsystems	Functional and Non-Functional Requirements
Command	Display information about the state of mission on a grid using informative images about obstacles encountered Enable users to control the movement of the rover Split website into several pages to display different information and increase security
Control	Establish a WiFi connection and a low-latency communication protocol between the rover client and web-server Ensure the web application and overall system architecture are reliable and accurate
Drive	Drive straight, fast and accurately for a set distance Precisely turn the rover according to a specific angle Adapt to different lighting conditions without compromising speed
Vision	Differentiate between different coloured balls and detect the distance to the ball Utilise edge detection to identify alien structures and determine the corresponding distance
Energy	Charge the battery with the solar panels safely and as efficiently as possible
Radar	Detect and locate the reflector The reflector's coordinates should be sent and mapped on the frontend

3 Team Work and Project Management

After reviewing the systems mapping and rover design, each member decided which subsystem they wanted to develop. Scrum-based project management methods were applied to enable agile and robust problem-solving. This was important when a member had an outstanding knowledge of another subsystem and could be asked to provide suggestions and feedback.

3.1 Team Structure

Our work was coordinated using a flat structure to improve communications and creative thinking. The table below shows which subsystems each member developed (all members also worked on integration).

	Amy	**Charmaine**	**Himanish**	**James**	**Kelvin**	**Matthew**	**Milan**
Subsections	Drive & Radar	Command & Control	Vision	Command & Control	Energy	Control & Vision	Control & Drive

3.2 Timeline

To coordinate our work and allocate tasks effectively, we created a Trello dashboard for each subsection and used a GANTT chart to set deadlines. New tasks were added to the dashboard and GANTT chart approximately twice weekly. See Appendix A for the full Trello Dashboard and GANTT chart.

3.3 Meetings

During the development of the project, we held daily meetings and continued working together throughout the entire duration of the project. This improved information flow and facilitated swift decision-making.

3.4 Reflection

In the beginning we found it challenging to map out the different subsystems, their exact requirements and the tasks these would entail, due to a lack of knowledge of the nature and functions of the different subsystems that would need to be developed. Upon reflecting on our work, we could have spent more time in the initial planning stage to map out the requirements at the beginning. This would have reduced the time spent on integrating the different subsystems.

4 Design Process

4.1 High Level Design

The high-level diagram of the rover provides a more detailed explanation of each subsystem and its linkage. It is shown by Figure 3. Notably, the connections and communication protocols used for linking these subsystems are shown in blue.

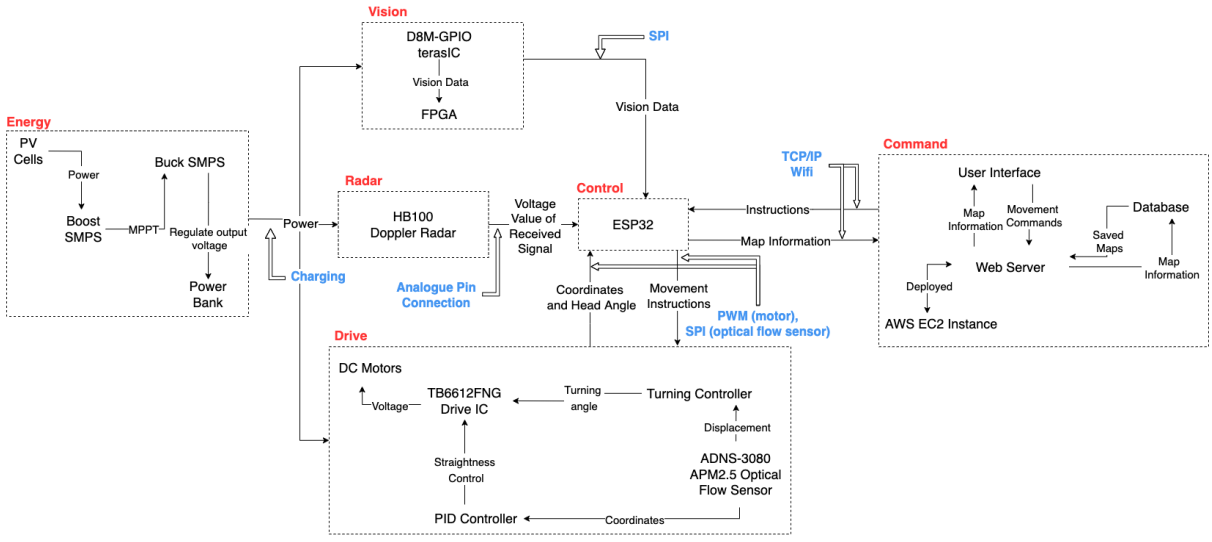

Figure 3: Block diagram of the structural design of the rover

4.2 Module Design and Implementation

4.2.1 Command

The Command subsystem allows the users to control the motion of the rover through a web application and also displays data retrieved from the rover.

Web Application

The web app is created as a Django Project, which uses quintessential frontend development languages and a Python backend. Through the use of forms and buttons, user inputs are sent with HTTP GET and POST requests. GET and POST were chosen depending on the content of the user's input (e.g. sensitive information used POST requests with CSRF tokens). The web application consists of 4 main pages (see Appendix B). The first display is a login page which provides security as it requires valid user credentials. Here, existing user credentials are stored using 256-bit SHA encryption. After logging in, users are directed to the main web page which displays a map of the rover's position and surroundings. On this page, users can then select either autonomous or manual modes for navigation. The web page also features buttons to control the heading angle of the rover when in manual mode. Initially, users were asked to input numerical values for distance and use a slider to control angle, however, buttons were chosen instead for ease of use.

The main map display is produced by looking up the rover position in the database and allocating a 61x61 grid map. From here, information about tiles in the rover's 9x9 vicinity will be processed and displayed accordingly on the map by looking up the database values (see Appendix B Figure 33). Allocating an extremely large map allows the rover to have the flexibility to start in any position and also ensures the rover will not move out of range. After conducting 30+ tests from random arena start points, the efficacy of the rover's navigability can be verified.

Databases

Databases played a pivotal role in our project and are essential for showing the current map and presenting previous mission logs. Even though Django supports a wide variety of databases, an SQLite3 database was implemented in our design as it was powerful yet lightweight and could fulfil the requirements of this project.

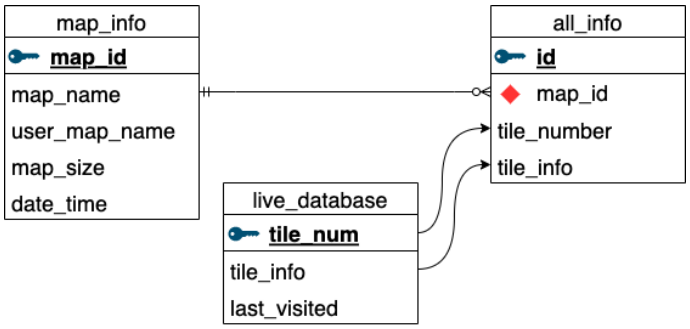

Figure 4: Schema of database

Our architecture consists of 3 tables: map_info, all_info, and live_database, each of which has important functionalities. The live_database table is the most critical for our subsystem as it is altered based on the data sent by the client and used to display the current map. The attribute last_visited is used to store the last known rover position while tile_num acts as the primary key. Meanwhile, tables map_info and all_info are useful when saving the current state of the map. A user can specify a map name and upon saving a map, a unique map_id and auto-filled date-time attribute will be generated. Following this, data from live_database will be transferred into all_info holding a foreign key (map_id) that matches the corresponding map. As seen in Figure 4, there is a 1 to many relationship between map_info and all_info, since many tiles correspond to 1 map. Hence, a foreign key is inserted into all_info. Attributes in live_database such as tile_num and tile_info will be inserted into all_info when the user saves the map.

Deploying on AWS EC2

After adding all functionalities to Django Project, the team decided to deploy the web app onto AWS EC2. An application server Gunicorn (WSGI) and an open-source server NGINX were needed to successfully deploy the system on EC2. Gunicorn translates requests between NGINX and the Django Project [1]. However, it was discovered through testing that deploying the web app onto EC2 would cause delays which would affect the functionalities of the rover. Therefore, EC2 was not used in the main implementation.

4.2.2 Control

Control Functionality and Operation Protocol

The Control subsystem plays a crucial role in the rover design and enables individual subsystems to be integrated. This system consists of a TCP server interfacing with the web application and a WiFi-Client on ESP32 that processes Vision and Drive data. Client data is sent as a comma-separated string containing coordinates, tile info and angle fields. The TCP server is vital for processing POST requests from Command while also performing CRUD operations on the database based on the data received. Additionally, the server must distinguish between the modes of operation. An overview of the flow diagram can be seen in Figure 5.

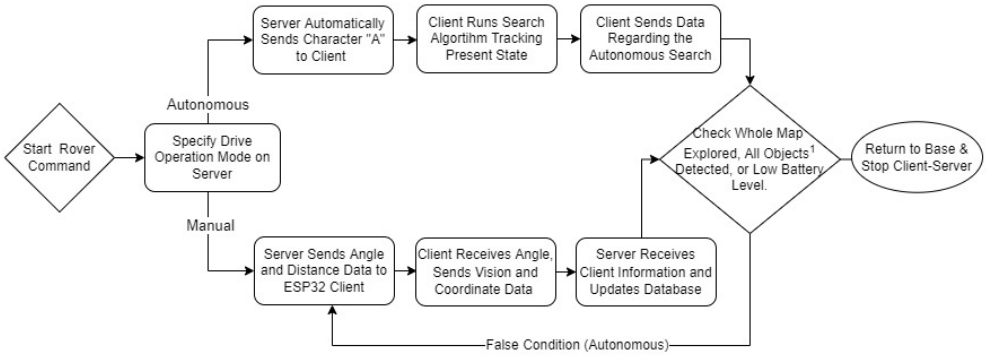

Figure 5: Overview of Control Module Flow Chart

Design Decisions

The Control subsystem prioritised high reliability and low latency. To ensure reliability, TCP and a database were used which will ensure OSI transport and session layer reliability respectively [2]. The database guarantees reliability as it stores critical map information (see Figure 4) allowing Command to continually display information independent of the TCP server.

To achieve lower latency we used a persistent TCP connection, which reduces the number of RTTs for transmission. When the rover navigates we send data directly from client to server, ensuring low transmission time and consistent packet delivery. The mean time taken to run the server code including accepting connections and both sending and receiving messages (using a dummy python client) was under 25ms across 30 iterations. When the ESP32 is used there were more delays but this does not hinder performance.

Control Subsystem Server and Client Conversions

The primary function of the Control subsystem is to obtain the current rover position and update the values in the database using the field of view information provided by the Vision subsystem see Figure 7. The client (rover) transmits its present coordinates, whether a fan is present, and information from each of the four squares in its field of view to the server. The server is also required to track the heading angle of the rover to ensure an accurate mapping of the environment (stored in a database and displayed on the website). To update the database Polar coordinates were converted to Cartesian coordinates for a more exact representation. To perform this conversion we utilised the standard 2x2 rotational matrices with heading angle used as parameter θ see Figure 6. This was extremely versatile as it could facilitate the mapping for any angle that the rover faces and at any position in the terrain allowing accurate mapping.

$$\begin{bmatrix} cos\Theta & sin\Theta \\ -sin\Theta & cos\Theta \end{bmatrix}$$

Figure 6: Rotation Matrix Shift Coordinates

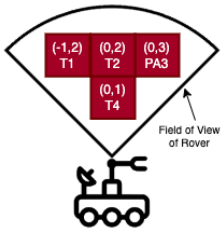

Figure 7: Vision Field of View

Searching Move

The initial search move was based on a depth-first search on a directed graph, where a coordinate represents a

graph node and an edge refers to an adjacent coordinate at 90^o. We only need to search for undiscovered nodes as discovered nodes are stored in a dictionary. To reduce backtracking, the search order would be left, up, right and down which results in a spiral-like search (see flow chart in Appendix C for implementation).

In a free open space, this algorithm would be very reasonable, however in the Mars arena with many walls, which results in a lot of backtracking. It was thus modified to a back and forth-like traversal of the map whilst avoiding obstacles which is more optimised for the arena. Figure 8 describes the process.

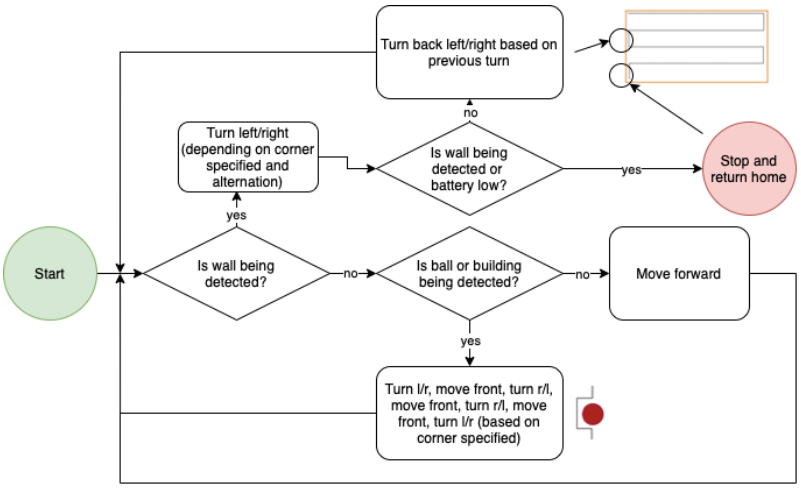

Figure 8: Modified Search Move Flow

Moving to a discovered Coordinate
Once the map was completed we converted our grid structure into a directed graph. This allows the rover to move to a specified coordinate based on its current position. We initially considered angling the rover to face the desired coordinate and avoid obstacles along the way. However, this was unsuccessful when navigating complex street/maze arenas. Our final implementation involves using the A-STAR algorithm which offers high performance and accuracy, whilst having lower time complexity compared with Dijkstra's Algorithm.

4.2.3 Drive

The purpose of this subsystem is to control the movement of the Mars Rover and determine its location on a map. This consists of PID controllers to drive straight and turn precisely, keep track of its coordinates, and determine its head angle. This subsystem consists of two generic DC motors controlled by a TB6612FNG Drive IC, along with the ADNS-3080 APM2.5 optical flow sensor and an MPU6050 gyroscope, connected to the central ESP32 via an Adafruit Feather board.

Several variables could influence the performance of the individual motors and the overall movement of the rover. These included battery charge, lubrication of the motors, electromagnetic interference of solar flares, and weather conditions such as temperature and wind. Due to these time-variant variables, the rover could be diverted from its path as a result of imbalances in the torques produced by the individual motors. In the following sections, the designs developed to meet the functional criteria with these variables are discussed.

Locating the rover on a map
The optical flow sensor was used to find the location of the rover accurately. It provided translation information into x-y directions, relative to the sensor's position. By converting these into polar coordinates, the head angle θ and the translation r could be used to locate the rover on the map and enable the control of both straight and turning movements, as discussed in the following sections.

Accurate control of forward movement
Due to the aforementioned variables affecting the individual torques of the motors, a closed-loop feedback controller had to be used. Since there were no sensors included in the motors providing feedback about the position of the rotors (how Hall Effect sensors do in Brushless DC motors), the optical flow sensor along with the gyroscope had to be used to provide the necessary feedback. The θ measurement feedback provided by these two sensors along with the required heading angle, which was fed from the control module, were used as inputs to the software-based feedback controller.

While there are various types of feedback controllers such as a proportional and derivate controllers, ultimately a full PID controller [3] was used due to its precision and robustness. [4] This ensured that all variable and non-

variable conditions affecting the torque were eliminated.

The structure of the controller developed is shown in Figure 9 below.

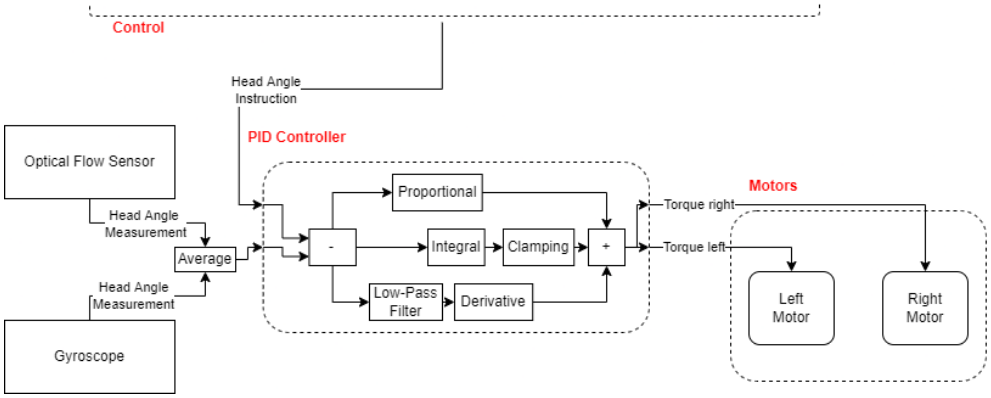

Figure 9: Functioning of the PID controller for straight forward drive

The control function used for ensuring straightness control can be described by the pseudo-code in Figure 10. Here, the function of the Clamping component is to ensure that the gain of the integral path is saturated to avoid excess overshoot. Which is a result of an undue buildup of integral error, for example when one wheel losing traction. A Low-Pass Filter was used to eliminate the acting of controller on high-frequency disturbances due to random noise.

```
1.  heading_angle = avg(optical_flow_measmnt(),gyroscope_measmnt());
2.  Kp = 0.5; Ki = 0.8; Kd = 0.6; //dummy values
3.  KP = head_angle - required_head_angle;
4.  KI = clamp(KI + KP) //implement clamping
5.  KD = avg(KP - four_previous_errors) //implementing the filter
6.  crtion = Kp*KP + Ki*KI + Kd*kD
7.  MotorLeft.drive(speed + crtion); MotorRight.drive(speed - crtion);
```

Figure 10: PID controller pseudo-code for motor control

The controller parameters were determined experimentally, using a trial-and-error approach and a testing library, that was developed to allow for the instantaneous wireless adjustments of the gain parameters. The adjustments were made based on three important intuitive principles: increasing K_i reduced the speed of response, increasing K_p reduced the rise time, while increasing K_d improved stability and reduced the overshoot.

Accurate control of rotational movement

To ensure accurate control of the rover's rotational movement, another PID controller was designed and implemented, using the same structure and trial-and-error approach as mentioned above. The difference was that this controller only used the gyroscope sensor (due to much higher accuracy) and was only operational during turning motion. Additionally, this function terminated once the head angle error was below 0.5º.

Fast movement in all lighting conditions

The accuracy of the optical flow sensor measurements was highly influenced by the focus distance used and by lighting conditions. The focus distance was adjusted by a trial-and-error approach, based on the focus quality feedback from the optical flow sensor. Accurate operation in varying lighting conditions was ensured by adding a high-luminosity and small, 30° viewing angle red LED providing concentrated light to the surface below the sensor.

To integrate the Drive module with the other subsystems, a C++ library was developed to control the motors, read the optical flow sensor data, implement all the controllers, and therefore enable seamless integration with the rest of the control code.

4.2.4 Vision

The overall Vision subsystem can be broken down into a few core components: colour detection, bounding, alien building/edge detection, distance approximation, and ESP32 communication.

Colour Detection and Filtering
Instead of using an RGB colour space, HSV colour space was used to obtain a representation that separates the colour (hue) information from the intensity (saturation and value). This enabled effective colour detection by considering different hue ranges and varying the saturation to account for changing lighting conditions. Initially, the group utilised the OpenCV RGB-HSV conversion algorithm [5] but then noticed there were several timing issues due to the FPGA not having embedded division modules. Hence, the algorithm was optimised by using right shifts to approximate division by constants and scaling the variables from 0-1 to 0-255 to avoid floating-point representations. The hardware support for multiplication ensured that timing constraints were satisfied.

A problem faced when determining HSV ranges for each colour was the inconsistent lighting conditions in differing environments. It was noticed that objects on edges of the field of view were darker than objects in the centre due to the lens hood causing a vignette effect [6]. Hence, the MATLAB "colorThresholder" [7] was used to obtain more accurate HSV ranges for each colour. We also split the image into 3 sections and used a different colour mask for each section enabling the camera to detect colours in a wider field of view.

Another issue that needed resolving was "salt and pepper noise", referring to erratic disturbances in the brightness of pixels. This results in colour detection picking out erroneous random background pixels. To mitigate this effect, we tested several variations of Gaussian smoothing operators in Python (which enabled quicker testing)[8]. This filtering involves using 2D convolutions to remove detail and noise from images by taking each pixel as a weighted sum of its neighbouring square RGB pixels. Whilst in Python, the filtering had positive effects in removing speckled noise, challenges were faced in the hardware implementation in Verilog. The matrix convolution requires storing previous rows of data. This memory buffering was achieved by using a shift register IP block in Qsys [9]. For a 5x5 square, off-chip memory had to be added to the pipeline which caused several timing issues and prevented video output from being viewed. Reducing to a 3x3 kernel greatly reduced memory requirements. The trade-off between image detail and noise reduction was noticeable as seen in Figure 11 but allowed for much cleaner colour detection. However, it was later found that the reduced image quality severely impacted distance estimation. Finally, a modified 1D Gaussian kernel that sufficiently met the objective of noise-rejection was used, as shown in Figure 12. This brought additional benefits of reduced compilation times, fewer memory bits and decreased power consumption (extremely important for FPGAs in remote environments).

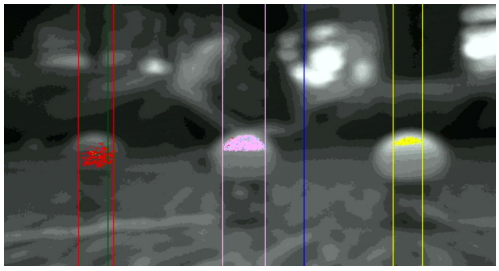

Figure 11: 3x3 Gaussian Smoothing

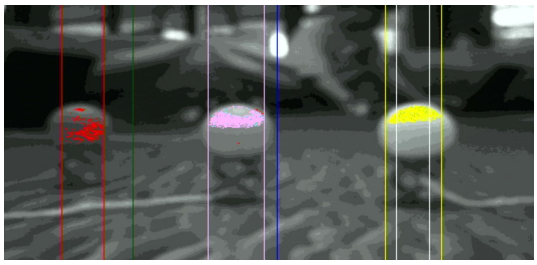

Figure 12: Modified 1D Gaussian Smoothing

As can be seen in Figure 11 and 12 the 1D implementation has comparable noise reduction to the 3x3 kernel whilst also maintaining the perspective of distance with the red ball being the furthest and yellow the closest.

Bounding Algorithm
This part of Vision deals with bounding/locating balls on the screen post colour detection. Bounding is needed to determine the ball position and the distance from the rover.

The group implemented an algorithm that will only bound the largest block of a colour and ignore other smaller specks of a colour. The initial algorithm was implemented using divide and conquer recursion in Python. To find the horizontal bound for the biggest blob of a colour the image would be split into two sub-images (left and right) and the largest blob for each image would be found. The base case is when the image is down to a width of 1, and here we would simply return the bound $[x_{current}, x_{current}]$ if the searched colour existed or [0,0] if not. For the combination step, there were 3 cases to consider: largest bound overlapping from left to right, left having largest bound and right having largest bound, this is very similar to the maximum sub array problem.

Although the initial algorithm worked in Python, it is not synthesizable due to recursion, so some changes and limitations had to be added to make this concept work in Verilog. Instead of splitting the image down to the last pixel, we decided to compromise by only splitting the image down to 8 blocks then use the naïve algorithm (of finding max and min coloured pixel) to find the bounds for each of the 1/8th block. Figure 13 shows this algorithm

being applied to a 1x16 image. Figure 14 also shows how the red bounding box bounds to the larger and closer red ball, it also ignores the smaller specks of red noise in the background. This will be extremely crucial for accurate distance detection.

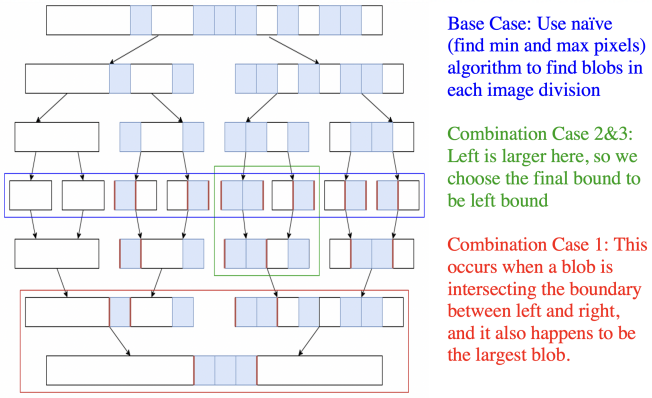

Figure 13: Divide and Conquer Algorithm

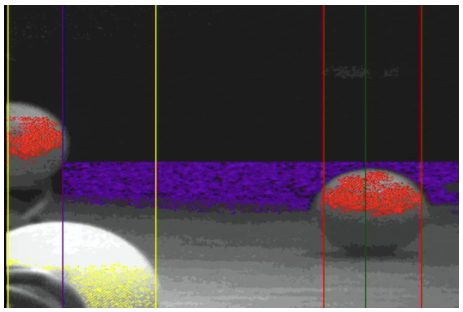

Figure 14: Ignoring Specks

Building and Edge Detection

Alien buildings had an easily recognisable black and white striped pattern. Our approach was to use registers for previous pixels. A building line would be detected if the previous pixel is black and the current is white and vice versa, and we then store the x-coordinate of the line. 8 registers were allocated to store the 8 different x-coordinates for multiple lines. To ensure that we were detecting a building and not just any edge, we calculated the difference between distances of x-coordinates and made sure they remained consistent.

Asides from improving colour detection, this method was further improved by applying a Gaussian filter to avoid detecting noise as false lines. The original convolution kernel used values [0.61, 0.242, 0.383, 0.242, 0.61]. However, these values were approximated as [10/16, 1/4, 3/8, 1/4, 10/16] as divisions by powers of 2 are more easily implemented using bit shifts in Verilog. Furthermore, instead of comparing the previous pixel and current pixel, we compared the previous 10th pixel, as smoothing would result in a similar effect to Figure 15 which meant using just the previous pixel would not work. [8]

Figure 15: Smoothing Effect

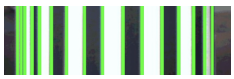

Figure 16: Building Bounding

Wall Detection

Asides from aliens and buildings, the group decided it would be ideal to avoid walls and other obstructions. It was observed that as the camera gets closer to a wall, the average brightness of the image falls as shown in Figure 17.

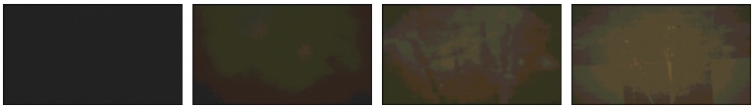

Figure 17: Rover Camera 0, 2, 5, 10 cm from Wall

To measure brightness, we can record luminance which assigns weight to RGB values:

$$luminance = red \times 0.3 + green \times 0.6 + blue \times 0.1$$

It was noticed that in the images in Figure 17 that increasing distance would increase the luminance to 34.1, 45.1, 57.8, and 67.8. These numbers can be tuned to match the distance that the user selects to start avoiding the wall. Presently, the rover will avoid the wall at a 10 cm distance with a 5 cm error. [10]

Distance Estimation

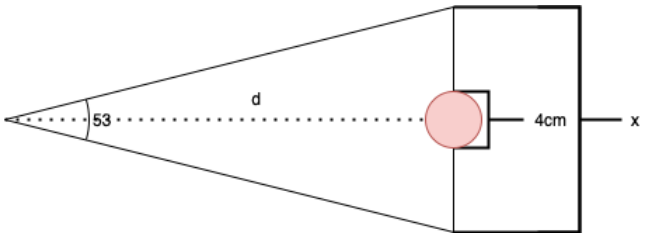

Figure 18: Illustration of distance approximation

$$(1)\ x = \frac{4}{f}$$

$$(2)\ d = \frac{x}{2}/tan\left(\frac{32}{2}\right) = x = \frac{4}{f} = \frac{4}{pixel\ width/640} = \frac{2560}{pixel\ width}$$

Figure 19: Equations for Distance Approximation

It was noticed that we could obtain the distance from the camera to the balls and buildings, as the width of the balls and distances between stripes are known.

First, we had to obtain the camera's field of view angle at the most zoomed-out state. This was done by simply measuring the camera distance to a wall, and then measuring how wide a distance the camera can capture. Measurements of 20cm were recorded when the camera was 18cm away and 30cm was found when the camera was 26cm away. This gave similar angles of 52.4 and 53.5.

Next, we could measure the fraction f of the screen's width that the ball/building stripe takes. This is done by taking $pixel\ width/640$, which is the screen width in pixels.

With this value we could find the value of x in Figure 18 with equation (1), then find d using equation (2) in Figure 19. This gives a good approximation of the distance to the ball when centred. When the ball is detected to be closer to the sides of the screen, using the hypotenuse instead of the adjacent gives a better approximate distance to the ball. The same concept was applied for building distance estimation whose stripe width is slightly different.

Communication to the ESP32

SPI (Serial Peripheral Interface) communication was used between Vision and the ESP32 over UART communication due to its faster data transmission times and higher payload size which is important in real-time image-processing [11]. SPI is characterised by a master/slave interface. We configured the ESP32 as the SPI master and the FPGA as the SPI slave with 4 interacting wires: SCK (serial clock generated by the master), SSEL (slave select), MOSI (Master Output Slave Input) and MISO (Master Input Slave Output). To synchronise these SPI signals, we utilised the FPGA clock observing that this rate is greater than the SPI bus rate. We altered the provided UART infrastructure in Qsys by adding SPI conduits and implemented an SPI module in the pipeline [12] to successfully transmit 16-bit messages from the FPGA to ESP32. The problem of colliding pins with Optical Flow Sensor (which also uses SPI) was solved by utilising both HSPI and VSPI.

The following shows the data format used to express the data being sent from the FPGA to the ESP32:

Function	15 6	5 3	2 0
STOP MOTORS	0	0	0
BALL FOUND	pixel width	ball position	ball colour code
NO BALL FOUND	lumina	NA	000
BUILDING FOUND	pixel width	NA	111

For accuracy (and since we have the luxury of larger bandwidth using SPI), we decided to send 5000 samples of data to the ESP32, and then use the ESP32 to decide which object was being detected, for example, if the ESP32 received many samples of red it indicates with a strong probability that a red ball is being encountered.

The ESP32 was used to estimate the distance based on pixel width, as it can handle divisions much better than Verilog. To obtain the final pixel width from the samples, the mode was used instead of the mean since the data was extremely left-skewed: using the mean resulted in very inconsistent distance approximations as it could be heavily affected with just a few outliers.

One of the encountered limitations was that only 16 bits could be sent at a time, as a result, only 3 bits could be allocated to the ball position which may lead to a slight angle error. A larger limitation is that even SPI has a restricted bandwidth with latency. To account for this and potential unreliable or slow connections between Earth

and Mars it was agreed that all image processing is performed on the hardware creating a more robust system.

4.2.5 Radar

The purpose of this subsystem is to use the HB100 Doppler radar module to detect a rotating fan located underground. A reflected harmonic signal should be received when the rotating fan is under the rover. To design a suitable solution, several performance aspects were considered: maximum detection range of radar, target location accuracy, and the ability to differentiate between the interference and target.

The radar measures reflections not only from the target but also from the surroundings. Sources of interference could include imperfections of the module or other RF transmissions, resulting in clutters. Since the radar output produces a signal with only a maximum 10mV amplitude (low SNR target), the radar wasn't able to differentiate between interference and the signal. Other factors such as DC components and multiple rotating blades also affected the reflected signal.

Clutter Rejection

There are several techniques to mitigate the effects of clutter such as Doppler Shift Separation, Moving Target Indication and Constant False Alarm Rate [13]. Doppler shift of different signals could be used to enhance the detection of moving targets, while different RC filters are used to allow the desired frequency to pass, thus rejecting unwanted signals. Given that a response of 366Hz is produced by the reflector (see Appendix D.1 Figure 43), a 4th order bandpass filter was implemented using 2 Sallen-Key stages with the following specifications: bandwidth 200Hz, centring at 366Hz. Though better filtering could be achieved by narrowing the pass-band, it was difficult to fit onto the rover, and it would result in instability and an increase in time delay for signal processing.

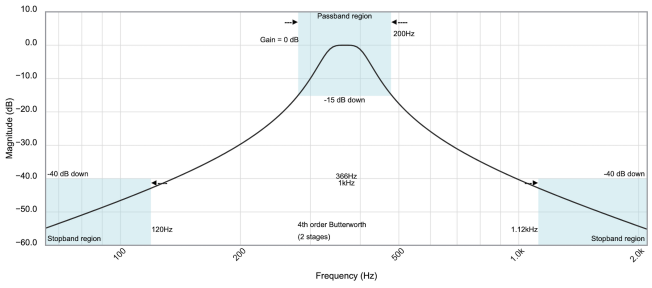

Figure 20: Bandpass Specification

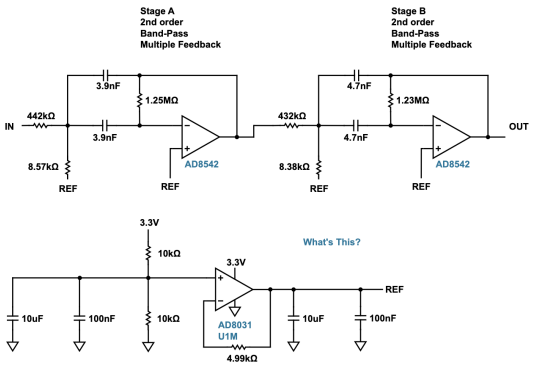

Figure 21: Bandpass Filter Circuit

Online tools were used to design a band-pass filter from the desired specifications [14]. 3.9nF and 4.7nF were chosen for the capacitors that are less than 0.1mF for better frequency stability. MCP6002 was used to substitute the AD8542 shown in Figure 21, since they both have the same GBP and a low input bias current that enables application with high source impedance. Reference voltage was derived to create a mid-supply voltage rail. As the ESP32 ADC analogue input pins are 3.3V tolerant, the supply voltage was then changed from 5V to 3.3V.

Signal Amplification

The target signature has to be amplified when the signal is received so that it could be analysed and detected more easily. By using the radar range equation, it was found that the further the target, the higher the antenna gain. When moving the reflector back and forth under the radar (separated by a wooden table), the range of the target is around 7cm. When it was moved further away, any changes in amplitude would be imperceptible. Although a gain of 100 is sufficient, we increased the magnitude to 210 for better detection.

As shown in Figure 22, the first non-inverting amplifier produces a gain of 21, then it is connected to an inverting amplifier with a gain of -10. The overall gain would be -210. One of the problems we encountered was failing to provide a DC return path for the input bias current. C2 was connected in series with the non-inverting input to AC couple it. This is useful when a high gain is required. However, coupling capacitors with a high-impedance input without a DC return path caused problems. C3 was added to the feedback network to provide a $\times 4$ gain to the AC component, but only a $\times 1$ gain to the DC bias. A simple potential divider wasn't used because any noise from the 3.3V supply would feed directly to the input of the amplifier. Instead, AP431i was used to properly derive the 1.65V DC bias.

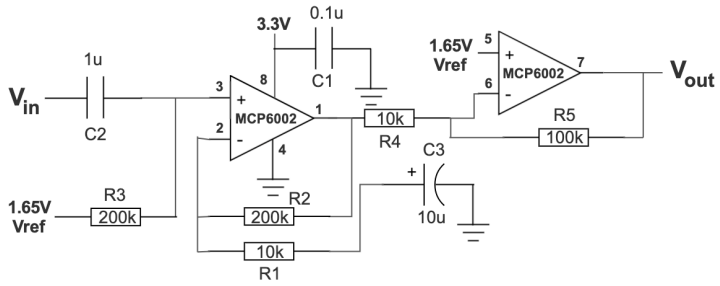

Figure 22: 2-Stages Amplification with a Gain of -210

ESP32 ADC Analog Inputs Reading

After amplifying the signal and filtering the noise, the signal is ready to be detected. A way to read the DC voltage of the received signal from the analogue pin A3 (GPIO 33) was devised [15]. By default, the ESP32 ADC is 12 bits, at which the ADC conversion range is 0 - 4095. The analogue channel produced a digital value between this range according to the voltage at the input of the analogue channel. The implemented ESP32 code (see Appendix D.2) performed DAC by multiplying the resolution of ESP32 ADC with $\frac{3.3}{4095}$ for determining the threshold voltage, which was set to 1.8V through testing. An LED was also connected which lights up when the threshold voltage is reached.

After implementing the voltage reading function, the FFT function was used to transform the signal to the frequency domain accordingly through ESP32. It updates the f_{peaks} array with the top five most dominant frequencies. When the rotating fan is right below the rover, the most dominant frequency should be 366Hz. Even though this resulted in an accurate identification of the required frequency in the signal spectrum, this was a resource-intensive approach. Therefore, ultimately the threshold voltage was used as it more than enough to detect the fan.

4.2.6 Energy

The primary objective of the energy subsystem is to design a solar charging station that is as efficient and effective as possible. It can be dissected into the four major components shown below.

Components	Design Criteria	Description
Solar Panels	Series/Parallel Combinations	4 in parallel
	Operating Characteristic	Characterise individual PV panel (see Appendix E.1)
	Partial Shading	Shading losses may reduce efficiency or damage the PV panels
SMPS	Number and Type of SMPS	2 (First Boost, Second Buck)
	Voltage and Current rating	The voltage-current ratio based on the duty cycles
	Efficiency	Different SMPS arrangements would affect the overall efficiency
Arduino	MPPT algorithm	Determine and obtain peak power point
	Data Collection	SD card reader and SD card are used to log data
	Charge Strategy	4.5V - 5.0V output voltage to the battery based on delivered power
	Communication	Communication and Integration between two SMPS
	Arduino Safety	Two Arduino Adapter boards with potential dividers were used
	Cell Safety	A relay module was used
USB battery pack	Voltage and Current behaviour	Relationships between voltage and current of the battery (see Appendix E.2)
	Rover Range and capability estimation	Estimate the capacity of the battery and rover range

Top Level Design

After considering the limitations of the SMPS (input, output and controllable), 2 SMPS were used in our design. The Boost SMPS utilised the MPPT algorithm to maximise power input from PV panels, while the Buck SMPS regulated the output voltage. This was primarily aimed at minimising power loss. For this design, using 4 PV panels in parallel is most feasible. Additionally, bypass diodes were added to prevent the panels from damaging by partial shading while blocking diodes, which also contributed to 0.5W power losses, were not needed in this configuration. The top-level design of our solar charging station is shown on Figure 23.

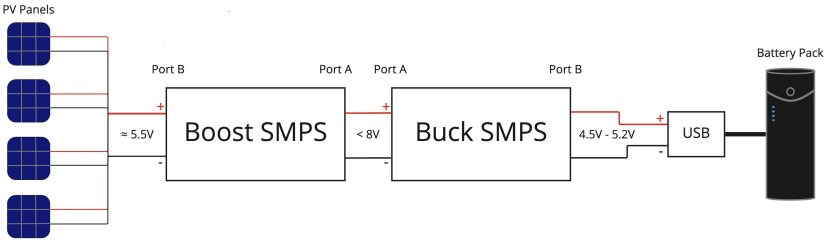

Figure 23: Top-Level Design

Boost MPPT Algorithm

Initially, the 4 parallel PV panels were manually tested under varying sunlight to determine the corresponding current and voltage delivered (see Appendix E.3). The measured maximum voltage and current output were around 5.95V and 1.16A, while a maximum output power of 4.43W was recorded. PV panels produce a non-linear and time-variant power-voltage relationship. Here, a perturb and observe approach was used [16], followed by Arduino code implementation see Figure 24.

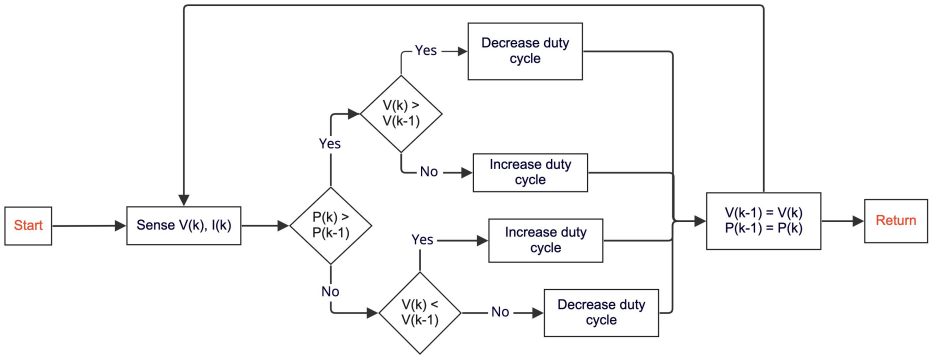

Figure 24: MPPT flow chart

Buck SMPS' Algorithm

Characteristics of the battery were tested (see Appendix E.2) to evaluate the power drawn by the battery at different voltage levels. Also, a relay module [17] with a 0.2V noise margin was added between the second SMPS output and the power bank's input to ensure that only 4.5-5V of the output voltage can flow into the battery. Code had to be modified after the relay module was added since it changed the behaviour of the system. For example, when the relay is switched off, the entire system becomes open-circuited, and no power and current flows into the SMPS.

Approach 1: Output voltage reference

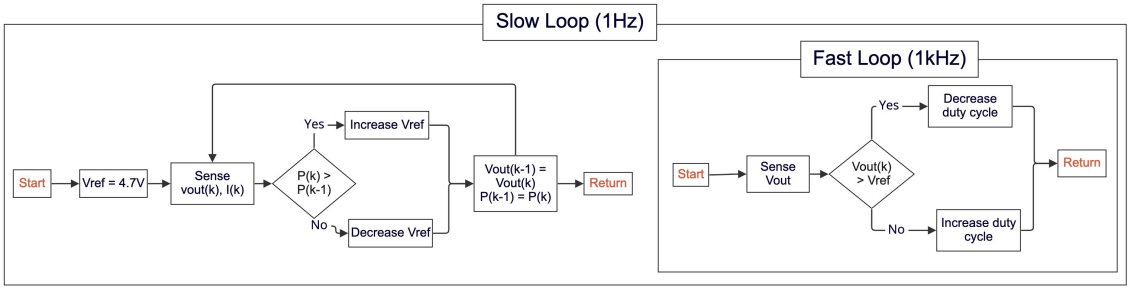

Figure 25: Output voltage regulation algorithm

To regulate the output voltage reference, perturb and observe method was again used following the algorithm in Figure 25. A 4.7V voltage reference was first used and the output voltage would be regulated within the fast loop. Voltage references were changed based on the current and previous power differences. It was implemented in a slow loop to allow the output voltage to converge to the reference voltage.

Approach 2: Input voltage reference
A 7V input voltage reference was set at first and the duty cycle was changed based on the difference between the input voltage and voltage reference. If the input voltage is above 7V, the duty cycle will be set by the formula $duty\ cycle = (v_{in} - v_{ref}) \cdot gain$. As the duty cycle increases, the output voltage will rise and the current will increase as well. Consequently, more power is drawn to satisfy the needs of the MPPT controller. When the input voltage is lower than 7V, no action is required as we do not care about the possibility of a negative current. The saturation function was used to ensure a positive duty cycle only. Changes in the MPPT algorithm were also made to prevent input voltage higher than the 8V limit (7.7V was used in our design).

Extra features
A photoelectric sensor was added to indicate the sunlight conditions. The rated capacity of the power bank (29Wh) and the power consumed by the rover (3.3W) were estimated experimentally. It was then integrated with the Command subsystem to present the battery level and status on our website.

4.2.7 Integration

The purpose of the integration subsystem is to connect all the different subsystems of the project, to ensure the smooth operation of the rover. However, there were also some challenges faced when integrating.

Challenges
The greatest challenge was to ensure the reliable operation of the optical flow sensor. The quality of voltage regulation posed as the largest problem to rover operations. The optical flow sensor would freeze and terminate communication of its measurement values when certain voltage spikes were produced by the change in rotational speed of the motors. This issue was solved by separating its supply from the 5V motor rail, moving it to the 3.3V pin of the ESP32, and thus eliminating excess interference between these two modules.

Since all subsystems were quite independently developed and due to the limited number of pins in the Arduino, we faced a big issue when integrating the Drive, Vision and Radar. There was an overlap of pins of the optical flow sensor and the FPGA. To solve this, there was a change of pin assignments in Arduino IDE and Quartus. VSPI and HSPI also had to be specified for optical flow sensor and Vision respectively.

5 Testing

5.1 Drive

Testing on Forward Movement:
The straight-drive control was evaluated using a set of 10 measurements, including the head angle and the deviation for 3 different distances (0.5m, 1m, and 3m), and 3 different speeds (7cm/s, 9cm/s and 11cm/s) respectively. Through testing, several parameters were adjusted to corroborate the straightness during forward drive.

Testing on Rotation:
When testing the accuracy of the rover's rotational movement, the rover would rotate in θ of 45^o, which is fixed on the frontend web application. 15 trials were done to ensure that the error margin was within 0.5^o.

5.2 Vision

Testing on Colour and Building Detection Accuracy:
We tested our colour and building detection by randomly showing different balls or buildings and calculating how often we detect the correct object for different lighting conditions and distances on 15 different spots on the Mars arena. Figure 26 shows how often out of 15 trials we detect the correct object.

Testing on Distance Approximation:
Since we approximated the distance of balls closer to the edges of the screen, and balls closer to the centre differently, we measured actual and approximated distances for both. For testing, we choose to measure distances within the range of 15 cm to 60 cm as anything below 15 cm may be in the rover's blind spot and anything above 60 cm would be too hard to distinguish the ball and specks of noise (due to how our bounding algorithm works). Furthermore, we tested balls in the centre and on the edges as they were calculated differently. See Figure 26.

Testing Speed of SPI Communication:
We chose SPI as it was significantly faster than UART. As a result, we can send more samples of coloured data in the same amount of time which greatly improved our distance, colour and building detection. For each loop of our process, we receive 5000×16 bits of data, this process took approximately 0.6 seconds using SPI and 4 seconds using UART, making us have to reduce the number of samples taken if UART was used.

Object and Lighting	Accuracy		
	10cm	25cm	40cm
Red Low Lighting	100%	100%	100%
Red Good Lighting	100%	100%	100%
Pink Low Lighting	100%	100%	100%
Pink Good Lighting	100%	100%	100%
Dark Green Low Lighting	87%	80%	73%
Dark Green Good Lighting	100%	100%	87%
Green Low Lighting	100%	100%	100%
Green Good Lighting	100%	100%	100%
Blue Low Lighting	100%	87%	67%
Blue Good Lighting	100%	100%	80%
Yellow Low Lighting	100%	100%	100%
Yellow Good Lighting	100%	100%	100%
Building Low Lighting	100%	87%	73%
Building Good Lighting	100%	100%	93%

Ball Colour and Position	Actual Distance (cm)			Calculated Distance (cm)			Error Margin (%)
Red Ball Edge	15	30	60	14.1	32.4	55.2	7.5
Red Ball Center	15	30	60	15.3	28.8	63.0	3.6
Pink Ball Edge	15	30	60	15.8	28.2	65.4	6.6
Pink Ball Center	15	30	60	14.9	28.8	61.2	2.3
Dark Green Ball Edge	15	30	60	11.6	27.0	72.0	19
Dark Green Ball Center	15	30	60	18.0	24.6	57.6	14
Green Ball Edge	15	30	60	13.2	33.9	72.0	13.9
Green Ball Center	15	30	60	14.0	28.5	55.2	7
Blue Ball Edge	15	30	60	10.7	36.0	70.8	24
Blue Ball Center	15	30	60	18.0	26.1	63.0	12
Yellow Ball Edge	15	30	60	16.1	32.1	66.0	7.5
Yellow Ball Center	15	30	60	16.2	28.5	56.4	6.4
Building	15	30	60	14.7	27.9	46.8	12.6

Figure 26: Test Results for Object Detection and Distance Approximation

5.3 Radar

Testing on the Detection Range on the bench:
When the radar was mounted above the fan, the signal oscillated between 1.1-2.1V. The maximum range of oscillation was 0.3-2.9V(see Appendix D.3), the threshold voltage was therefore set to 2V for initial testing.

Testing on the Detection Range in the arena:
Given that the fan is placed near a corner of the arena, the following tests were conducted to maximise the accuracy of mapping the fan. During testing, the rover was instructed to move towards the corner from different directions, brake when a fan was detected and send current location to Command. The threshold voltage and the inclined angle of the radar module were adjusted through sets of measurements of the displacement of the radar from the fan shown in Figure 27. The final inclined angle and threshold voltage are now fixed at 50º and 1.8V.

Inclined Angle (°)	50		
Threshold Voltage(V)	1.8		
	Length(cm)	Width(cm)	Displacement from the fan
Fan Location from the corner	35	25	0
1st Attempt	45	33	12.79
2nd Attempt	45	25	8.46
3rd Attempt	40	30	11.07
4th Attempt	45	24	7.99

Figure 27: One set of measurements for Detection Range

5.4 Solar Charging Station

MPPT Algorithm
Multiple resistor values were connected at the end of the Boost SMPS to determine whether maximum power could be obtained (see Appendix E.4). For resistors of 10Ω, 75Ω and 120Ω, the duty cycle is increased to provide more power and this saturates when power is maximum. The maximum input power we can obtain was around 3.1W and one of the panels was found to be defective. The MPPT did not work when the resistor values were out of range because the duty cycle was saturated and power was capped.

Overall Performance

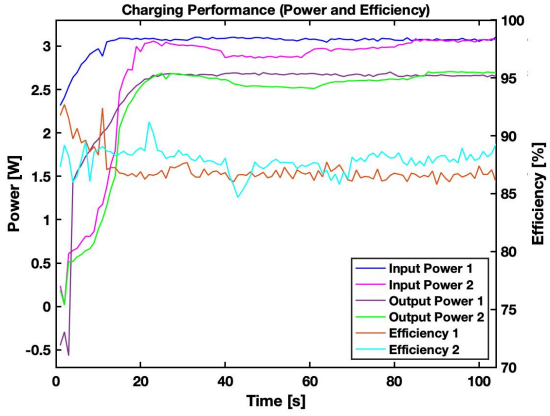

Figure 28: Output Power and Efficiency

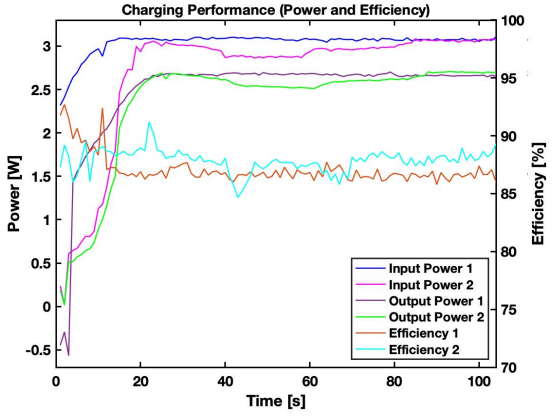

Figure 29: Output voltage and current

Extensive Testing on both approaches was performed under the sunlight. Results were stored on the SD card and processed in Matlab. As seen in Figure 28, both approaches provided an average power of 2.7W to the battery. Also, the efficiency in approach 2 (87.8%) was slightly higher than that of approach 1 (86.9%). In the beginning, the output voltage and current kept increasing until the maximum power point. In Figure 29, we can see that the output voltages and current in both approaches were saturated at the same level and solar charging stations worked in both approaches. Although the second approach has higher efficiency, it's less stable than the first one as plugging in the cable multiple times was needed occasionally to start charging the battery. The battery was estimated to take 10 hours to be fully charged on a sunny day.

5.5 Command, Control and Integration

A comprehensive testing strategy was employed to ensure the reliability of our rover while also ensuring effective integration.Tests were conducted in 3 stages including Command, Control and Integration Testing:

Command Side Testing:
1. Construct 20+ unit tests in Python for the matrix rotations on a fixed set of coordinates. This will ensure that at each iteration, the tile will be successfully mapped.
2. Create a dummy client to interact with the main server and simulate rover exploration on a reduced map. This should ascertain whether aliens could be inserted into the database and assess server functionality.
3. Test rover movement on 30+ simulated full maps produced randomly by a Python script. This is done by feeding the relevant data into the client and using the server to facilitate discovery (see Figures 37-41).

ESP32 Client Side:
1. Develop a WiFi transmission protocol to transmit data across to a dummy server.
2. Test and ensure other subsystems can integrate effectively on the ESP32 side.

Integration Testing:
1. Assess module integration by testing a stationary rover and observing how the map changes after simulation.
2. Observe the rover movement in manual and automatic mode on a reduced map with only 1-2 aliens present.
3. Test the autonomous and manual discovery modes on the whole scale arena. Passing this test would ensure that the system is ready for deployment.

Through testing, several errors in Command and Control were resolved. On the Command side, these included being unable to select the relevant tiles around the rover and solving website crashes when the rover reached the edge of the screen. Additionally, in Control, errors related to WiFi transmission protocol could be debugged.

6 Conclusion and Recommendations for Future Work

In conclusion, after an extensive software development process, all subsystems of the rover effectively fit together. Through assembling the Drive subsection and tuning the controller, the rover can be configured to drive in a straight and precise manner. Additionally coordinates of the rover can also be recorded by this module. Meanwhile, on the Vision subsystem balls can be accurately located and mapped by the camera with other additional features such as wall-detection. On Command, a comprehensive and clear system is used to place and locate aliens based on the data stream provided by the clients. On Energy, two models were implemented and tested throughout the project, both models can charge the battery with relatively high efficiency. However, further improvements can be made to adapt to the Mars environment and increase stability. When combining these modules, there is relatively good integration across all subsystems demonstrating the efficacy of the rover developed. Ultimately, this suggests how the rover is designed brilliantly and capable of exploring different locations and territories.

We have identified several areas to further develop our current design, and thus improve the speed and accuracy of our rover. By using error compared to a defined path, as opposed to a defined head angle, the forward movement PID controller can be developed for improved robustness. For the Command subsystem, Huffman Coding can be applied when sending messages from the client to the server. This is because the source alphabet is a non-uniform distribution as most of the map is empty terrain and the rover identifies this most often. Therefore, by utilising Huffman coding we reduce source entropy and data storage requirements which is important in remote environments. For the Energy subsystem, our design could be further stabilised. We believe that it is possible to develop the energy system in one converter (Ćuk), as well as to increase efficiency. For the Vision subsystem, we could implement more complex classifiers using neural networks which could detect balls and other objects better. Additionally, testing environment could have been altered to match the environment, weather conditions, or extreme cases on Mars, such as dust storms and solar flares.

References

[1] apirobot, "What is wsgi and why do you need gunicorn and nginx in django," Available at $https://apirobot.me/posts/what-is-wsgi-and-why-do-you-need-gunicorn-and-nginx-in-django$ (2021/05/22) [Date Accessed: 2022/06/14].

[2] Firewall.cx, "Osi layer 5 - session layer," Available at $https://www.firewall.cx/networking-topics/the-osi-model/176-osi-layer5.html$ [Date Accessed: 2022/06/18].

[3] B. Douglas and MATLAB, "Understanding pid control," Available at $https://www.youtube.com/watch?v=wkfEZmsQqiA$ (2018/05/22) [Date Accessed: 2022/06/10].

[4] SlideToDoc, "Pid controller," Available at $https://slidetodoc.com/pid-controller-different-types-of-feedback-control-onoff/$ [Date Accessed 2022/06/20].

[5] OpenCV, "Color conversions," Available at $https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html$ [Date Accessed: 2022/05/22].

[6] A. Marr, "What causes dark corners in photos? — how to fix vignetting," Available at $https://explorelandscapephotography.com/what-causes-dark-corners-in-photos-how-to-fix-vignetting/$ [Date Accessed: 2022/06/15].

[7] MathWorks, "Colorthesholder," Available at $https://www.mathworks.com/help/images/ref/colorthresholder-app.html$ [Date Accessed: 2022/05/25].

[8] A. W. R. Fisher, S. Perkins and E. Wolfart, "Gaussian smoothing 2003," Available at $https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm.$ [Date Accessed: 2022/05/30].

[9] S. A. Edwards, "Qsys and ip core integration, spring 2020," Available at $http://www.cs.columbia.edu/sedwards/classes/2020/4840-spring/qsys.pdf$ [Date Accessed: 2022/06/05].

[10] C. in Color, "Luminosity color," Available at $https://www.cambridgeincolour.com/tutorials/histograms2.htm$ [Date Accessed: 2022/05/30].

[11] R. W. World, "Uart vs spi vs i2c — difference between uart, spi and i2c 2012," Available at $https://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html$ [Date Accessed: 2022/06/07].

[12] fpga4fun, "Spi 2 - a simple implementation," Available at $https://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html$ [Date Accessed: 2022/06/10].

[13] R. Technologies, "Radar clutter reduction techniques for warfighters," Available at $https://radausa.com/blog/radar-clutter-reduction-techniques$(2021/05/05) [Date Accessed: 2022/06/15].

[14] A. Devices, "Analog filter wizard," Available at $https://tools.analog.com/en/filterwizard/$ [Date Accessed: 2022/06/15.

[15] R. N. Tutorial, "Esp32 adc – read analog values with arduino ide," Available at $https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/$[]Date Accessed: 2022/06/10].

[16] Hackaday.io, "Overview of mppt algorithm modelling — details —," Available at $https://hackaday.io/project/4613-arduino-mppt-solar-charge-controller/log/24093-overview-of-mppt-algorithm-modelling/$ [Date Accessed: 2022/05/25].

[17] L. M. Engineers, "In-depth: Interface two channel relay module with arduino," Available at $https://lastminuteengineers.com/two-channel-relay-module-arduino-tutorial/$ [Date Accessed: 2022/06/13].

Appendices

A Team Work and Project Management

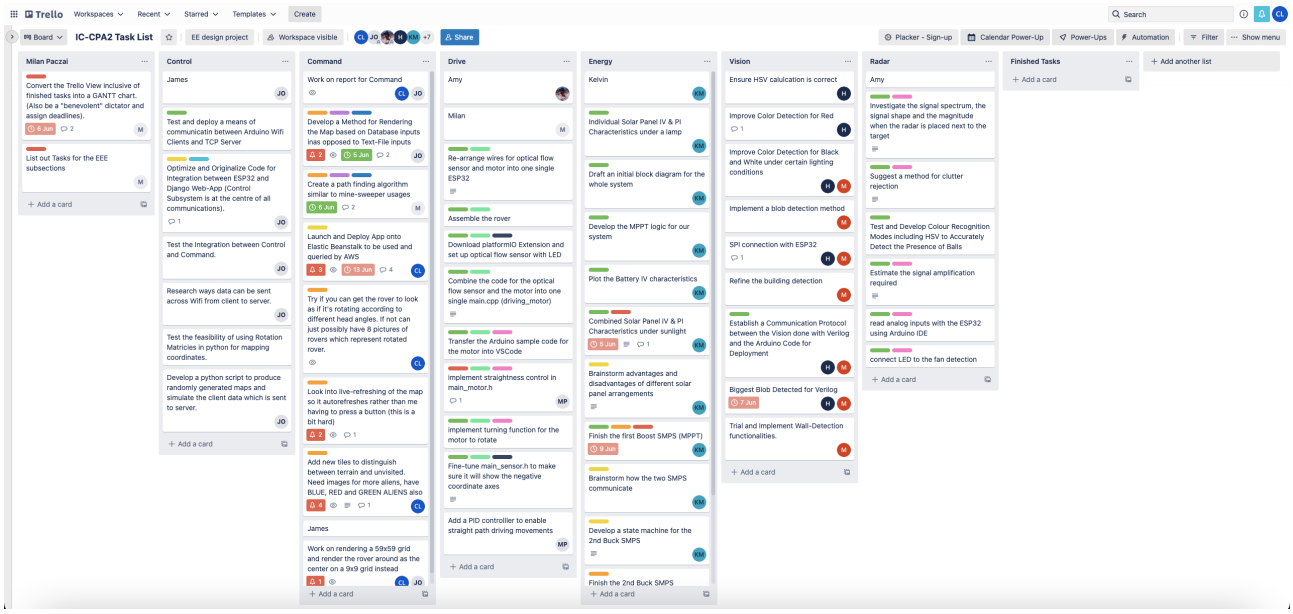

Figure 30: Trello Dashboard for Task Coordination

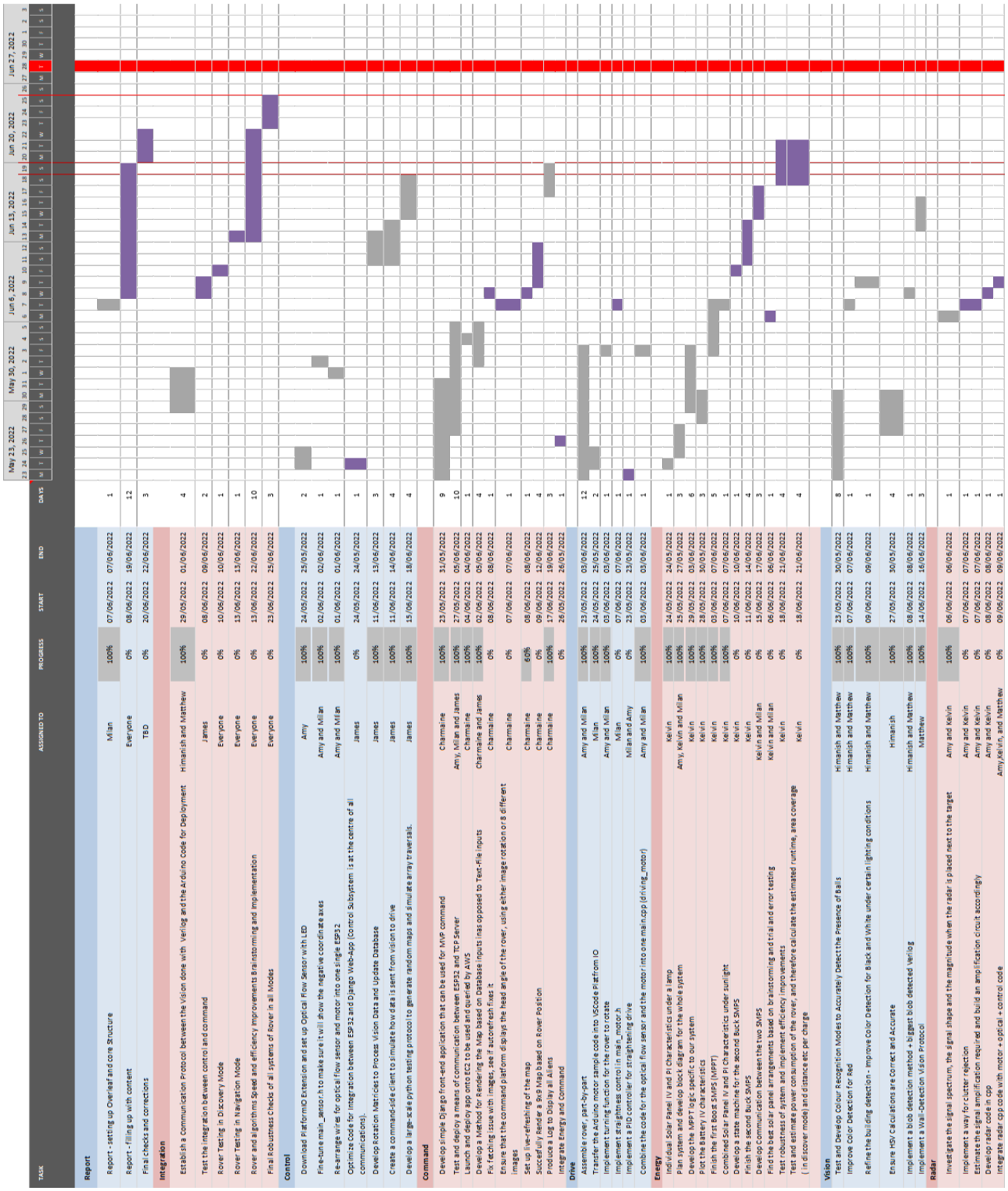

Figure 31: GANTT Chart to Show List of All Tasks

B Command

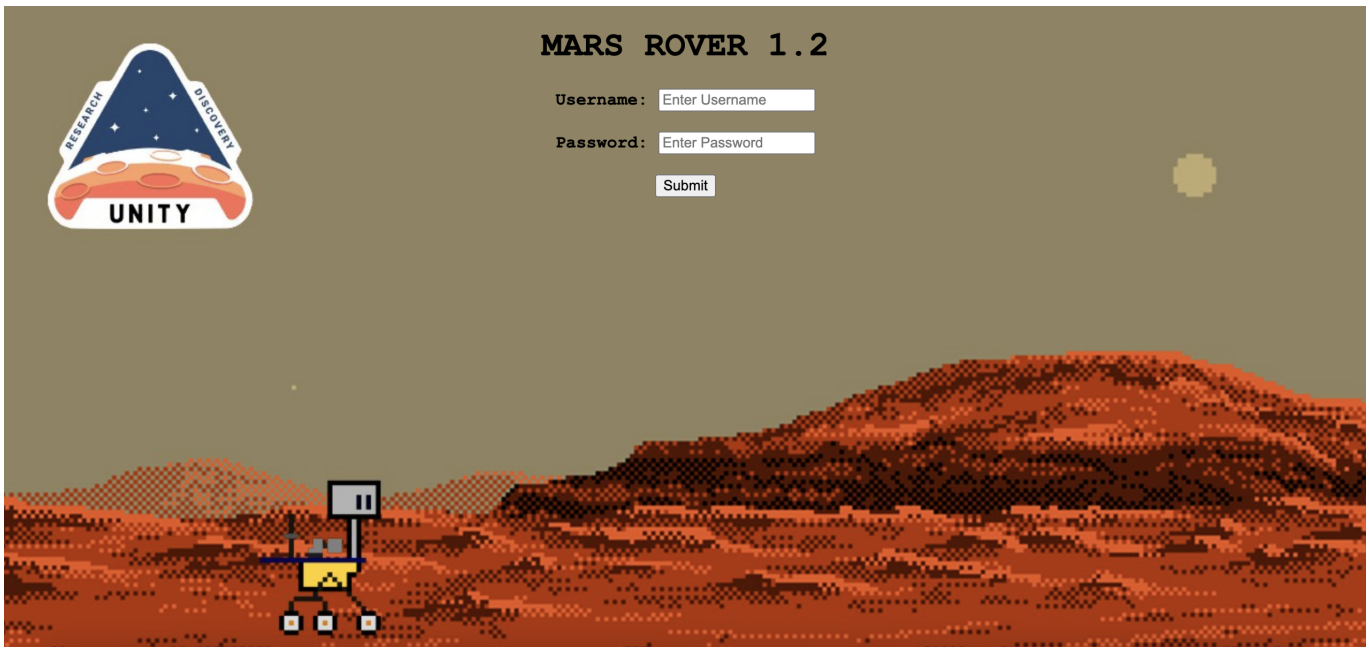

Figure 32: Login Screen

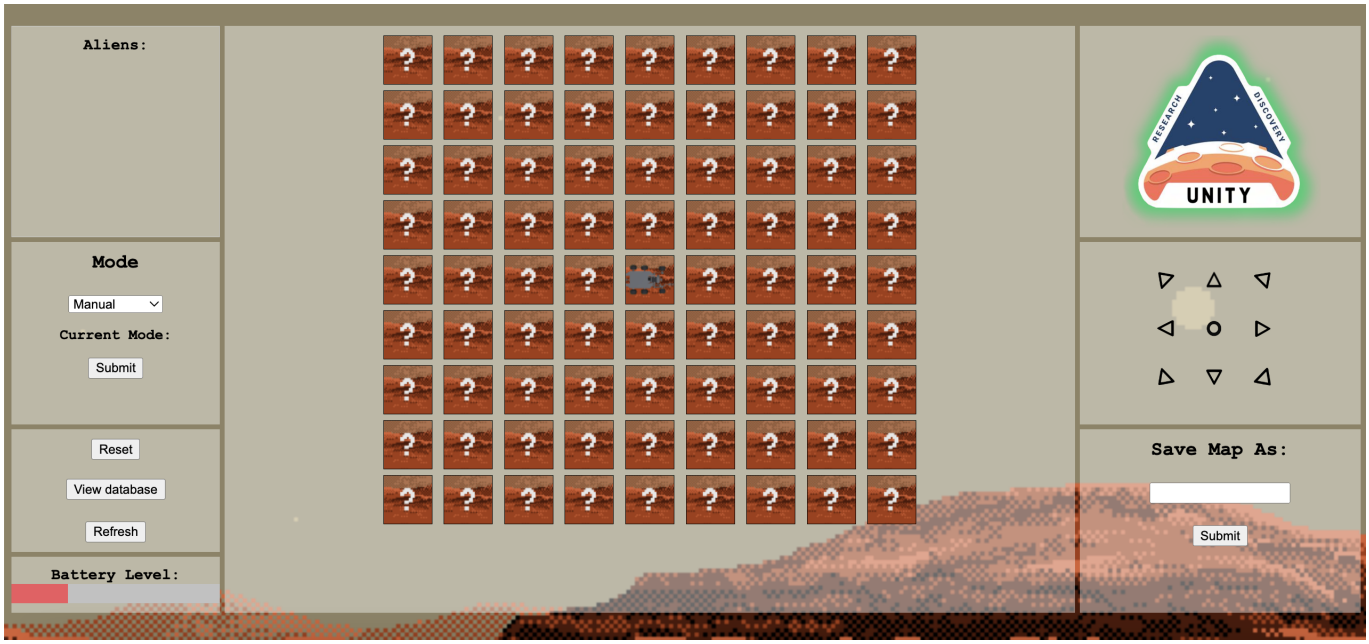

Figure 33: Main Screen

Figure 34: Database Query

Figure 35: Previous Map Display

C Control

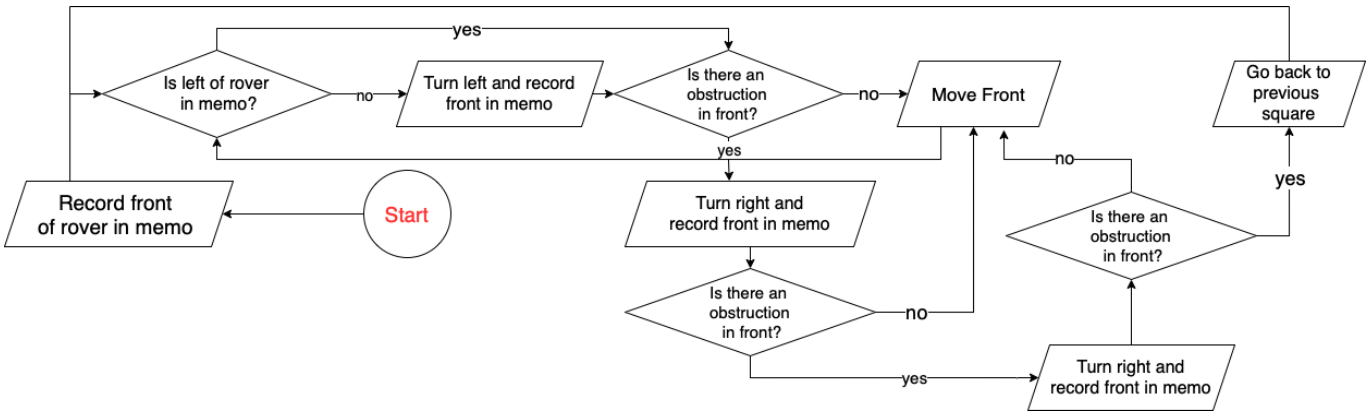

Figure 36: Search Move Flow

```python
import random

def generate_array():
    #this needs to generate a 20x16 array approximately to map all rover positions
    assigned_array = [['T' for col in range(20)] for row in range(16)]
    return assigned_array

def assign_aliens():
    alien_dict = {}
    pink_ali_x = random.randint(0,20)
    pink_ali_y = random.randint(0,15)
    store_pink = str(pink_ali_x)+","+str(pink_ali_y)
    alien_dict[store_pink] = "PA"

    red_ali_x = random.randint(0,20)
    red_ali_y = random.randint(0,15)
    store_red = str(red_ali_x)+","+str(red_ali_y)
    while store_red == store_pink:
        red_ali_x = random.randint(1,20)
        red_ali_y = random.randint(1,15)
        store_red = str(red_ali_x)+","+str(red_ali_y)
    alien_dict[store_red] = "RA"


    ba_x = random.randint(0,20)
    ba_y = random.randint(0,15)
    store_bl = str(ba_x)+","+str(ba_y)
    while (store_bl == store_pink or store_bl == store_red):
        ba_x = random.randint(1,20)
        ba_y = random.randint(1,15)
        store_bl = str(ba_x)+","+str(ba_y)
    alien_dict[store_bl] = "BA"

    gra_x = random.randint(0,20)
    gra_y = random.randint(0,15)
    store_gra = str(gra_x)+","+str(gra_y)
    while (store_gra== store_pink or store_gra == store_red or store_gra == store_bl):
        gra_x = random.randint(1,20)
        gra_y = random.randint(1,15)
        store_gra = str(gra_x)+","+str(gra_y)
    alien_dict[store_gra] = "GA"

    ora_x = random.randint(0,20)
    ora_y = random.randint(0,15)
    store_ora = str(ora_x)+","+str(ora_y)
    while (store_ora== store_pink or store_ora == store_red or store_ora == store_bl or store_ora==store_gra):
        ora_x = random.randint(1,20)
        ora_y = random.randint(1,15)
        store_ora = str(ora_x)+","+str(ora_y)
    alien_dict[store_ora] = "OA"
    return alien_dict
```

Figure 37: Code to Generate a Random Map

```python
def render_vals(param,inp_arr):
    print(param)
    for i in param:
        buffer = i.split(",")
        print("bugger",buffer)
        print(param[i])
        inp_arr[int(buffer[1])][int(buffer[0])] = param[i]
        print(inp_arr)
    return inp_arr

def pretty_print(vals_arr,dict):
    counter = 40
    file_opener = open("temp_data.txt","w")
    file_opener.write(str(dict)+"\n")
    for row in vals_arr:
        print(counter,":",row)
        file_opener.write(str(counter)+":"+str(row)+"\n")
        counter+=1
    file_opener.close()




tmp = assign_aliens()
new_arr = generate_array()
make_vals = render_vals(tmp,new_arr)
pretty_print(make_vals,tmp)
```

Figure 38: Code to Generate a Random Map 2

```
{'14,10': 'PA', '6,9': 'RA', '1,12': '  ', '5,12': 'GA', '17,1': 'YA'}
40:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
41:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'YA', 'T']
42:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
43:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
44:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
45:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
46:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
47:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
48:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
49:['T', 'T', 'T', 'T', 'T', 'T', 'RA', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
50:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'PA', 'T', 'T', 'T', 'T']
51:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
52:['T', 'BA', 'T', 'T', 'T', 'GA', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
53:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
54:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
55:['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']
```

Figure 39: Random Output Map from Code

```python
stored_things = {'14,10': 'PA', '6,9': 'RA', '1,12': 'BA', '5,12': 'GA', '17,1': 'YA'}
assigned_array = [['T' for col in range(20)] for row in range(16)]


def render_vals(param,inp_arr):
    print(param)
    for i in param:
        buffer = i.split(",")
        inp_arr[int(buffer[1])][int(buffer[0])] = param[i]
    return inp_arr

testing_store = render_vals(stored_things,assigned_array)

def traverse_row_90(inp_arr,row_num):
    append_vals = open("angles.txt","a")
    to_ret = "18,"+str(row_num)+";U1;Ux;Ux;Tx;90"
    to_ret2 = "19,"+str(row_num)+";U1;Ux;Ux;Ux;90"

    for i in range(0,18):
        front = inp_arr[row_num][i+1]
        two_up = inp_arr[row_num][i+2]
        two_right = inp_arr[row_num+1][i+2]
        coords = str(i)+","+str(row_num)
        to_return = coords+";"+"U1;"+two_up+"x;"+two_right+"x;"+front+"x;"+"90"


        if row_num!=0:
            top_left = inp_arr[row_num-1][i+2]
            to_return = coords+";"+top_left+"x;"+two_up+"x;"+two_right+"x;"+front+"x;"+"90"
        print("Line 33:"+to_return)
        append_vals.write(to_return+"\n")#this return
    append_vals.write(to_ret+"\n")
    print("36:"+to_ret+"\n")
    print("36:"+to_ret2+"\n")
    append_vals.write(to_ret2+"\n")
    return "19,"+str(row_num)
```

Figure 40: Python Script to Simulate Random Traversal

angles.txt

```
0,0;U1;Tx;Tx;Tx;90
1,0;U1;Tx;Tx;Tx;90
2,0;U1;Tx;Tx;Tx;90
3,0;U1;Tx;Tx;Tx;90
4,0;U1;Tx;Tx;Tx;90
5,0;U1;Tx;Tx;Tx;90
6,0;U1;Tx;Tx;Tx;90
7,0;U1;Tx;Tx;Tx;90
8,0;U1;Tx;Tx;Tx;90
9,0;U1;Tx;Tx;Tx;90
10,0;U1;Tx;Tx;Tx;90
11,0;U1;Tx;Tx;Tx;90
12,0;U1;Tx;Tx;Tx;90
13,0;U1;Tx;Tx;Tx;90
14,0;U1;Tx;Tx;Tx;90
15,0;U1;Tx;YAx;Tx;90
16,0;U1;Tx;Tx;Tx;90
17,0;U1;Tx;Tx;Tx;90
18,0;U1;Ux;Ux;Tx;90
19,0;U1;Ux;Ux;Ux;90
19,0;Ta;Ta;U1;Ta;180
19,1;Ta;Ta;U1;Ta;180
19,2;Ta;Ta;U1;Ta;180
19,3;Ta;Ta;U1;Ta;180
19,3;T1;T1;T1;T1;270
18,3;T1;T1;T1;T1;270
17,3;T1;T1;T1;T1;270
16,3;T1;T1;T1;T1;270
15,3;T1;T1;T1;T1;270
14,3;T1;T1;T1;T1;270
13,3;T1;T1;T1;T1;270
12,3;T1;T1;T1;T1;270
```

Figure 41: Simulated Random Traversal Text File Sent from Client to Server

D Radar

D.1 Final Signal Shape and Spectrum

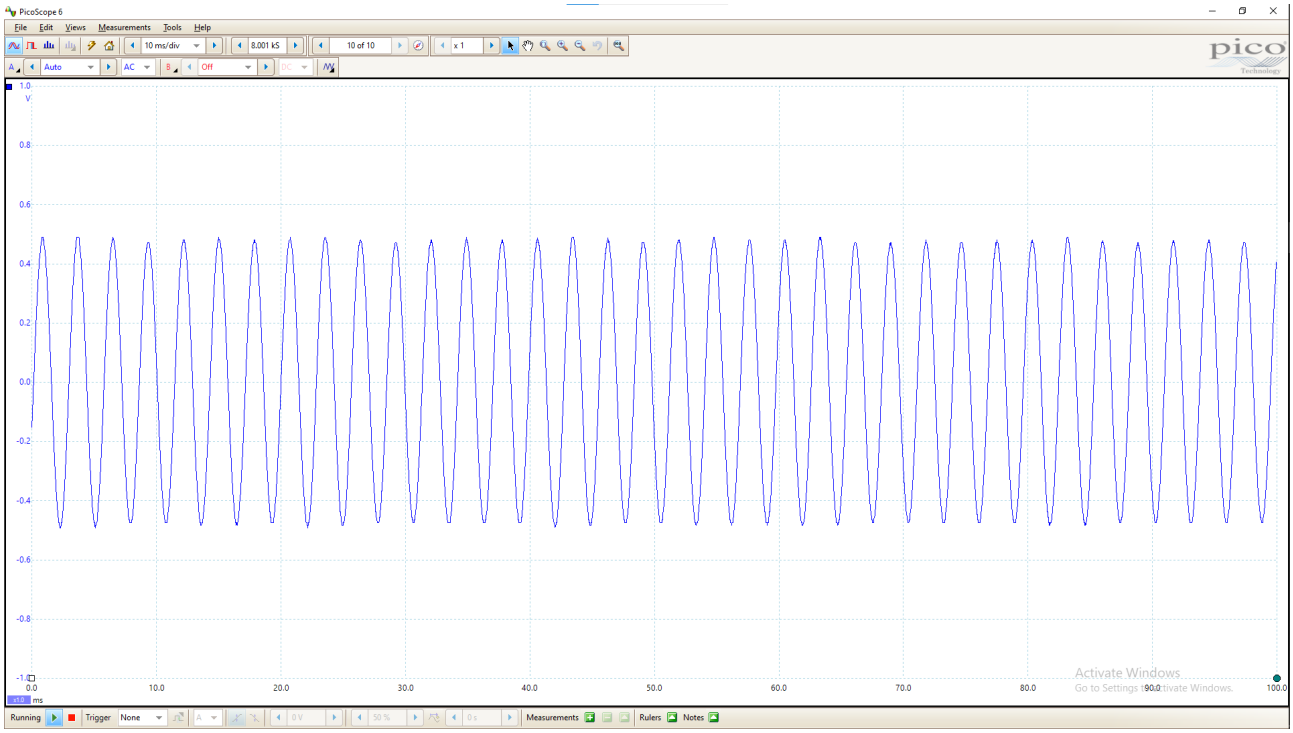

Figure 42: Time-domain graph when a fan is detected

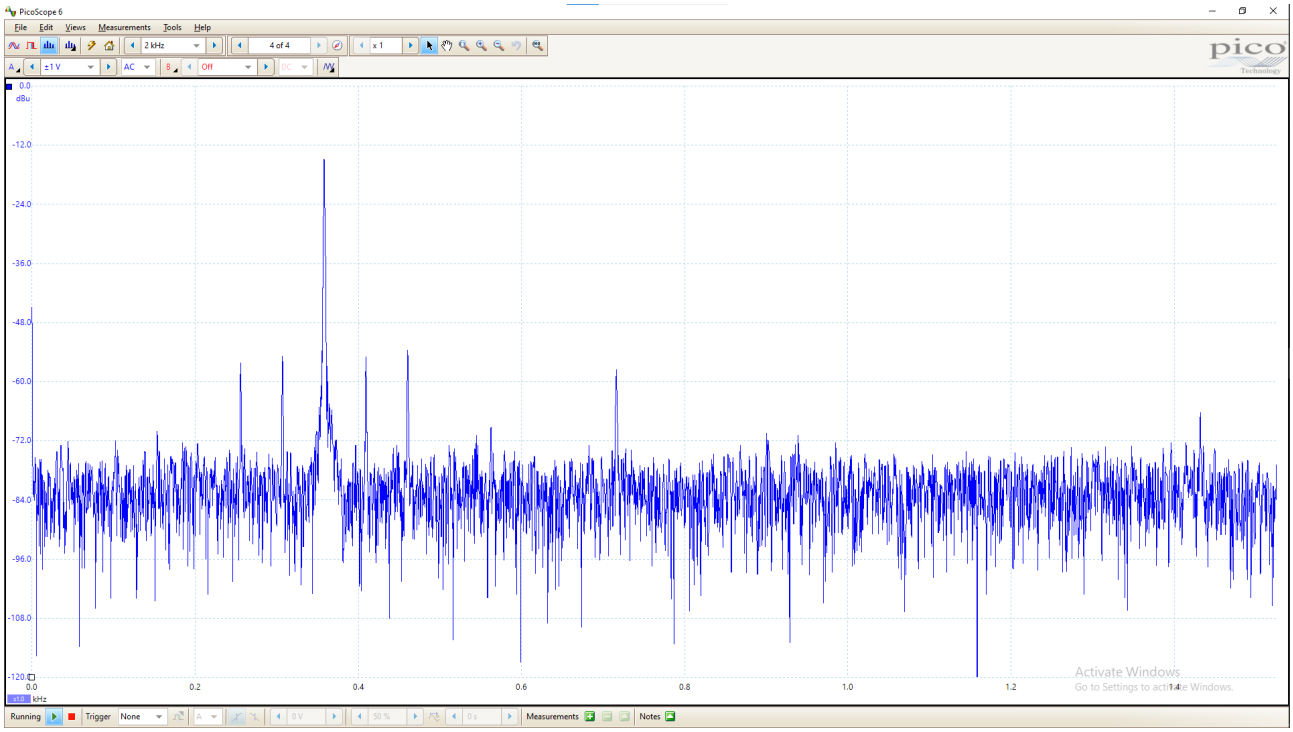

Figure 43: Frequency-domain graph when a fan is detected

D.2 File <Radar.h> for Integration:

```
#include <Arduino.h>
#include <Wire.h>
```

```
class Radar{
//analog read
private:
    const int Analog_channel_pin = 33;
    const byte LED_GPIO = 32;

public:
    int ADC_VALUE = 0;
    float voltage_value = 0;

void setup() {
    Serial.begin(9500);
// initialize analog pin A3 as an input.
    pinMode(A3,INPUT);
// initialize digital pin LED_BUILTIN as an output.
    pinMode(LED_GPIO, OUTPUT);
}

bool fan_detect(){
    for (int i = 0; i < 50; i++){
      ADC_VALUE = analogRead(Analog_channel_pin);
      voltage_value = (ADC_VALUE * 3.3 ) / (4095);

        if (voltage_value > 1.8 ){
          digitalWrite(LED_GPIO, HIGH);
          Serial.println("fan detected!");
          return true;
          break;
        }

        else{
          digitalWrite(LED_GPIO, LOW);
          Serial.println("no fan");
          return false;
        }
    }
    return false;
  }
};
```

D.3 Final Simulation

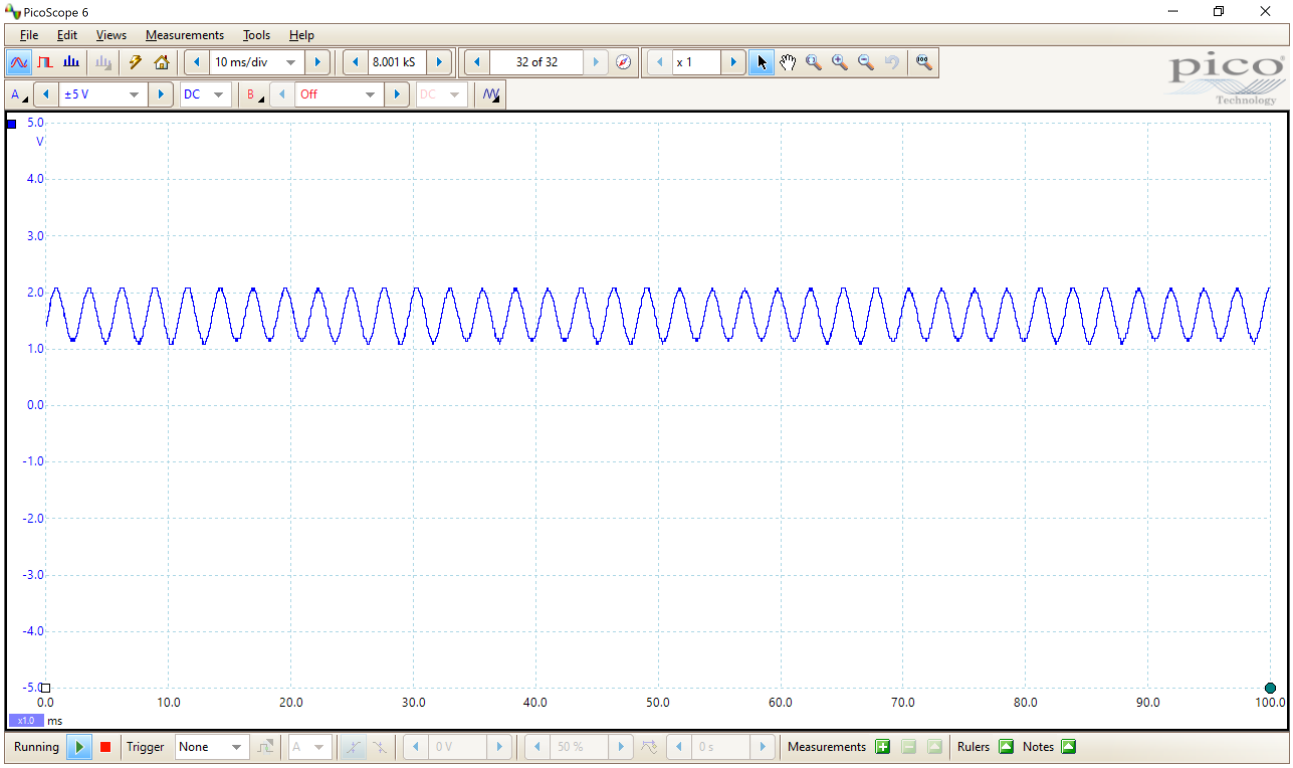

Figure 44: When the fan is within the detection range

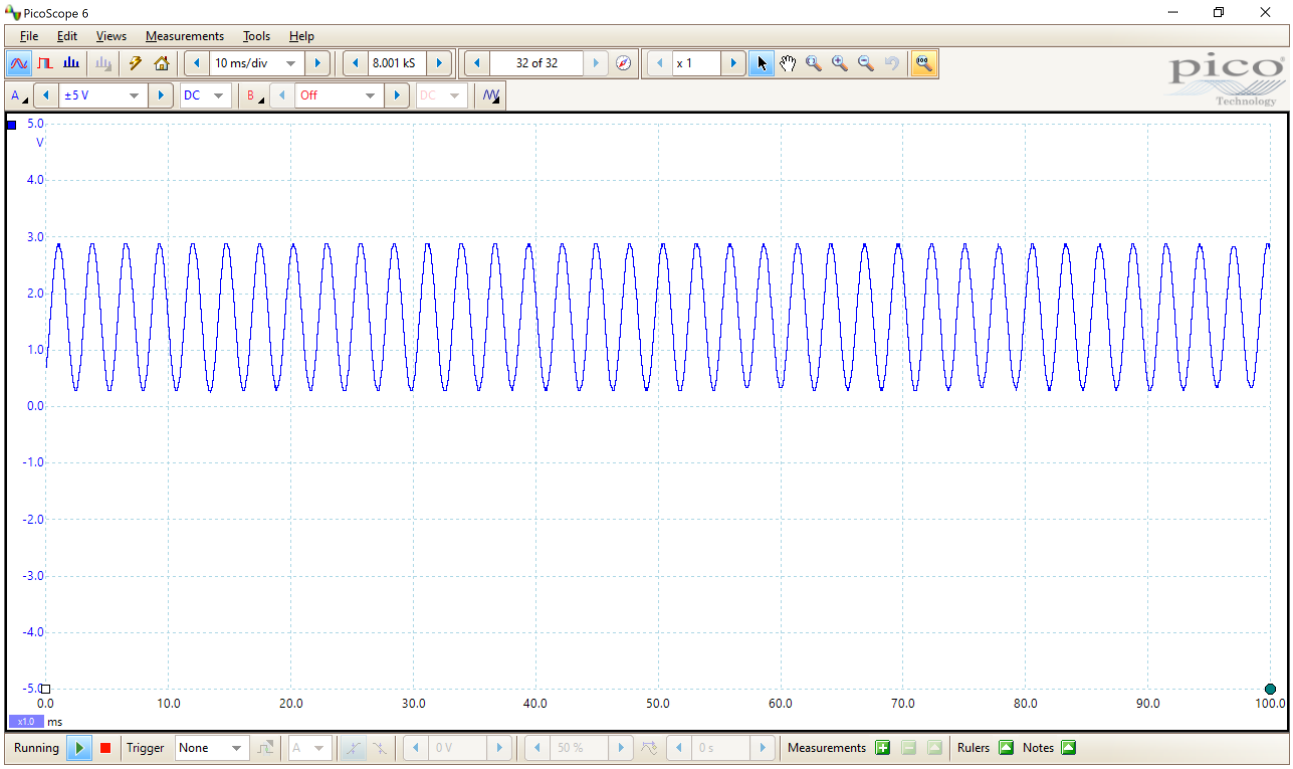

Figure 45: The maximum range when the fan is within the detection range

E Energy

E.1 Individual Solar Panel Characteristic

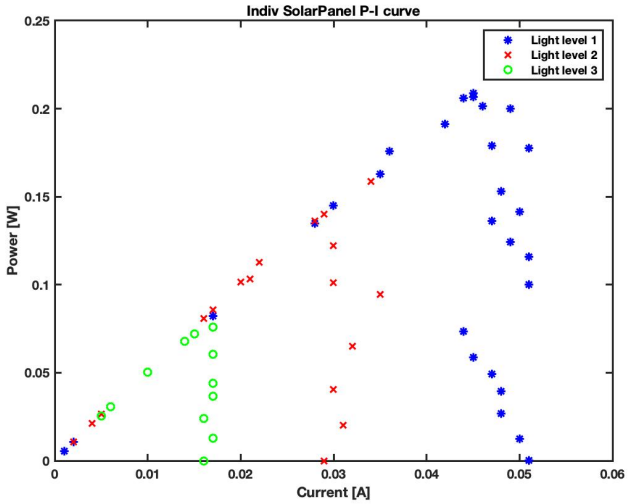

Figure 46: Individual Solar Panel PI characteristic

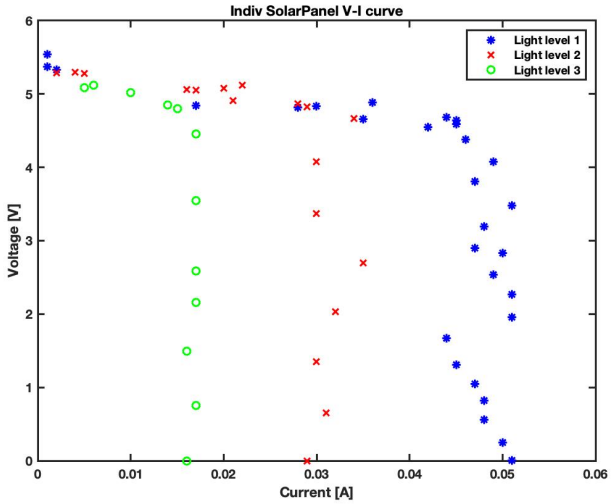

Figure 47: Individual Solar Panel VI characteristic

E.2 Battery Pack Characteristic

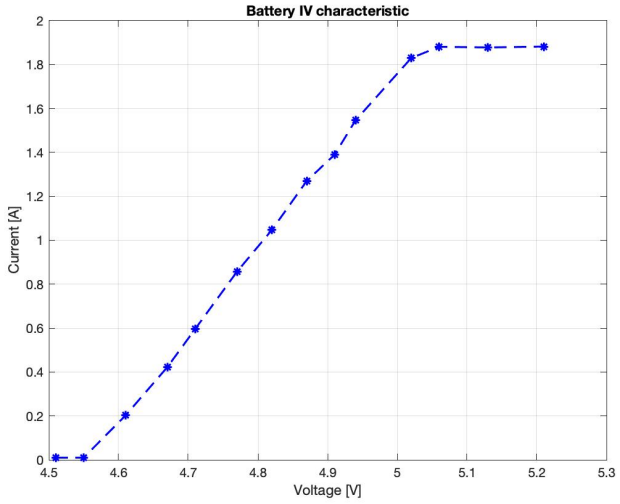

Figure 48: Current and Voltage Relationship

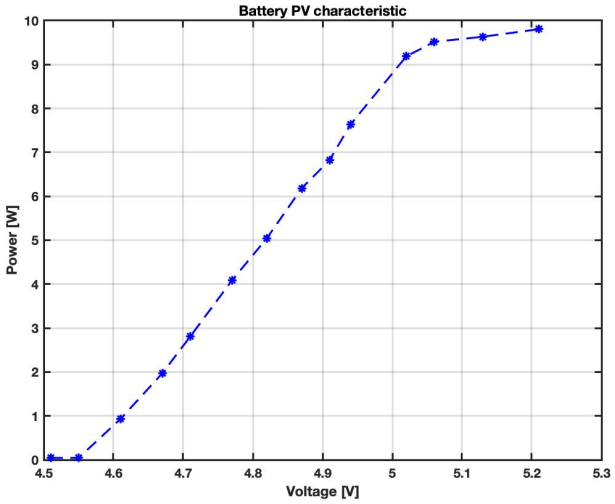

Figure 49: Power and Voltage relationship

E.3 4-in-parallel Solar Panels Characteristic

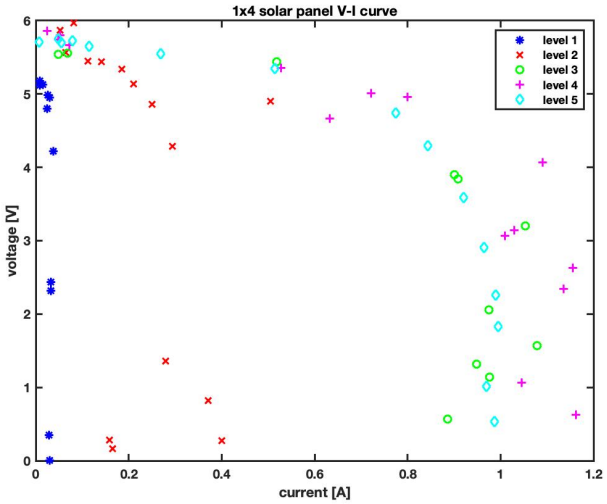

Figure 50: Current and Voltage Relationship

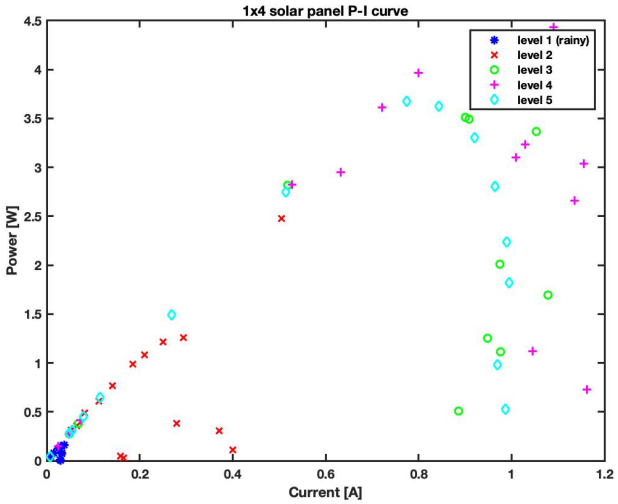

Figure 51: Power and Voltage relationship

E.4 MPPT Testing

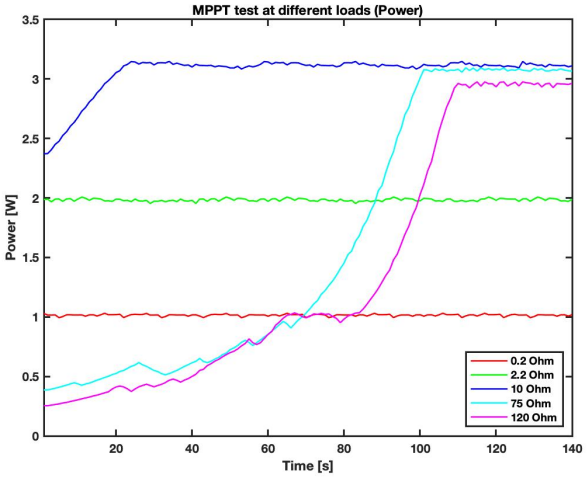

Figure 52: MPPT Power tracking

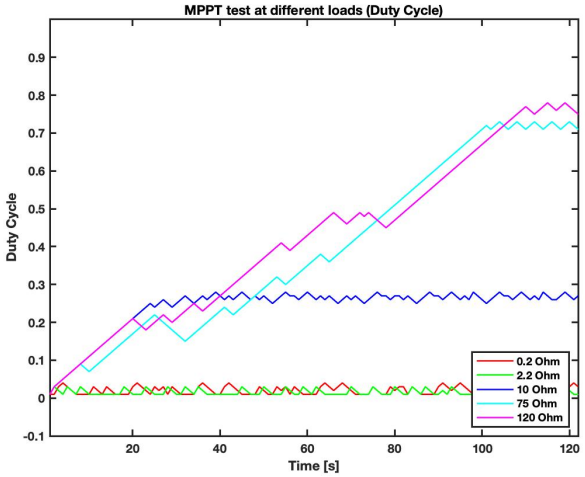

Figure 53: MPPT Duty Cycle tracking

F Integration