

MARS SQUAD

2nd Year EEE/EIE Group Project Report

ELEC50008 – Engineering Design Project 2

Rohan Gandhi (01845920), Aleera Ewan (01857182), Valia Giannopoulou (01933859), Omar Zeidan (01854974), Anthony Jones (01857797), James McManus (01870085), Sam Hesketh Fatchen (01861697)

Title and the name of the lecturer the work is submitted to: Adam Bouchaala

Github link: <https://github.com/ozeidan9/mars-rover>

Abstract

“This project aims to design and build an autonomous rover system for exploring an alien colony on Mars”. Autonomous rover systems have been vital for space exploration, especially in regions that are dangerous for humans to navigate or difficult to access. A reliable autonomous system with minimal human assistance is therefore necessary to collect data and navigate the Mars terrain. The following report outlines the design and implementation of a rover that satisfies all the requirements to navigate through the mars landscape.

1 Introduction

1.1 Project Requirements

The project is split into seven subsystems, each with its own requirements to enable the rover to navigate the test arena and build “a map showing the locations of aliens and their underground power infrastructure.” Specific requirements for each subsystem were devised to ensure that the entire system functions cohesively. Such requirements are outlined below:

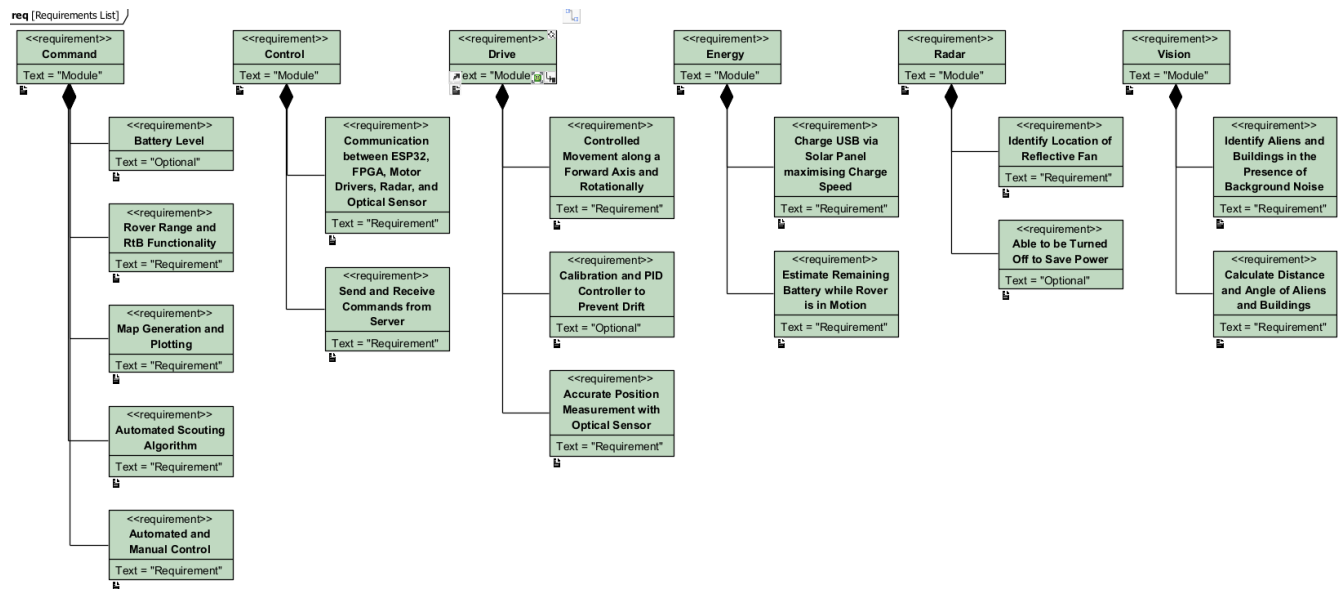


Figure 1: SysML requirements diagram for autonomous rover project

1.2 Project Management & Inter-module communication

In the beginning of the project, each project member was allocated a subsystem. Initially, the roles were split in a way that meant that team members were matched to their strongest subsystems. However, these roles were flexible and reshuffled to ensure that responsibilities were appropriately allocated to the highest priority subsystem.

There were biweekly full group meetings to monitor progress and specify goals for the week, and a Gantt Chart [Appendix A] outlined the overall project plans. A Trello page including each subgroup’s weekly tasks and relevant documents ensured that all group members remained updated and one-to-one subgroup communication and collaboration occurred throughout the week on WhatsApp, Microsoft Teams, and on campus.

2 Command Subsystem

The Command subsystem consists of a webapp, which displays battery level and a live map of the arena, a mobile app, which allows inputs from the user, and a TCP server which communicates with the rover and mobile app as well as functions to process data for the webapp and for the rover.

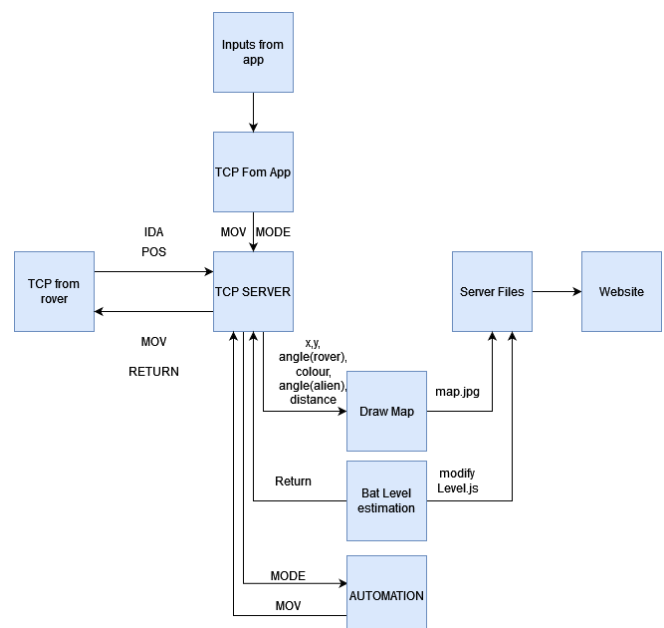


Figure 2: Command system architecture

2.1 User Interface

2.1.1 Website

The web application is composed of a front end running on a JavaScript ReactJS framework, as well as a back end in python. ReactJS was chosen for the front end as it is non-blocking and constantly refreshes upon any change made to the front end sent from the back end. Through updates from the back end, the real time arena map and battery level are received to display on the web browser.

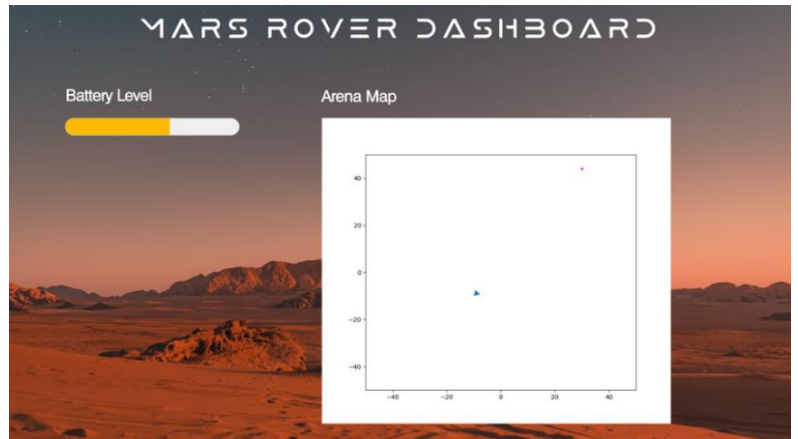


Figure 3: Web application front-end

2.1.2 Mobile Application

The mobile application is made up of a React-Native front end and a Python Flask back end. React-Native was chosen for our front end as it is the mobile app version of React, giving us the same advantages as with the website. The front end sends data to the back end using a POST (REST-API) request from the front-end modules and a POST response in the Flask back end for two routes – ‘move’ (manual rover movements) and ‘mode’ (auto/manual).

The mobile application is designed to control the rover using a switch to toggle ‘manual’ or ‘auto’ mode. In the case where the ‘manual’ mode is selected, left, right, forward, and backward buttons allow discrete movements of the rover. In addition, it sends user inputted initial conditions to adapt the rover’s start position required at the beginning of ‘auto’ mode. A python Flask framework was chosen, because by using Flask it would be easy to integrate a TCP client with the REST API, this is needed to facilitate the forwarding of data, inputted by user, from the mobile app to the TCP server.

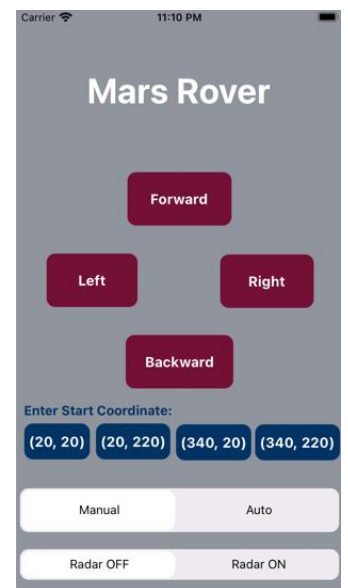


Figure 4: Mobile application front-end

2.2 Data Processing

Data processing is made of several python components which takes the relevant data from a TCP server and creates visual data for the user or dictate what the rover should do.

2.2.1 TCP

A TCP server was chosen because a stable, reliable connection to the rover must be prioritised. It also allows monitoring of the connection status, whereby if the connection drops for too long, it acts as a trigger to indicate to the rover to return to base. In addition, packets don’t drop frequently, and key data arrives in order.

The TCP messages are designed in a purposeful manner so that a short ‘opcode’ is received immediately before any incoming data, which acts as an identifier for different message types. As such, the TCP server processes data depending on what opcode is sent and ignores messages if they are formatted incorrectly. For instance, “IDA” (Identify Alien) takes in the distance and a ball colour code which is plotted on the arena map. The command subsystem receives Message1: “IDA”, Message2: “34”, Message3(1) to say the red ball is 34cm away in the direction of the rover. ‘Dead zones’ are created at these locations to indicate where the rover cannot go.

Sophisticated error handling was also implemented to prevent any potential server crashes should any undefined behaviour occur. This allows for a more robust setup in the face incorrect data being present. [1]

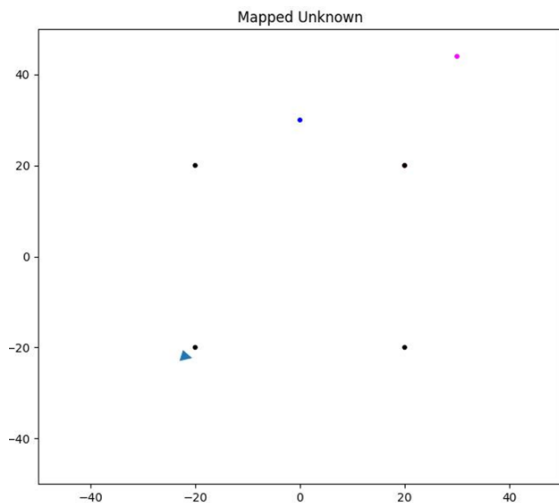





Figure 5: Arena map

2.2.3 Automation

The automated navigation of the rover is designed so that the rover moves to predetermined points to scan the map. These points, along with the actions the rover takes at these points enables the vision subsystem to sweep the entire course with little overlap, while ensuring every part of the arena is scanned.

-  A start point and possible radar point. Upon reaching the point it will turn on the radar turn 65 degrees to look for balls in the corner. Then turn back, turn off the radar and continue onwards.
-  A midpoint of the longest length of the arena, a sweeping scan is done here to cover the most ground.
-  Points in the middle of the arena. Once the rover reaches these points it will do a 360 to cover the rest of the space.

To navigate from one point to another point, the A* algorithm is used to find the shortest route between a start and end coordinate. The A* algorithm requires the arena map to be represented by a 360x240 matrix, corresponding to the dimensions of the arena in centimetres. The matrix is first initialized with zero elements. Once an alien or building is detected, the coordinate of an obstacle is processed by implementing a circle of 1's in the matrix elements around the coordinate using the equation of a circle and modifying elements within a chosen radius. This then allows the A* algorithm to identify the 1's as dead zones that must be routed around if they lie between the start and end points that the rover is currently moving between.

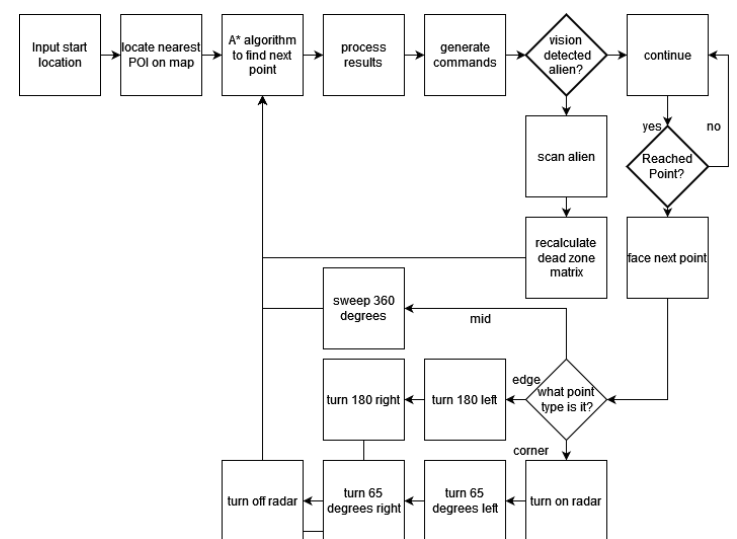


Figure 7: Automation flowchart

The A* algorithm was chosen over Dijkstra's algorithm as both algorithms eventually find the shortest path between two points, however the A* algorithm is less complex computationally, since it employs

a heuristic function which considers estimated low-cost routes first, instead of Dijkstra's algorithm which considers all routes.

The output of the A* algorithm is an array containing the coordinates of the generated route, however, since the A* algorithm works with a matrix, the route generated can only consist of small journeys horizontally or vertically, rather than straight lines at other varying angles, which have a shorter distance. Therefore, to optimise the rover path, a post-routing algorithm that smoothen the unnecessary perpendicular segments was designed. A range of algorithms were tested by testcases made from different start/end coordinates and dead zones created on a test matrix. At first, a smoothing algorithm was implemented to smoothen out the small perpendicular line segments by applying weightings on an error correction algorithm which smoothed segments that lay below a threshold. However, this still resulted in angle differences between points that could be represented on a straight line and so would need further processing if used, which wasn't ideal. Hence, a post-routing algorithm was designed to reduce the array of coordinates generated from the A* algorithm to only contain critical points that approximate the boxed-out route to straight lines. The routes of the two algorithms considered were plotted over testcases along with the A* route's output to compare the post-routing algorithms, which led to choosing critical point estimation due to its more optimized and accurate results, as it compares the gradient of each line segment and compares their gradient difference to a threshold to extract the main points and reduce the number of commands sent to the rover. The critical point estimation algorithm returns an array representing the coordinates on the optimized path and is passed into a function that iterates over the points in the array and compares pairs of corresponding points to extract the angle and magnitude of each line segment. The angles and distances of each line segment are then passed to the TCP server to send the commands to the ESP32. [4][5][6]

During a traverse, if vision detects an object, it will temporarily take control and identify the object. Once identified relevant information is sent from the rover to the server to plot aliens or to add dead zones to the A* map's matrix. Then, Command recalculates the path to the next point and follows the new path, in case the current previously calculated path entered a region that is now a 'dead zone'.

3 Drive Subsystem

The drive subsystem is primarily responsible for the movement of the rover and communicates with the command subsystem to receive the desired direction and distance to move. The subsystem uses an optical sensor to ensure that the movements are accurate, as well as to send to the control and command subsystems the rover's current position.

3.1 Optical Sensor

The optical sensor is used to track the rover's current position in terms of distance travelled and angle rotated relative to the starting position. The initial system consisted of the optical flow sensor sending to the ESP32 via SPI an x value regarding its current angle compared to the starting angle, and a y value which measures the distance travelled forwards as positive and backwards as negative. X is measured in degrees and Y is measured in cm. The values outputted by the optical flow originally did not conform to any unit measurement and so scale factors were found experimentally.

Since the angle rotated and distance travelled need to be updated as frequently as possible to ensure the drive and rotate commands are accurate and up to date, the second core of the ESP32 microcontroller was utilized by multitasking the optical sensor measurements onto the unused core. This allows the system to have a constant loop running in parallel to the control system that solely takes optical sensor data and updates the rover's angle and location for the motor control system.

3.2 Motor Control

Next, the motor control system used to either rotate the rover in place for a given angle or move the rover forwards or backwards a certain distance was designed. To deal with actual motor commands, the Robojax_L298N_DC_motor module was used, whereby each wheel can be instructed easily to move either clockwise or counter clockwise with a given power.

When designing the system to execute rover rotation, trial and error code was initially used to rotate the rover in place. To increase precision, a basic feedback system was designed using the rover's current angle (x value), provided by the sensor, and the given target angle. If the target angle is larger, the rover rotates clockwise, else the rover rotates counterclockwise. By using a slower speed of rotation and the fast rate of optical sensor updates, the subsystem required no further developments.

The design for moving in a straight line for a specified distance requires both moving the set distance and maintaining a constant angle. To ensure the rover braked or adjusted its distance when required, a simple feedback system was designed in a similar way to rover rotation but using the y value. To ensure that the rover remained in a straight line, a PID controller was used which uses the rover's current x value (angle) to calculate an error to be applied to the power of each motor, moving the x value towards the desired angle specified at the start of the command. The coefficients used in the PID code were calibrated using trial and error. [7]

4 Energy Subsystem

The objective of the energy subsystem is to design and create a charging station for the rover where the USB battery is charged by solar panels as efficiently as possible.

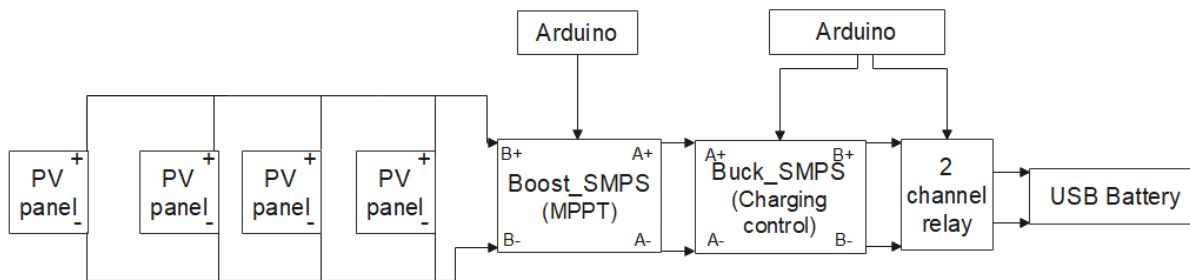
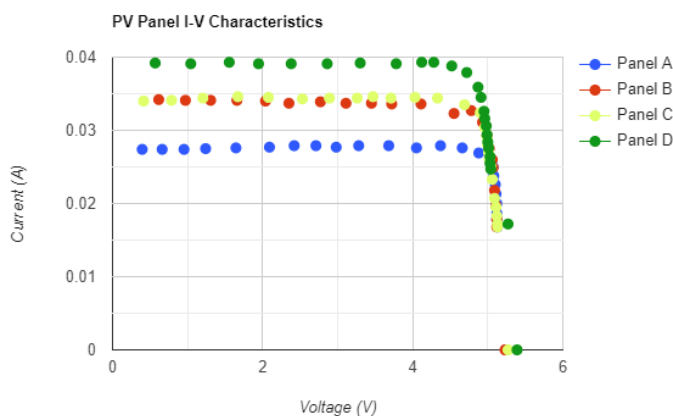


Figure 8: Design of energy subsystem

4.1 Photovoltaic Panels



Figures 9: Current-Voltage characteristics of PV panels

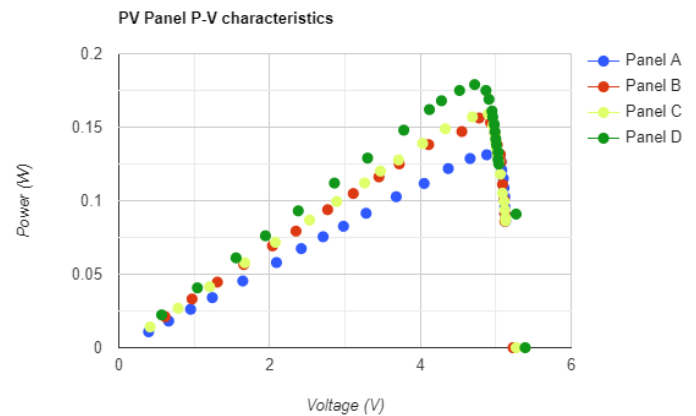


Figure 10: Power-Voltage characteristics of the PV panels

The I-V characteristics of the four PV panels model them as a current source in parallel with a PN diode, with a maximum power point. The corresponding maximum power voltage is between 4.62V and 4.9V for each PV panel.

The three design options considered for the PV panel array were parallel, series, and a parallel-series combination:

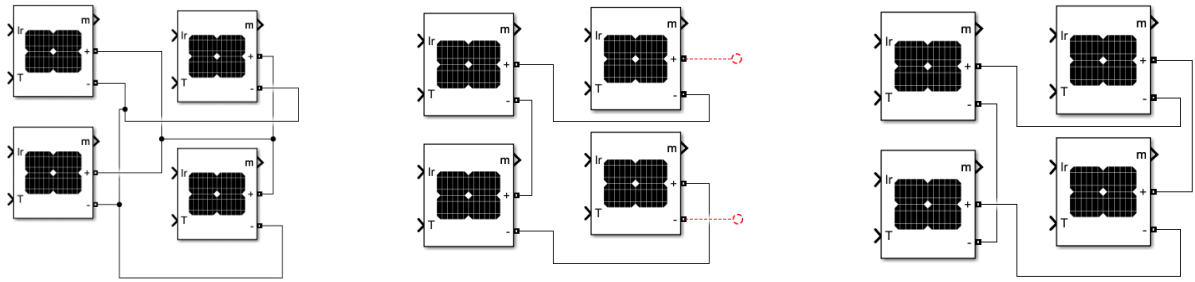


Figure 11: Parallel, Series and Series-Parallel PV

In situations of stable irradiance, both the parallel and series configurations operate with similar power outputs; the series will have a higher voltage, and lower current output, whereas the parallel will have a higher current and lower voltage [8]. However, choosing the series configuration would restrict our subsequent design options due to its high voltage output: the maximum input of the Buck SMPS is 8V, so the first SMPS could not be a Boost SMPS as it would increase the already high PV voltage. Therefore, the design options for a series PV array would require a Buck SMPS first followed by either a Boost SMPS or a Buck SMPS.

An additional issue arises when considering partial shading, which has a greater effect on a series PV array. If one panel is operating at less than 100% power, all the panels will be affected whereas with parallel, the other panels can still operate at 100% power. [9]

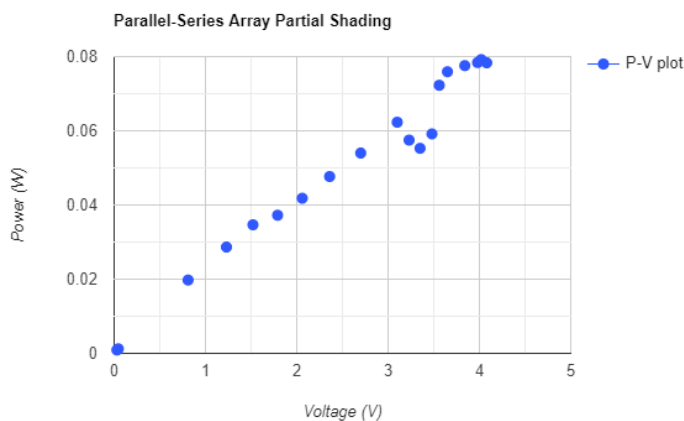


Figure 12: Parallel-Series with bypass diodes

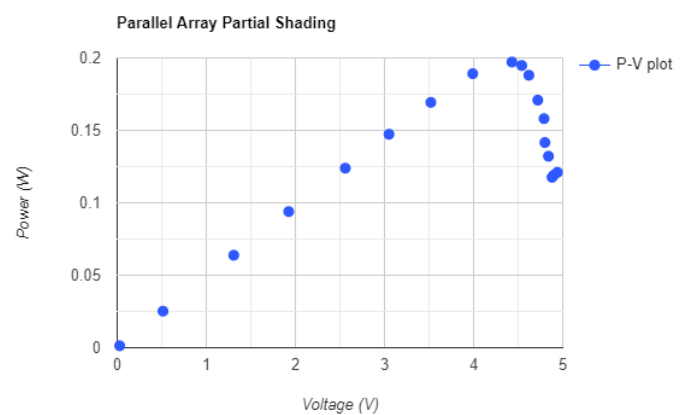


Figure 13: Parallel in partial shading

A potential design solution was the parallel-series combination (which has a less significant effect from partial shading than series) combined with bypass diodes. However, this resulted in two maximum power points with one at a much lower voltage. To resolve this, adding an additional slow loop to our MPPT code was considered, which would increase the duty cycle by a large amount to push the voltage back up to the second maximum voltage point in case of partial shading. However, to ensure that the design also considered simplicity, the parallel configuration was chosen. Although a parallel array requires a larger minimum voltage to operate, which affects the panels in situations of lower irradiance, it is less affected by partial shading. A smaller decrease in output power is exhibited in comparison to the series and parallel-series configurations.

The output voltage of the parallel PV array at its maximum power point is close to the input voltage range of the USB battery (4.5V to 5.2V). Therefore, the complete system would need to include both a Buck SMPS and a Boost SMPS to prevent the output voltage of the second SMPS from increasing or decreasing too much. The final design implemented is a Boost SMPS followed by a Buck SMPS because there is greater control over the final output voltage. If the second SMPS was a Boost SMPS (which is more unstable), it would be more difficult to predict the voltage and therefore implement the charging control.

4.2 Maximum Power Point Tracking

The variation of irradiance on the photovoltaic panels results in varying voltages and currents. This affects the output power of the panels. The objective of the MPPT algorithm is to use the Perturb and Observe method to push the system to operate around the maximum power point and therefore ensure maximum efficiency, by continuously modifying the duty cycle.

The code includes a fast loop and a slow loop. The fast loop measures the input voltage and current to the SMPS and the slow loop uses states for the Perturb and Observe method. The code also limits the maximum output voltage to 7.5V so that the following Buck SMPS remains operational.

The MPPT code has three states: state 0, where the duty cycle is set to 0, essentially turning off the SMPS. This permits a level of manual control using the OL-CL switch. When the system reaches state 1, the duty cycle decreases and when it reaches state 2, the duty cycle increases. The algorithm uses the measurements of voltage and current (and therefore power) from the fast loop and compares them to the previous voltage and power. The corresponding states are described for positive and negative changes in voltage (dV) and power (dP) in *figure 14*.

dV	dP	New State
-	-	2
-	+	1
+	-	1
+	+	2

Figure 14: MPPT state table

4.3 USB Battery Charging

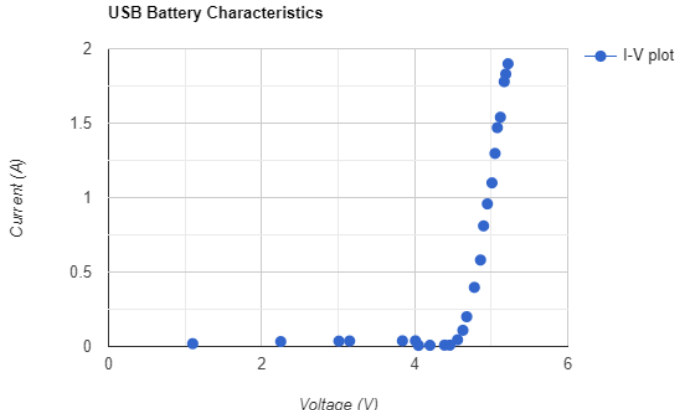


Figure 15: USB battery I-V characteristics

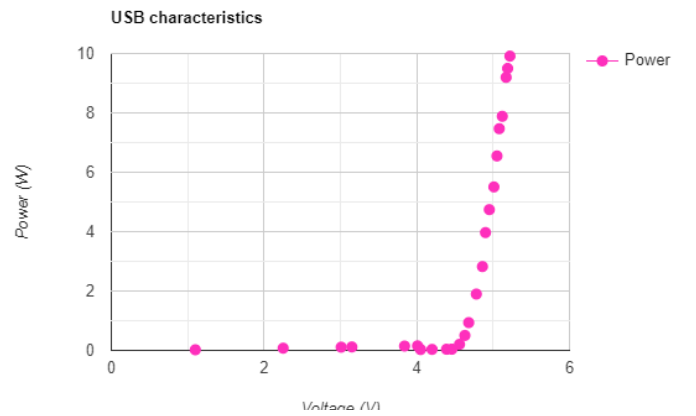


Figure 16: USB battery P-V characteristics

Within the USB input voltage range, as voltage increases, the current drawn by the battery also increases, and the charge time decreases (since the battery capacity is 5000mAh). This behaviour must be controlled since the maximum power available from the photovoltaic panels is approximately 4.6W. At the maximum voltage of 5.2V, for example, the current drawn is approximately 1.9A which equates to 8.55W. Since the wattage is not available from the input, it results in a large voltage drop from the solar panels to less than 0.1V.

The solution was to control the current drawn using the Buck SMPS input voltage. An additional load line was created to set a reference voltage of 7V, which aligned an input voltage of 7.5V to an output voltage of 4.8V and an input voltage of 6.5V to an output voltage of 4.5V, using the equation: $\delta = \frac{V_O}{V_I}$

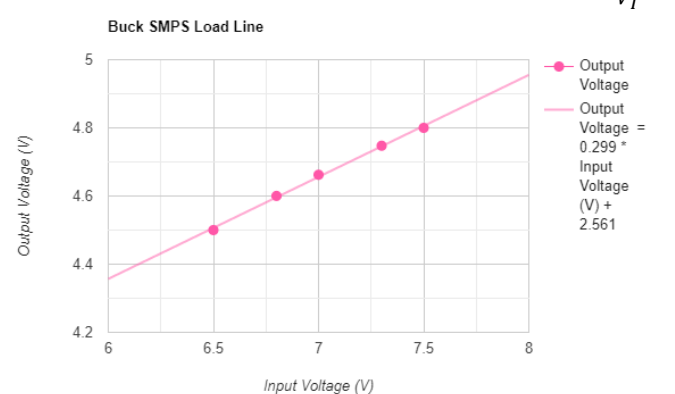
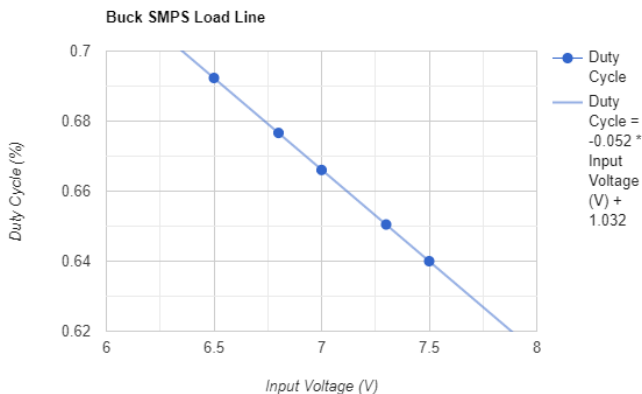


Figure 17 (left): Buck Duty Cycle vs V_{in} and Figure 18 (right): Buck V_{out} vs V_{in}

This load line ensures that as the input voltage of the Buck SMPS increases, the output voltage also increases, causing the USB battery to draw more current. This minimises the charge time whilst an additional line of code ([Appendix D](#)) ensures the output remains within the USB voltage range by reducing the duty cycle if the output voltage increases above 5.2V. If the increased voltage causes more current, and therefore more power, to be drawn than is supplied, the system will undergo a dramatic voltage drop, and so the input to the Buck SMPS will dramatically reduce. The load line at this point results in the output voltage to also decrease dramatically. Consequently, the current drawn by the battery decreases, therefore ‘reversing’ the voltage drop since the power required at the USB battery input is no longer more than the power supplied.

As an additional safety precaution, a channel relay was inserted between the second SMPS and the USB battery to restrict the input voltage of the battery to within its voltage range of 4.5 to 5.2 Volts. However, the charge controller was still implemented such that the USB battery input voltage remains within the range as much as possible. This prevents the relay from acting as an open switch and the battery from not charging.

4.4 Battery Percentage

The energy subsystem also provides an approximation of the battery percentage to the command subsystem. The capacity of the battery is 5000mAh, which equates to 25000mWh. Taking measurements of the average current and power supplied to the main parts of the integrated rover, as well as the power analysis of the FPGA produced by Quartus and the datasheet values for the ESP32 [10], enabled an approximation of the total output power from the USB battery:

Subsection	Current (mA)	Voltage (V)	Power (mW)
Motor	125.8	3.3	415.14
Optical Sensor	157	5	785
FPGA	-	-	413.01
ESP32	500	3.3	1650
Radar	42	5	210
Total (without radar)	-	-	3263.2
Total (with radar)	-	-	3473.2

Figure 19: Rover’s power consumption

From the above results, the approximate total run time of the rover is 7.5 hours. The average power values are included in the capability code within the web app backend which uses the following equation:

$$\text{Battery percentage} = \frac{(25000 - ((P_a \cdot t_a) + (P_b \cdot t_b)))}{25000}$$

Where P_a is the total average power with the radar turned off, P_b is the total average power with the radar turned on and t_a and t_b are the respective total run times for both modes: t_a and t_b are updated as the rover runs and as it switches between the two modes (radar off and radar on).

4.5 Range Capability

The rover’s displacement from the base (its starting point) is continuously updated using (X,Y) coordinates. Using the rover’s average speed, the equation $\text{time} = \text{distance}/\text{speed}$, calculates the time taken for the rover to return directly to the base in a straight line. Since the battery capacity equations approximate the total possible run time of the rover, the ‘time remaining’ for the rover before its battery runs out is calculated by subtracting the current ESP run time. If the time remaining is less than the time required to return to base plus chosen buffer time, a message is sent to return the rover to the base. The

buffer time is required to account for any obstacles such as buildings or aliens that will require the rover to adjust its route back to the base.

5 Radar Subsystem

The *HB100 Doppler radar module* was used to detect the blades of the fan, which rotate at 366Hz. This radar module uses the doppler effect to locate the position of the fan beneath the arena. A signal of frequency f_0 is sent, and one of frequency $f_0 + \Delta f$ is received, of which the Δf component can be retrieved and processed. The signal is first filtered and amplified to remove clutter, and then processed as a digital input to send a signal to the control subsystem when the radar detects the fan.

5.1 Input Signal

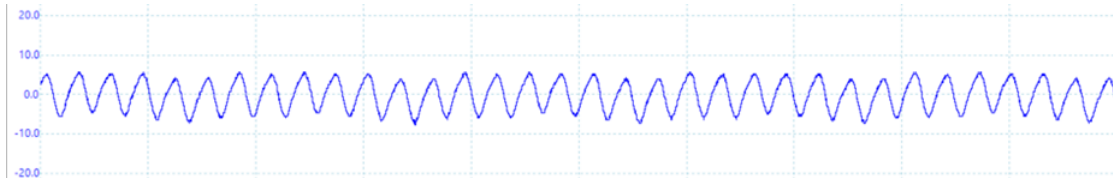


Figure 20: Input signal

The retrieved signal has a frequency of 366Hz, and an additional DC component caused by f_0 , i.e., -100mV. When the radar module was placed next to the fan, an input signal of approximately 10mV amplitude was measured. Therefore, to detect the blades, an amplification stage was required to strengthen the received signal. The A-D converter of the ESP32 has a resolution of 12 bits, however, since its pins are non-linear ([Appendix E](#)) [11] without amplification the signal would be almost indistinguishable from background noise.

5.2 Measurements

Three main sources of noise for the radar were observed. The first was the peak at 0Hz, which represents the stationary objects surrounding the radar. The second was the peak at approximately 50Hz caused by the mains power noise in the background, and the third was white noise. Other visible frequency components were general clutter, due to surrounding objects such as electronic devices and people entering and leaving the area. Since the relevant frequency required was within the 366Hz region, a bandpass filter for clutter rejection was necessary to eliminate noise at unwanted frequencies. The signal shape observed in the time-domain plot was almost sinusoidal, however, it was not a perfect sinusoid due to the multiple rotating blades of the fan.

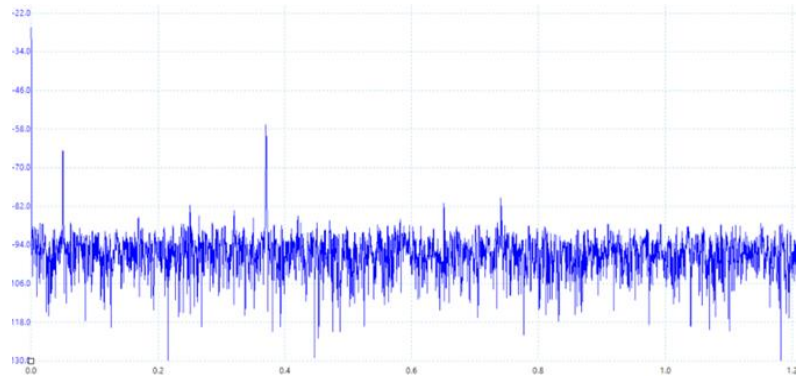


Figure 22: Radar signal frequency spectrum



Figure 23: Input signal at a further distance

As the radar moved away from the target, the amplitude of the 366Hz peak decreased. This corresponds to the effect of distance on the radar as described by the radar range equation, that the power received by the radar decreases as the distance from the target increases by the ratio $1/R^4$.

The angle of the radar above the target also influences the magnitude of its output signal, and therefore the distance from which it is possible to locate the fan. Directly above the target, the optimum angle for the radar is 0 degrees (horizontal). However, when the radar is further from the fan, a more vertical angle provides better detection by ensuring the radar signal incident on the target remains perpendicular.

When moving around and sideways from the target, the peak on the frequency spectrum varied from the 366Hz point. This demonstrates the effect of the rover in motion, being that the Δf plot is proportional to the velocity of the blades relative to the velocity of the radar, rather than solely the velocity of the blades. This influenced the chosen bandwidth for the bandpass filter to ensure that the relevant peak would not be attenuated due to small frequency changes from the motion of the radar.

5.3 Signal Processing

For the amplification and filtering, a 3-stage bandpass filter consisting of 2nd order bandpass filters with multiple feedback was used. A 2-stage bandpass filter ([Appendix E](#)) was considered but was more inaccurate due to its more gentle roll-off rate. A 4-stage filter ([Appendix E](#)) was also discarded because its design was more complicated compared to the insignificant improvement in accuracy. Thus, the final design was the circuit schematic shown in Figure 4, with the transfer function shown in Figure 5.

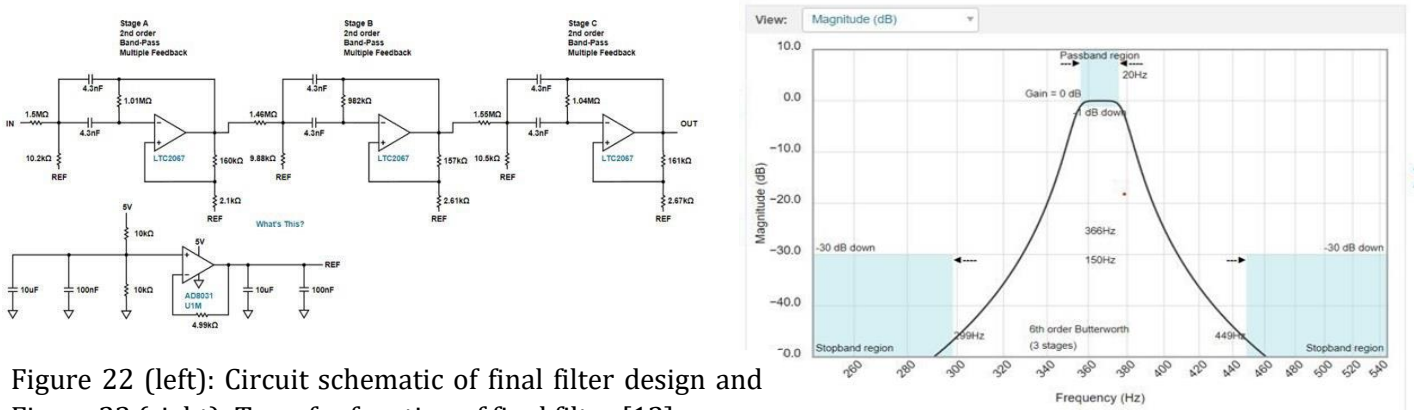


Figure 22 (left): Circuit schematic of final filter design and Figure 23 (right): Transfer function of final filter [12]

The gain at 366Hz \pm 50Hz is 40 dB, corresponding to a gain of 100 of the input signal.

By testing the filter at different frequency values, it was concluded that the design amplifies a small window of frequencies (50Hz away from the centered frequency of 366Hz) and attenuates all other frequencies.

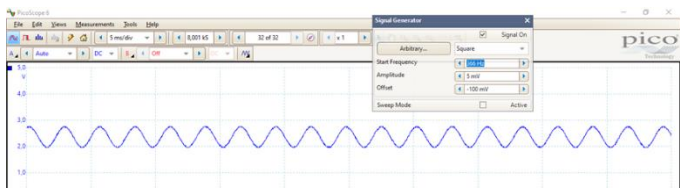


Figure 24: Output when inserting a 366Hz signal

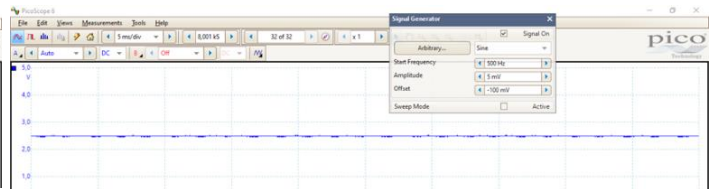


Figure 25: Output when inserting a 500Hz signal

5.4 Analogue to Digital

The output of the filter is connected to port A0 of the ESP32, which acts as an A-D converter. It samples the analogue value at a frequency of 840Hz to satisfy the Nyquist frequency requirements, since the maximum frequency at the A-D converter input is 416Hz. When the output of the filter is greater than

the threshold voltage, the ESP32 sends a HIGH output to the control subsystem so that the fan can be located on the map.

An alternative method was to connect a peak detector circuit directly to the output of the filter. A capacitor removes the DC offset of 2.5V, the diode stops the capacitor from discharging during the negative half cycles, and the capacitor in parallel with the resistor averages the DC value of the peak. The output of this circuit was then connected to the ESP32 port.

N.B Professor Bouchaala advised the group to include in the report that the radar module was broken and therefore complete testing was not possible.

6 FPGA and Camera Subsystem

The objective of the FPGA and Camera subsystem is to use the rover's camera to detect several aliens (coloured balls) and alien buildings (black and white striped objects) and relay this information to the control subsystem. To achieve this, the subsystem was broken down into three smaller objectives:

- 1) Identify unique objects of interest accurately using object detection algorithms, eliminating as much noise as possible through adequate filtering.
- 2) Obtain accurate distance measurements such that the control subsystem knows how far the objects of interest are from the rover.
- 3) Set up a communication interface where the FPGA can communicate this data with the ESP32 (control subsystem).

To accomplish the first task, data is passed through an image processor which modifies the input video data and applies appropriate filtering and object detection.

6.1 Image Processor

The image processor can be separated into four main sections which have different functions for processing the video data to achieve accurate detection of the objects of interest. The first reads video data, including pixel position, pixel colour, and packet information. Pixel data is transmitted from the camera to the FPGA one pixel at a time, from the bottom right to the top left of the current frame. Each pixel has a varying 24-bit RGB value which can be separated into 8 bits for each colour. Secondly, there is a colour detection sub-module to accurately identify the colours of the balls. Filtering is then applied to filter out noise and allow for accurate object detection.

6.1.1 HSV Conversion

The first key design decision was to implement a conversion between RGB (red, green, blue) values to HSV (hue, saturation, value). This was implemented in the module `rgb_to_hsv.v`. This conversion was essential to our object detection algorithm because HSV is more robust in different lighting conditions. Given the variable lighting conditions during testing and in the actual Mars arena, hue values vary less than RGB values. The conversion was performed using a widespread Verilog implementation of an RGB to HSV convertor. Credit to unknown author [13].

The thresholds for the hue, saturation and value for each coloured ball were developed through several methods. Initially, simple thresholds from an online colour picker [14] were used to determine the range of hue values for each of the coloured balls. These initial thresholds were tested in the arena to simulate similar lighting conditions on the day of the demonstration. Adjustments were made using the MATLAB colour threshold tool (**see Appendix F**), which provided a fast method to determine the hue, saturation and value thresholds in the arena light conditions.

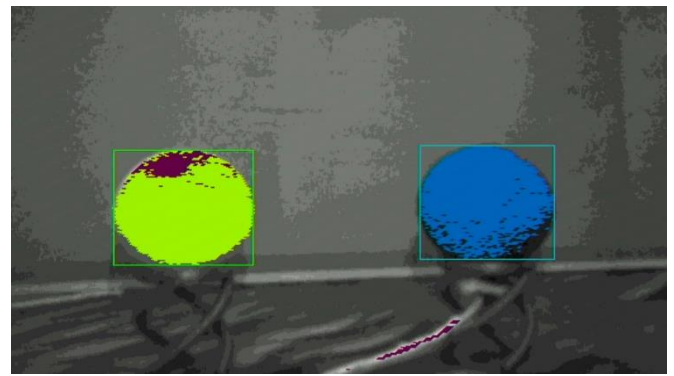


Figure 26: Light green and dark blue ball detection in arena

6.2 Filtering and Ball detection

The second stage in the object detection process was a suitable filter to filter out unwanted colours and noise. Implementation of certain types of filters such as median 3x3 filter or Gaussian filter would require large amounts of memory usage because pixels above and below would need to be stored and accessed. Given the memory constraints on the FPGA and the fact that large amounts of memory stores and accesses may affect frame rate, therefore increasing the delay between receiving a frame and processing it, deliberate thought was put into the types of filters and techniques used to achieve accurate object detection. Each filter was evaluated on its hardware complexity (thus also on power consumption) and its capacity to remove background noise. The walls of the arena already reduced the level of background noise by a substantial amount, leading to the decision to prioritise hardware simplicity over filtering performance. As a result, complex edge detection algorithms, such as Sobel or Canny edge detectors were not implemented. A median 3x3 filter was initially implemented due to the promise that the colours within a 3x3 area of each pixel will produce accurate object detection; a greater coloured pixel density compared to the background is most likely to be that coloured ball. This method, however, requires buffering of entire rows of pixels which uses a large amount of on-chip memory. Simpler filters were subsequently considered to reduce complexity. The second filter implemented was a 5-pixel mode filter which takes in 5 consecutive pixels and evaluates the pixel to a ball colour if all 5 pixels are within one of the HSV colour thresholds – otherwise, it is evaluated to grey. This method proved very effective at reducing background noise and computationally inexpensive. The third filter implemented for comparison was a simple weighted average filter, which is an extension to the 5-pixel mode filter, assigning higher importance to pixels closer to the current incoming pixel. This filter also proved effective at reducing background noise though was evaluated to be more computationally expensive than the 5-pixel mode filter. There was no noticeable difference in the effectiveness of reducing background noise between the two filters. A performance comparison of the weighted average filter and the 5-pixel mode filter can be given by the following power and timing analysis.

5-pixel weighted-average filter

Timing Analysis (MAX10_CLK1_50):

- Slow 1200mV 0C Model: 83.74MHz
- Slow 1200mV 85C Model: 74.32MHz

Power Analysis – Total power dissipated: 461.27mW

5-pixel mode filter

Timing Analysis (MAX10_CLK1_50):

- Slow 1200mV 0C Model: 74.52MHz
- Slow 1200mV 85C Model: 68.36MHz

Power Analysis – Total power dissipated: 445.31mW

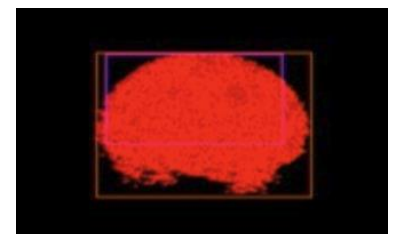


Figure 27: Weighted average filter

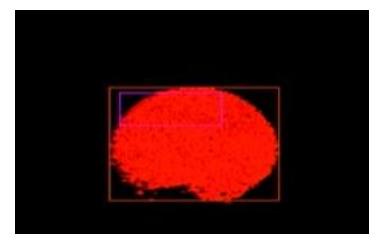


Figure 28: 5-pixel mode filter

Though both filters provided very similar results in producing accurate bounding boxes for object detection, the 5-pixel mode filter achieved a slightly better performance and was implemented in the final product. However, as it would be impossible to design a filter which removes all noise, to mitigate the effect of any unremoved noise, additional logic was added to ensure that bounding boxes would only form if the bounding box was approximately square shaped. That way, any anomalies would be ignored.

6.3 Building Detection

The buildings to be detected are vertical cylinders consisting of vertical black and white stripes. Three main cases were considered, each with increasing complexity: detection when there is only one building in a frame, detection when there are multiple buildings separated by a space between them, and finally detection of multiple buildings that overlap each other.

To deal with the first case, an algorithm was designed to scan each frame horizontally and count the number of times there is an abrupt change in pixel colour from black to white or vice versa. The algorithm was implemented using the same 5-pixel mode filter used for ball detection.

When implementing the case with multiple buildings separated by space, the solution was to simply restart the building detection algorithm once a building had been detected, but then the colour of a pixel returned to being neither black nor white. Since all stripes have the same width, if multiple buildings were detected, the building with the larger maximum pixel stripe width would be chosen as the building to be detected and measured, as it is closer to the rover.

Finally, an algorithm was implemented to find the closest building to the rover when there are two overlapping buildings. As all stripe widths are the same for any given building, when scanning horizontally along a building, stripe widths increase towards a maximum, and then decrease. This meant that when the stripe widths start to decrease, if they begin to increase again before the building ended, a new building has been detected. The maximum stripe width of each building was stored, and the building with the largest stripe width would be chosen as the closest building.

6.4 Distance Calculations

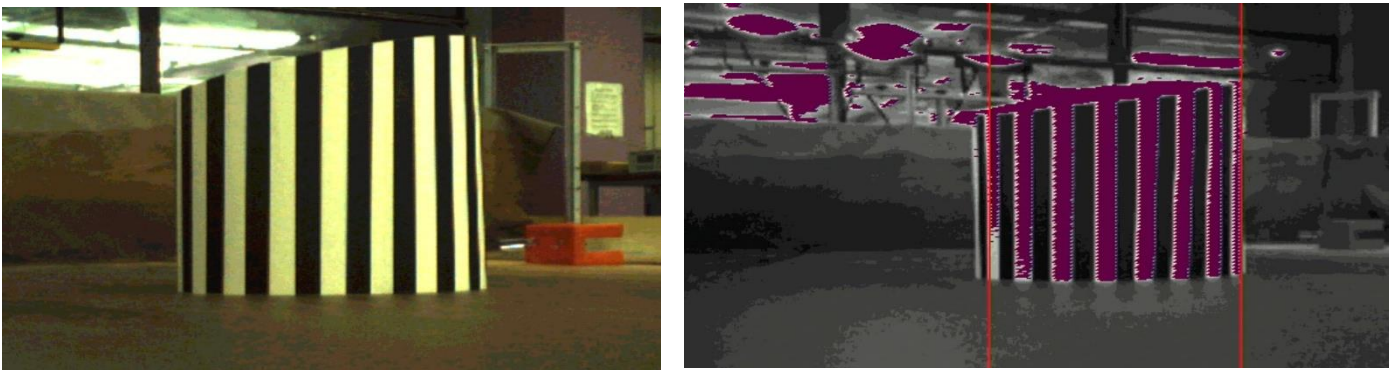


Figure 29: Building detection

The distance between the rover and objects is calculated using the size of the bounding box generated by the object detection algorithm. The distance calculation system calculates the distance between an object and the rover only if the object is sufficiently in the centre of the camera display. If not, the drive control system is used to rotate the rover until the object is centred.

The relationship $D \propto W/P$ was used [15] where D is the distance from the rover to the object, W is the width of the object, and P is the number of pixels between the minimum x pixel of the object and the maximum x pixel. The constant of proportionality in this equation is the focal length of the camera, and in this equation, the focal length has been approximated as linear. In reality, the focal length is non-linear with respect to distance, however, to use ensure that this approximation is usable, any distance

calculations performed by the FPGA were set to only be valid between the range of 20-60cm, where it was experimentally (see **Appendix F**) proven that the constant of proportionality only varied slightly.

The constant of proportionality, in this case the approximated focal length, was found experimentally, by measuring and plotting graphs of W/P against D for a range of values and finding the gradient. As the stripe width of buildings is known and constant, the width of the largest stripe in camera view was used as the P value. By also including the number of visible stripes of the measured building in the communications between the rover and control, the size of any measured buildings could also be registered and mapped.

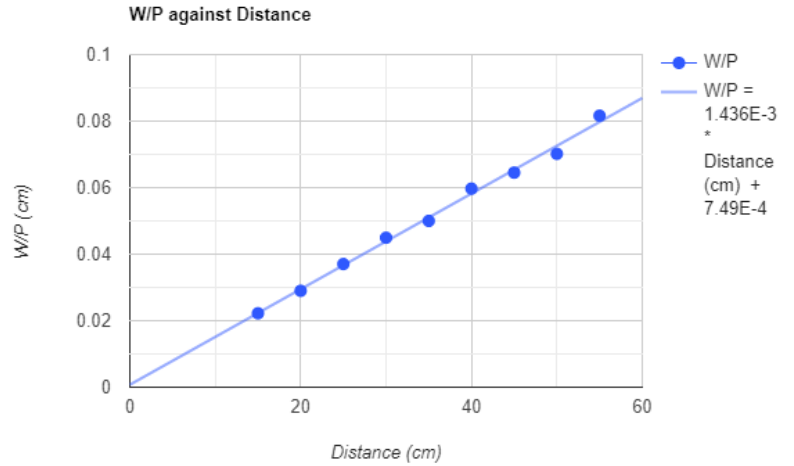


Figure 30: Graph of W/P against distance of a ball

Similarly, as the stripe width of buildings is known and constant, the width of the largest stripe in camera view was used as the P value. By also including the number of visible stripes of the measured building in the communications between the rover and control, the size of any measured buildings could also be registered and mapped.

6.5 SPI Communication

SPI communication was used to handle sending data about the distance between the rover and objects to the ESP32 control system, with the FPGA being the SPI slave and the ESP32 the SPI master [16]. The FPGA either sends drive instructions to rotate the rover and centre the object in the screen or sends object data that informs the rover of a detected ball's colour and distance from the rover. To avoid re-measuring ball colours which have already been measured, additional logic values were added to store which colours have already been measured; if any of these colours are found again, the FPGA will ignore them. (see **Appendix F**) for code.

7 Control Subsystem

The Mars rover was controlled using an ESP32 microcontroller running an Arduino script. The script runs as a TCP client connecting to the TCP server in the web app back-end. The control script is governed by states that are maintained using Boolean variables which affect the conditions of if-statements that manage the transition of each state – making the control script analogous to a state machine.

7.1 Drive

The microcontroller waits for a message from the TCP server and then updates a Boolean variable once the message is received. This then allows the control flow to transition to the state where the command received from the server, either a target distance or angle, is passed to the drive functions that control the motors.

The optical sensor is designed to continuously send the rover's current distance and angle position to the TCP server in the web app back-end by sending an encoded message over the existing TCP socket. The optical sensor measurements and TCP communication are run in parallel to the rest of the processing on the script by utilising the second core on the ESP32 to perform multitasking, which allows for the measurements to update at a faster rate and hence send a position to the server that closely maps the actual position of the rover.

7.2 Radar

The control module returns the value 1 if there is a fan and 0 if there is no fan and then forwards it to the TCP server. In addition, when 'ARA' or 'DRA' is received from the TCP server it turns the radar on and off, respectively, to save power.

7.3 Vision

To avoid message complexity, only three sets of information are sent to the control subsystem from vision. The vision subsystem determines if there is an object in view and determines which is closest. It then sends a message to the control subsystem to indicate if the rover must adjust its rotation left or right to point directly at the object in question for a clearer reading. Once the rover has adjusted accordingly, the vision subsystem sends a message indicating the colour of the ball and the distance of the ball from the rover. The control subsystem sends an acknowledgement back to vision to indicate that the object in question has been registered.

8 Integration

The method of implementation of modules was done in the order implied by the following diagram.

The way we approached the overall project was by implementing module components, testing them so that they meet the basic standards of the project then moving on to the next components. This way the rover was slowly built up and merged to achieve a final product

9 Conclusion:

In evaluating the use of the hardware provided, we reduced the need of additional hardware by exploring methods to optimize the software implementations where possible. This allowed for a greater focus on achieving the project requirements by tending towards the minimal viable product, in order to achieve cost effectiveness and compactness. Each subsystem in the rover can execute any tasks assigned by the specification, and the control system and sophisticated automation process allowed these tasks to perform harmoniously in performing the action of scanning an unknown arena.

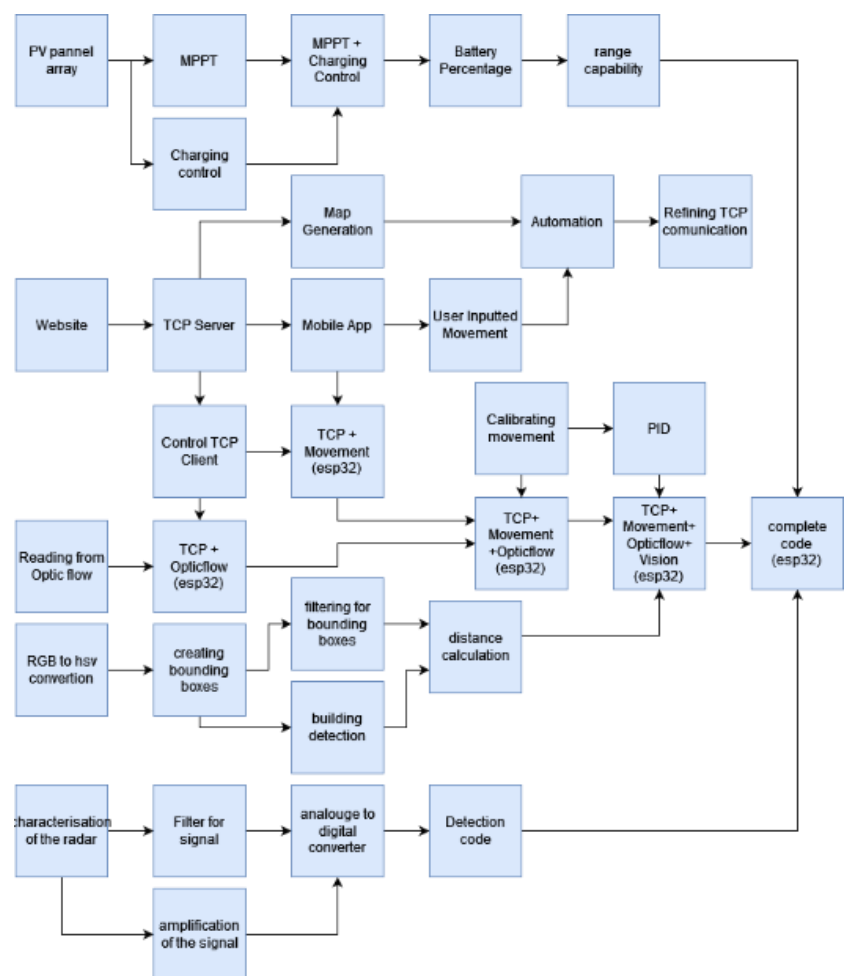


Figure 31: Implementation diagram

For future work in the energy subsystem, the battery's state of health and its effect on its discharging behaviour would be considered in the battery percentage calculation algorithm to improve rover run time accuracy. Another design change that was considered throughout the project was a gyroscope, to deal with optical sensor uncertainty, however a well-designed PID controller was able to decrease uncertainty regarding imprecise drive functionality.

Bibliography

- [1]"TCP Chat in Python - NeuralNine", NeuralNine, 2019. [Online]. Available: <https://www.neuralnine.com/tcp-chat-in-python/>[Accessed: 16- Jun- 2022].
- [2]A. Qassim, "Easy Steps To Plot Geographic Data on a Map—Python", Towards Data Science, 2019. [Online]. Available: <https://towardsdatascience.com/easy-steps-to-plot-geographic-data-on-a-map-python-11217859a2db>. [Accessed: 14- Jun- 2022]
- [3]"Matplotlib documentation — Matplotlib 3.5.2 documentation", *Matplotlib.org*. [Online]. Available: <https://matplotlib.org/stable/index.html>. [Accessed: 16- Jun- 2022].
- [4]N. Swift, "Easy A* (star) Pathfinding", *Medium*, 2017. [Online]. Available: <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>. [Accessed: 19- Jun- 2022].
- [7]J. Cook, "PID Controller Basics & Tutorial: PID Implementation in Arduino", *arrow.com*, 2019. [Online]. Available: <https://www.arrow.com/en/research-and-events/articles/pid-controller-basics-and-tutorial-pid-implementation-in-arduino>. [Accessed: 17- Jun- 2022].
- [8] N. Yarbrough. (2021). *Solar Panels – Series or Parallel?* [Online]. Available: <https://www.explorist.life/solar-panels-series-vs-parallel/>
- [9] M. Aravinda and K. Padmavathi, "Simulation study of partial shading effect on series, parallel and series-parallel connected PV modules," *2017 International Conference on Smart grids, Power and Advanced Control Engineering (ICSPACE), Bangalore, India, 2017*, pp. 1-10
- [10] Expressive Systems, "ESP32 Series Datasheet", 3.9, Aug. 2016 [Revised Mar. 2022]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- [11] Random Nerd Tutorials. (2019). ESP32 ADC – Read Analog Values with Arduino IDE [Online]. Available: <https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/>
- [12] Tools.analog.com. (2022). *Filter Design Tool | Filter Wizard | Analog Devices*. [online] Available at: <https://tools.analog.com/en/filterwizard/> [Accessed 23 June 2022].
- [13] Programmerclick.com. n.d. *rgb a hsv (Verilog) - programador clic*. [online] Available at: <https://programmerclick.com/article/54951095995/> [Accessed 13 June 2022].
- [14]N. Eduardo Lundgren, "Color Picker - HSV Palette Example | AlloyUI", Alloyui.com. [Online]. Available: <https://alloyui.com/examples/color-picker/hsv.html>. [Accessed: 16- Jun- 2022].
- [15]A. Rosebrock, "Find distance from camera to object using Python and OpenCV", PyImageSearch, 2015. [Online]. Available: <https://pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/>. [Accessed: 15- Jun- 2022].
- [16]"fpga4fun.com - SPI 2 - A simple implementation", Fpga4fun.com. [Online]. Available: <https://www.fpga4fun.com/SPI2.html>. [Accessed: 18- Jun- 2022].

Appendix A: Project management

Mars Squad 2022

PGB2

Aleera Ewan/Rohan Gandhi/Valia Giannopoulou/

Anthony Jones/James McManus/Omar Zeidan/

Sam Hesketh Fatchen

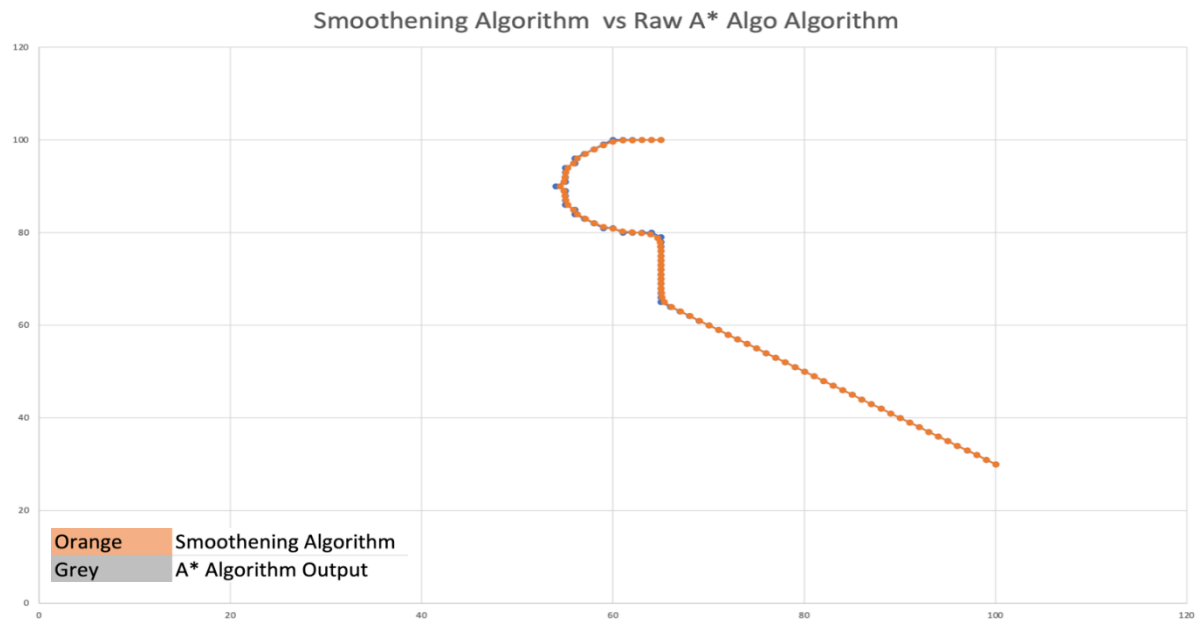
Project Start:	23/05/2022
----------------	------------

Today:	23/06/2022
--------	------------

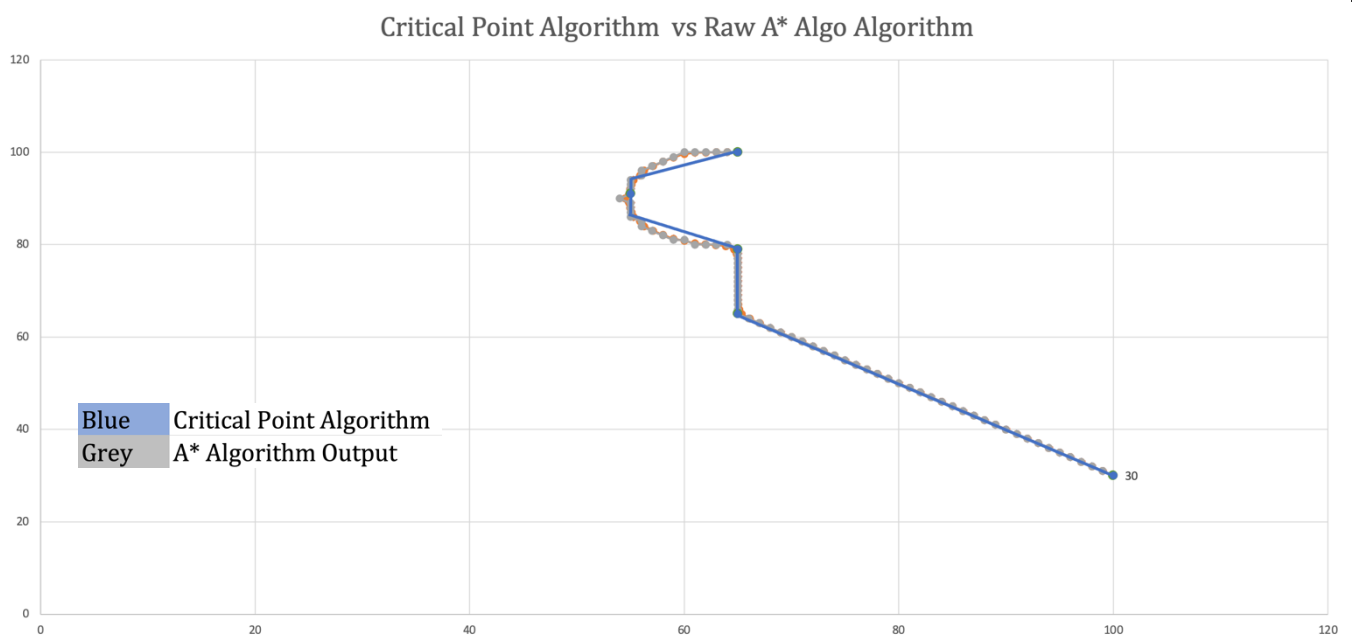
Display Week:	1
---------------	---

[illegible]

Appendix B: Command



Result of smoothing the A* algorithm route



Result of critical point estimation of the A* algorithm route

Appendix C: Drive

Y-Axis Calibration (Distance)

Data set	real start (cm)	real end (cm)	real distance (cm)	measured start (cm)	measured end (cm)	dist travelled (cm)	scale factor
1	61	100.3	39.3	-932	-2672	1740	0.022586207
2	50.2	91.2	41	-2227	-4053	1826	0.02245345
3	31	71.3	40.3	-3296	-5088	1792	0.022488839
4	31.4	71.7	40.3	-4835	-6674	1839	0.021914084
5	28.1	70.7	42.6	-6543	-8428	1885	0.022599469
6	32.2	74.4	42.2	-8206	-10092	1886	0.022375398
Average Y SF							0.022402908

X-Axis Calibration (Angle)

Data set	Travelled angle (deg)	initial x	final x	dx	scale factor
1	90	81	-1049	1130	0.079646018
2	90	-792	-1937	1145	0.07860262
3	90	-959	-2080	1121	0.080285459
4	90	-1096	-2216	1120	0.080357143
5	90	-1136	-2256	1120	0.080357143
6	90	236	-928	1164	0.077319588
7	90	126	-1017	1143	0.078740157
8	90	-235	-1344	1109	0.081154193
Average X SF					0.07955779

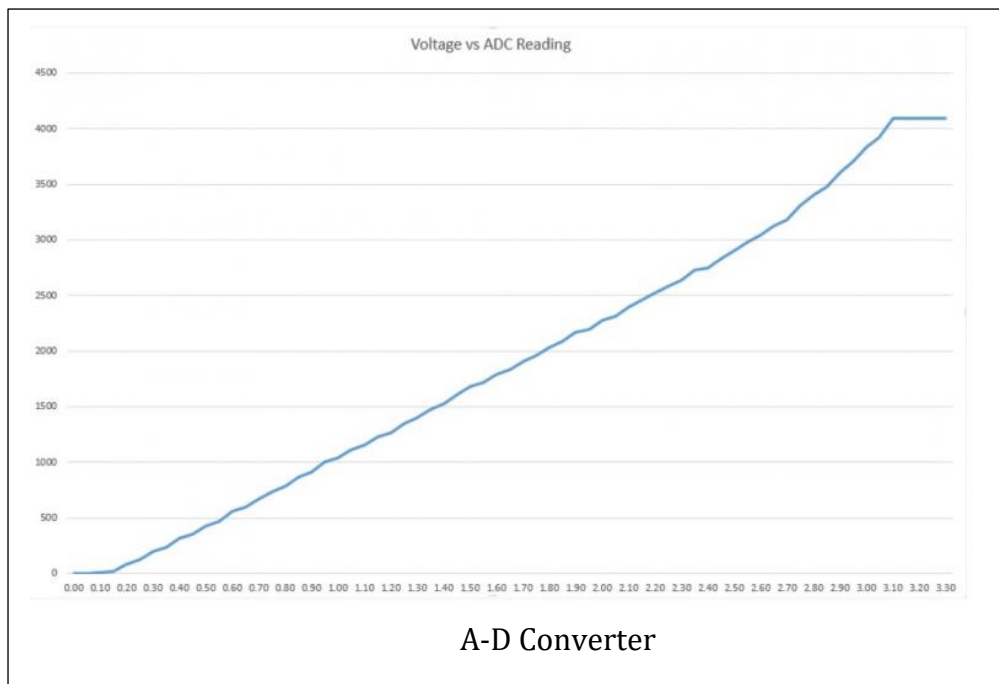
Experimental Values to Calibrate the Optical Flow

Appendix D: Energy

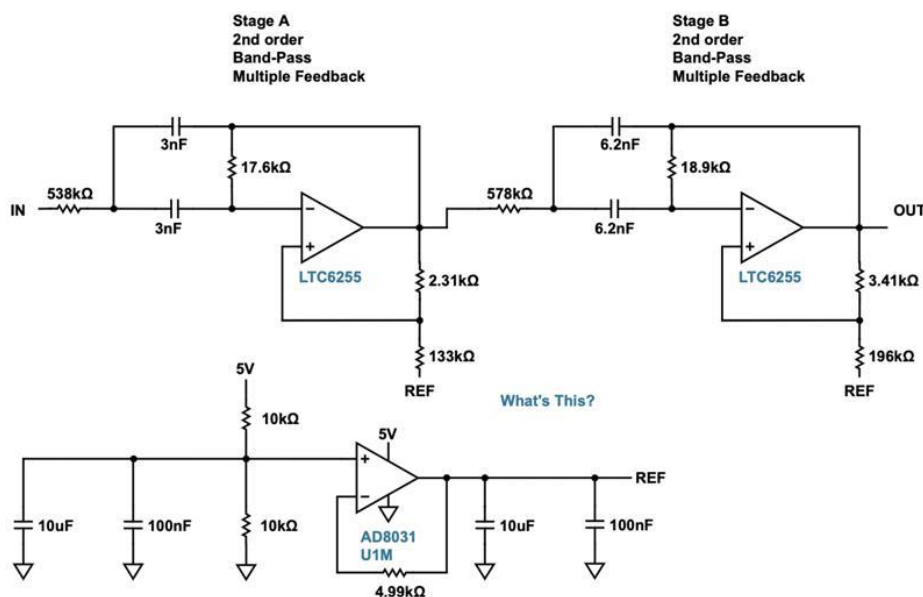
```
//Vmax=5.2  
if(Vout>Vmax){  
dutyicycle-=0.05;  
}
```

Charging Code Limiting Output Voltage

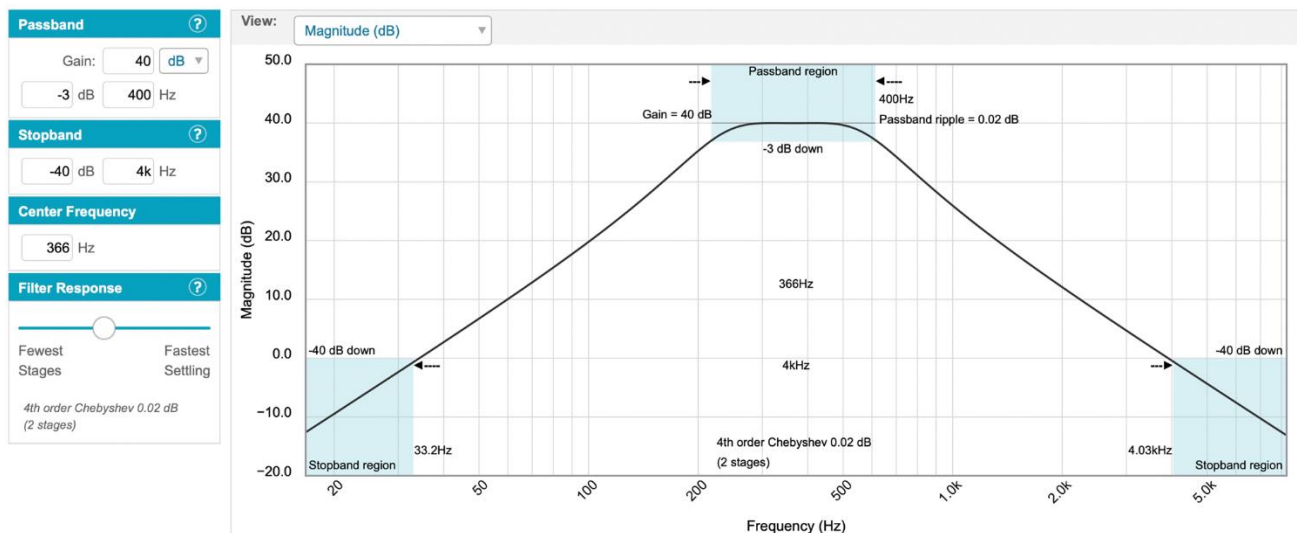
Appendix E: Radar



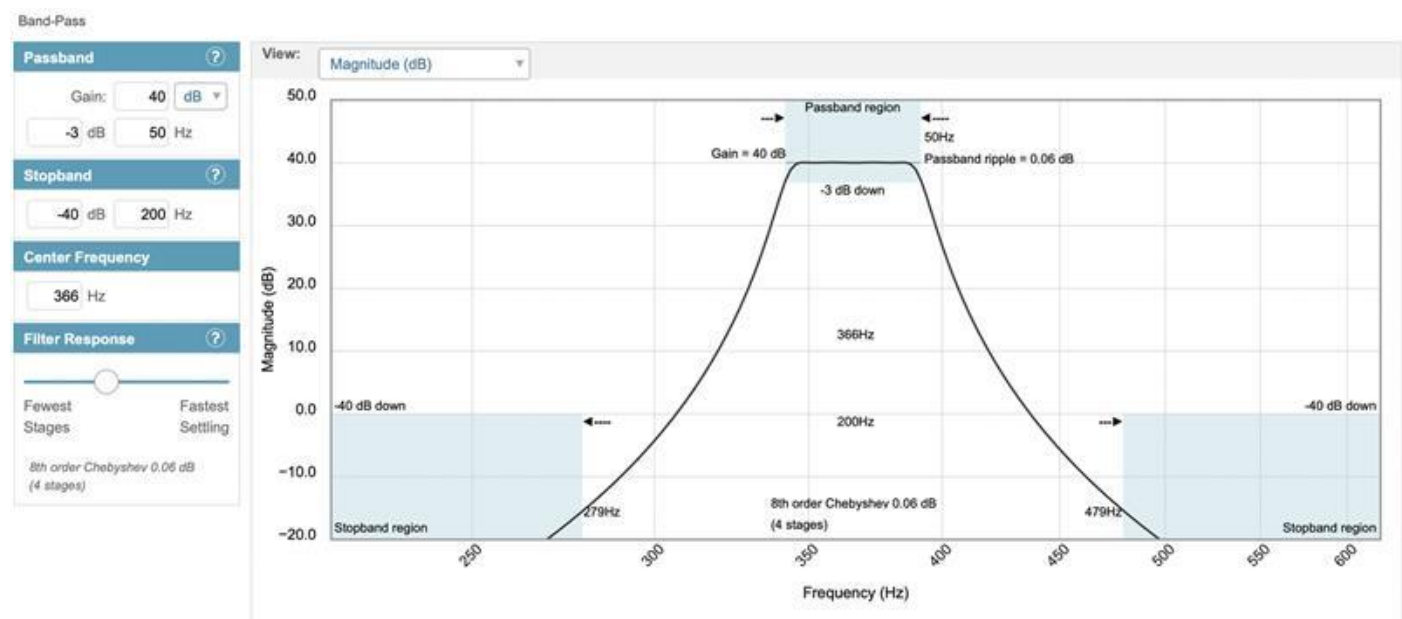
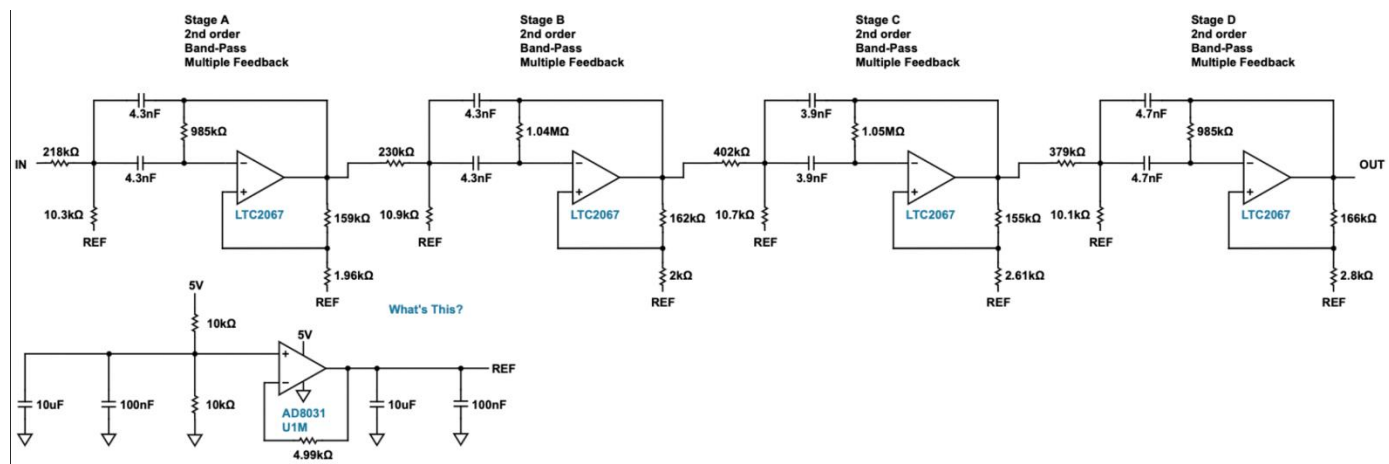
2: 2-stage Bandpass Filter Design and Transfer Function [\[REFERENCE NUMBER 2\]](#)



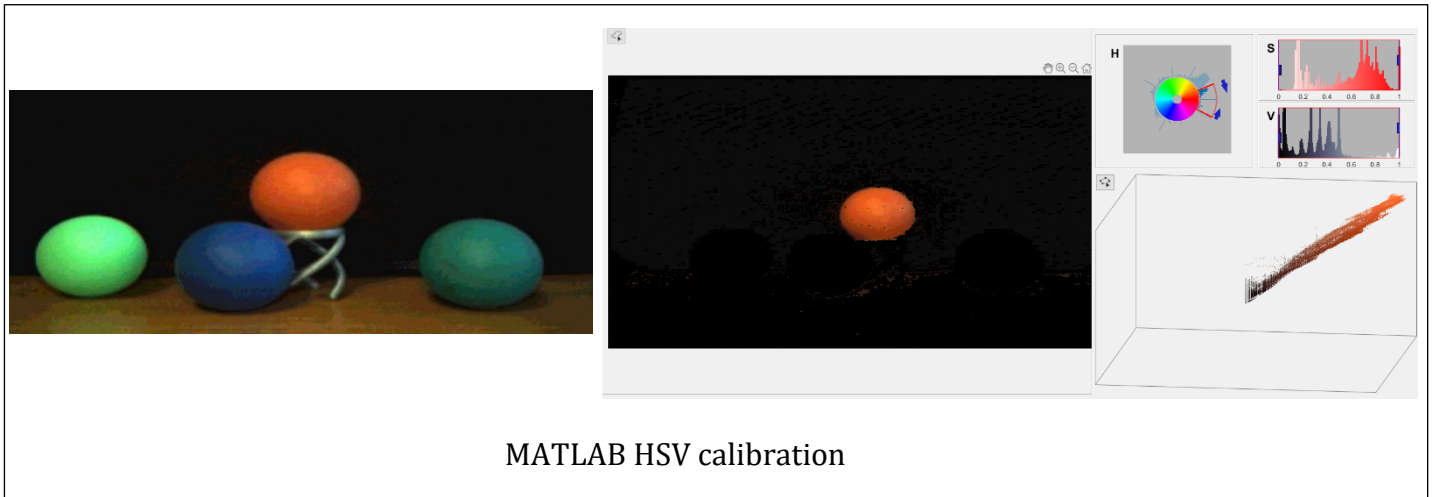
Band-Pass



3: 4-stage Bandpass Filter Design and Transfer Function REFERENCE NUMBER RADAR 2



Appendix F: Vision



```
if(byte_data_received == 1 | no_red_ball_counter > 3)begin //If we have already found a red ball
    outbuffer <= 0;
    r_ball_watching <= 0;
    distance_measure_active <= 0;
    if(byte_data_received == 1) begin
        r_ball_registered <= 1;
    end
end
end
```

Logic declaring when to set that the red ball has been registered on the map based on the value of ESP32 input

Distance	Xmin	Xmax	W
15	243	423	180
20	267	405	138
25	280	388	108
30	303	392	89
35	298	378	80
40	310	377	67
45	284	346	62
50	304	361	57
55	311	360	49
60	315	364	49

Experimental data gained when finding the focal length of the camera in a range of 20-60cm by measuring pixel width against distance from ball to rover