Imperial College London

2022 Second Year Summer Project

ELEC50008

Technical Report

—Group AJFB2—

Klio Balliu - 01706972

Brendon Ferra - 01760303

Muhammad Haaris Khan - 01926488

Pranav Madhusudhana - 01845668

Ali Orkun Ozkan - 01890200

Sivashanth Sathiyathasan - 01735397

Bryan Tan - 01861592

Submission Date: 24th June 2022

Submitted To: Dr. Adam Bouchaala

1 Introduction

The goal of the project is to create an autonomous Mars Rover that manoeuvres around an arena whilst detecting coloured balls, obstacles, and an underground fan.

As expected, most but not all tasks were able to be completed due to time constraints and are expanded upon in the conclusion. Project task separation was based on member specialties and interests, and tasks were set on timeframes of half-weeks, as shown in Figure 1.1. A key strategy which was found to be effective was having each team member spend time each week keeping up to date with other modules relevant to their course. For example, the member assigned to Radar spent time understanding developments in Energy. This was done to ensure that if a team member encountered large issues, another member could be assigned to help, without needing to be caught up on.

Although an initial design specification and materials were provided, modifications were made where deemed worthwhile, such as improving the wheels, or changing the radar's location to enhance the performance of the rover.

A Python simulation of the rover driving was also created, which enabled more effective planning for integration. This allowed for the finalisation of data formats, and for real data to be plugged into the simulation for testing whilst the other submodules were not finished yet.

Group 3 (AJFB2)	KLIO, ORKUN	Project Start:	23/05/2022																
	BRENDON, BRYAN	Project End	28/06/2022																
	PRANAV, SIVA,HAARIS																		
							23 to 25	26 to 29	30 to 1	2 to 5	6 to 8	9 to 12	13 to 15	16 to 19	20 to 22				
TASK	ASSIGNED TO	PROGRESS	START	END	WEEK HALFS		First	Second	First	Second	First	Second	First	Second	First				
Control/Rover/Project																			
Construct Rover	Klio	100%	23-May	25-May	1														
Inter-module communication	Bryan, Orkun	100%	24-May	17-Jun	5														
Write-up	Klio	100%	30-May	21-Jun	6														
Testing and Preparation	Whole team	100%	13-Jun	21-Jun	3														
Command																			
Architecture choices	Bryan, Brendon, Haaris	100%	23-May	14-Jun	5														
Server and rover comms	Orkun, Bryan	100%	24-May	17-Jun	3														
Front-end and Database	Brendon, Bryan	100%	30-May	23-May	4														
Simulation and automation	Haaris, Orkun, Bryan	50%	23-May	10-Jun	4														
Drive																			
Basic library drive functions	Siva, Orkun	100%	23-May	27-May	2														
Design control system	Siva, Bryan, Orkun	100%	30-May	10-Jun	4														
Tune values and test	Siva, Orkun	100%	07-Jun	21-Jun	5														
Bezier motion	Orkun	20%	30-May	17-Jun	2														
Energy																			
Characterise panel and battery	Pranav, Siva	100%	23-May	11-Jun	4														
Implement SMPS and control	Pranav	100%	30-May	10-Jun	4														
Implement relay	Klio, Pranav, Siva	100%	06-Jun	15-Jun	3														
Design and test MPPT	Pranav	100%	08-Jun	19-Jun	3														
Test complete system	Pranav, Klio, Siva	100%	13-Jun	20-Jun	3														
Radar																			
Design processing circuit	Klio	100%	25-May	06-Jun	5														
Test range	Klio	100%	07-Jun	15-Jun	3														
Vision																			
Edge Detection	Haaris, Brendon	100%	30-May	14-Jun	5														
Colour Detection	Haaris	100%	06-Jun	17-Jun	4														
Location & Bearing	Haaris, Brendon	100%	10-Jun	19-Jun	3														

Figure 1.1 – Gantt Chart

This project highlighted the importance of systems engineering. As shown in Figure 1.2, there is significant interconnection between the sub-modules. An initial structural diagram was created to help map how the sub-modules communicate and what type of information they share and was constantly revised. Effective communication between EEE and EIE members facilitated the implementation of the sub-modules into the structural diagram. It was deemed important as a group that each member have a significant understanding of each module's place within the system of the rover and its function. Although there may be little benefit to the progress made on each member's

module, this understanding would reduce the risk of miscommunication and non-compatibility between systems, saving time in the long run.

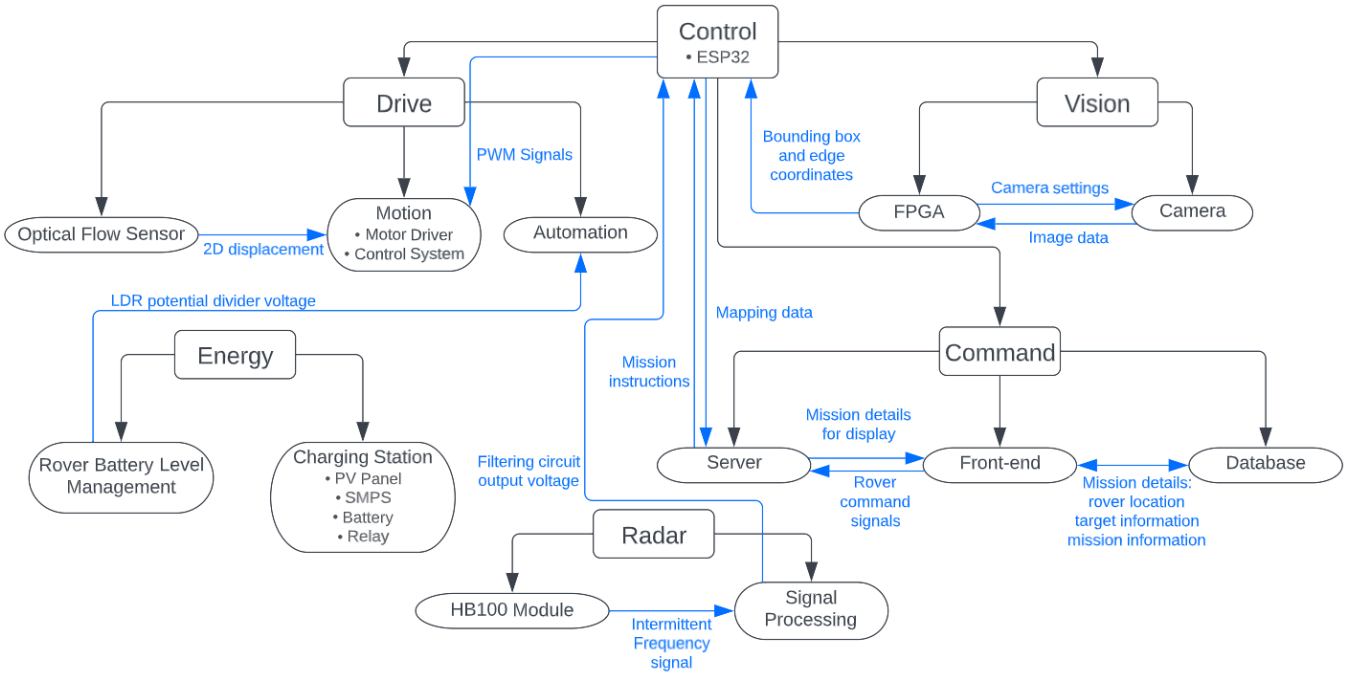


Figure 1.2 – Structural Diagram showing subsystems of the rover and inter-module communication

2 Control

The control subsystem acts as the root of all subsystems (Fig 1.2) and manages most of the inter-module communication. It is implemented on an ESP32 microcontroller (ESP). All operations take place on a single loop, multitasking was simulated by interleaving non-blocking function calls.

Drive: The closed loop control algorithm for rover motion is implemented on the ESP. The rover's current position and orientation are calculated from readings from the optical flow sensor, connected via SPI. On each loop iteration, the motor speeds are adjusted slightly to keep the rover's course.

Radar: The output of radar module is an analogue voltage signal which the ESP reads through an ADC pin. The fan is registered if the voltage signal exceeds a threshold.

Vision: The output of the image processing hardware (FPGA) has an output which can take the value of the co-ordinates of a bounding box or an edge. NIOS II sends this output to the ESP over UART.

Command: The ESP connects to LAN over Wi-Fi and communicates with the webserver over TCP. The rover's current position and any targets within its FOV are sent to the server every ~20ms. Non-blocking input from server to ESP was achieved by buffering the bytes received over Wi-Fi, and only parsing the data when the length of the buffer is large enough to include an entire TCP message. TCP was chosen for server communication because it provided reliable, in-order transfer of data. HTTP was also considered, but the execution time of each HTTP request was found to be too long, blocking the other operations in the main loop. There is a major delay in waiting for an empty HTTP response from the server, as well as additional overhead in constructing and parsing each message in the HTTP format. In a benchmark test, it was found that the mean time taken for an HTTP GET request from the ESP to the server is 112.4ms, while it was only 0.73ms for a TCP message (see Appendix A for testing method).

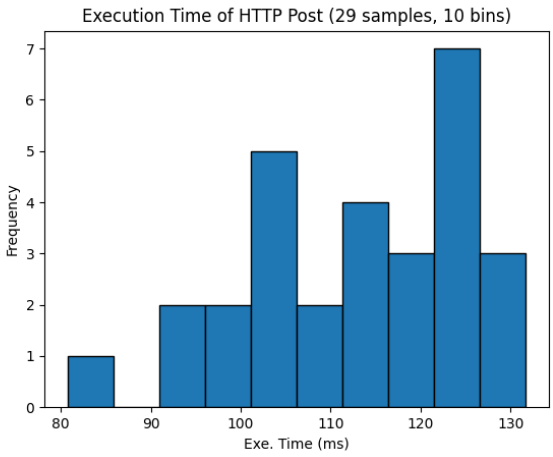


Figure 2.1 – Execution Time of HTTP Post

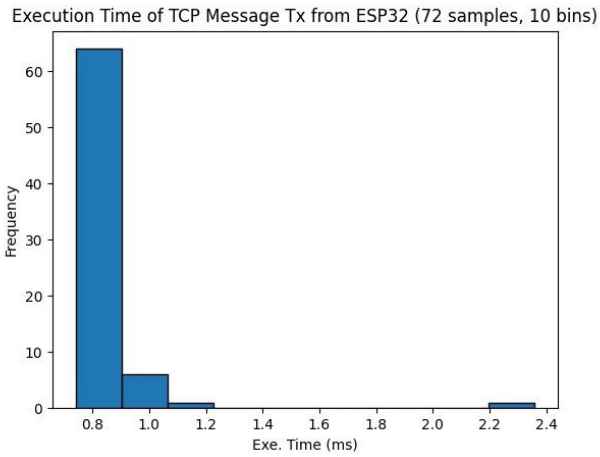


Figure 2.2 –Execution Time of TCP Message

3 Command

The command subsystem consists of a server and a webpage on the client. The webpage renders a live representation of the rover and the targets it has found within the arena. It allows the user to switch between manual and automated modes of operation. In manual mode the rover can be commanded to move to a specific location on the map by placing a waypoint. There is also a database of past missions which can be replayed in live time on a separate map.

The server facilitates rover-client communication and handles access to a cloud-hosted database. The server is hosted locally to reduce latency but can easily be migrated to AWS if necessary.

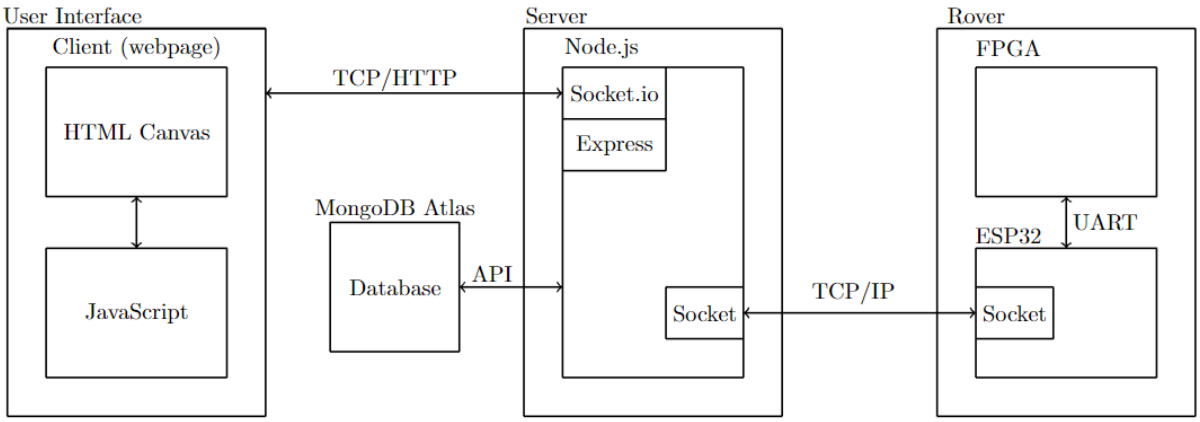


Figure 3.1 – System Architecture Diagram

3.1 Server-side Architecture

Web-framework – Node.js: This was chosen for the backend because it has native asynchronous programming features and useful libraries/APIs available. Express was used to handle URL routing and serve webpages to the client. The Socket.io, a TCP based API was used to provide fast, bi-directional communication between the server and client in the form of native JS data structures. Data from the rover and user commands could be transferred very quickly to the server over this link, allowing the implementation of real-time mapping of the arena on the webpage, and responsive user commands to the rover in manual mode. Python based frameworks such as Django, Flask and Fast API were also considered, but decided against because they are slower at serving web requests than Node.js. In a benchmark test the fastest Python framework Fast API served 560 HTTP requests per second, while Node.js served 840 HTTP requests [1]. While both rates ended up significantly exceeding the requirements for the project (the rover only sends to the server at 50 Hz) Node.js was chosen to allow for scalability in the future.

Database – MongoDB: This is a No-SQL, document-store database. An embedded model was used to store the data generated by the project: the database consists of only one collection and each document in the collection contains data pertaining to a single mission. Each document stores fields such as the total distance travelled during the mission, the final location of any targets found, as well as time series data of the rover's location on the map. Therefore, a No-SQL implementation was ideal: the data stored is non-relational, and most queries would be done to one specific document, such as retrieving the replay of a selected mission. Simple aggregation queries are also possible, such as calculating the overall success rate (targets found out of seven) of all past missions.

3.2 Front-end Architecture

User Interface: The webpage user interface was built with basic HTML/CSS/JavaScript, using some elements from Bootstrap. Front-end frameworks such as React and Flutter were considered, but they were decided to be unnecessarily complex for the scope of this project.

Map rendering - Design: The server sends data as a JSON object to the client which is processed by a client-side JavaScript program. This data is used to render a map on the webpage using the HTML Canvas element, a 2D graphics engine. Canvas was selected as it is fast and flexible; elements can be drawn very efficiently resulting in a high frame rate. An alternative server-side rendering approach was also considered: using the Python Matplotlib library to render and save a map as a JPG image, then posting it to the client over HTTP.

Map rendering – Testing: In a benchmark test, both rendering methods were tested on a sequence of randomly generated map data. A maximum of 27 FPS was achieved with the server-side method; the critical path was the image transmission to the client, taking an average of ~40ms. A maximum of 142.6 FPS was achieved with the client-side method (see Appendix B for details of testing method and results).

Figure 3.2 – Front-end rover command webpage

4 Drive

The drive subsystem consists of two parts: an autonomous driving algorithm which maps a path in the arena that ensures all targets are found while evading obstacles, and a control algorithm that moves the rover along said path.

4.1 Control System

Rover motion is produced by two brushed DC motors; PWM signals and H-bridges are used to control the rotation speed and direction respectively. An Optical Flow Sensor (OFS) measures displacement in the x and y directions relative to itself on each clock cycle. The ESP32 calculates the rover's

displacement and orientation relative to an initial position using the accumulation of these readings. These two elements form the basis for the closed loop control scheme of the rover's motion.

Figure 4.1 – Drive Control Scheme

Two types of rover motion were implemented: straight line movement and rotations. Straight line movement is facilitated by two controllers. The first is a distance controller which ensures the rover travels the correct distance. Error is defined as the difference of the y-displacement (OFS-relative) and the input setpoint. The dynamics of the system are sufficiently stable for a bang-bang controller to be used: both motors spin at the same constant speed until the error is within bounds, when the motors switch off. A second deviation controller is required to ensure the rover's path is straight; error is defined as the OFS-relative x-displacement (setpoint is always 0). The output of this controller adds a small offset signal on top of the base PWM values set by the distance controller (e.g., if the rover deviates right, slightly increase the speed of the right wheel and decrease the speed of the left as correction). A PID scheme was needed for deviation control: P term used as base, D term to dampen oscillations and I term to correct for steady-state errors such as wheel misalignment.

Rover rotation is similarly facilitated by two controllers. A P controller for angle ensures the rover rotates through the correct arclength, the error between current angle as well as destination angle is used to calculate a base PWM for both motors. A PID rotational deviation controller ensures that the centre of rotation is about the centre of the rover, defining error as OFS y-displacement, which is 0 for ideal circular motion.

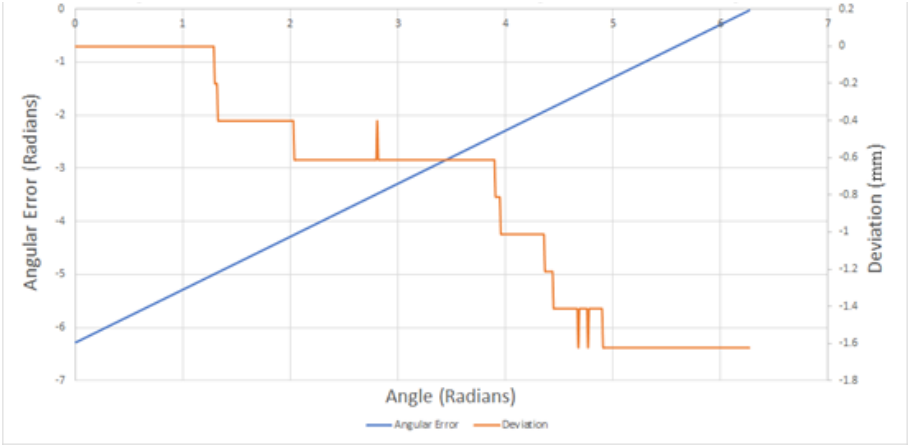


Figure 4.2 – Angular and rotation error against an angle

The P, I, and D coefficients of the control system were tuned when the rover was in its final constructed state to ensure no further physical changes would disturb the optimal PID coefficients. The resulting system produced accurate angle and distance control, with angle instructions having an uncertainty of ±1.1% and deviation error accumulated to 1.6mm after one rotation.

4.2 Error Mitigation

Dead reckoning error is defined as the error accumulated as a device attempts to calculate its position using previous values and sensor information.

To minimise this from accumulating, the OFS sample rate is set to a high frequency (100Hz). Due to this, the difference between readings is minimised, allowing utilisation of small angle approximations to determine the angle the rover is facing. As shown in equation 4.1, the angle travelling at is observed

to be the arclength measured divided by the rover's radius, the difference between the centre of rotation and wheels. Referencing equation 4.2, the temporary angle was found to be the current angle appended to the average change in angle measured. Referencing both equations 4.3 and 4.4, the rovers' exact position was computed by accumulating the magnitude of travel multiplied by the sine and cosine of the rover's temporary angle. Due to the automation script referenced in section 4.3 relying on 90° turns, the rotation was tuned to reach such angles, as discussed in section 4.1.

$$s = r \cdot \Theta \qquad \textit{Equation 4.1– Arclength formula}$$

$$\hat{\Theta} = \hat{\Theta} + \frac{dx}{2r} \qquad \textit{Equation 4.2 – Temporary angle formula Equation}$$

$$x = dy \cdot \cos(\hat{\Theta}) \qquad \textit{Equation 4.3 – Position formula 1}$$

$$y = dy \cdot \sin(\hat{\Theta}) \qquad \textit{Equation 4.4 – Position formula 2}$$

One drawback that arose from having such a high sample rate was the introduction of quantization errors. This caused the accuracy of the OFS's readings to be limited to 0.2mm. Although quantization error was detected, its influence was found to be negligible due to the scale of the rover.

4.3 Automation

The autonomous drive path decided on is illustrated in Figure 4.3. This path was decided on as it permits navigation of the whole arena, allowing all objects and obstacles to be identified. As the camera detection range is larger than the radar, the width between paths, *x*, is constrained by the radar sensor. In order the reduce the distance travelled, a wide path is used to cover the whole arena (shown in black), and if the underground fan has not been found at the end of the first path, END 1, the rover performs a second pass which covers the areas in between the first path (shown in blue), ending at the location END 2. The exact dimensions of the paths, *x* and *y*, were determined by testing, to ensure that paths do not cause collision with the wall due to deviation from the path, due to the inaccuracies of the OFS.

4.4 Obstacle avoidance

Obstacle avoidance is implemented by the method shown in Figure 4.4. If there is an obstacle in the rover's way, shown in blue, it will take the red or green path, depending on whether the rover is to the left or right of the obstacle. The total displacement *k* is determined by the obstacle's radius added to an avoidance margin *z*. For balls, the width is fixed; however, since the buildings are different sizes, widths need to be calculated using the spatial frequency and number of edges. Multiple object avoidance is possible through recursive function calls. In the case that the rover travels far off course due to avoidance, it will attempt to take a shortcut to the nearest section of the original path.

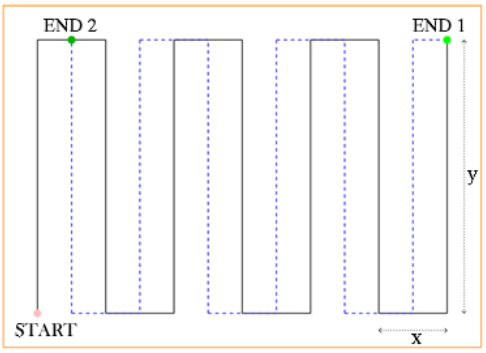


Figure 4.3 – Automation path of the rover

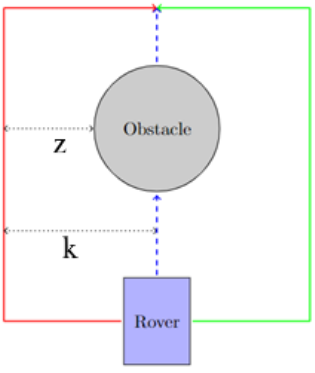


Figure 4.4 – Obstacle avoidance diagram

5 Vision

The goal of the vision system is to identify, in screen-space, the location of objects of interest (table tennis balls and striped buildings), using a camera connected to an FPGA. They are differentiated from

the background by their colour and edges, allowing the use of computer vision techniques to locate them. This is done on hardware as it offers reduced latency compared to server-side computation.

5.1 Blurring and Convolution

An important step for detecting colour is blurring, since the process eliminates high-frequency sensor noise, which alters bounding boxes.

The naïve approach to convolution is an $O(n^2)$ matrix multiplication for each pixel [2], however there are faster techniques available. Some kernels, such as Sobel, mean, and Gaussian, are separable into 1×n and n×1 vectors [2]. These can be convolved with the image in separate passes, yet still produce the same result, in only $O(n^2)$. This works due to gathering, where information in the horizontal direction is distributed to adjacent pixels, so that the column vector only needs to refer to one pixel per row during a convolution [2]. A further improvement of this is through the $O(1)$ method of accumulation (in place of convolution), where a moving average is maintained by subtracting the oldest pixel and adding the newest pixel [3]. A moving average can only be implemented for a mean blur, however multiple passes of a mean blur approximate a Gaussian blur through the Central Limit Theorem [3], permitting the utilisation of pipelining on an FPGA. The use of ring buffers was considered, but unlike in software, ring buffers are slower than regular shift registers in hardware.

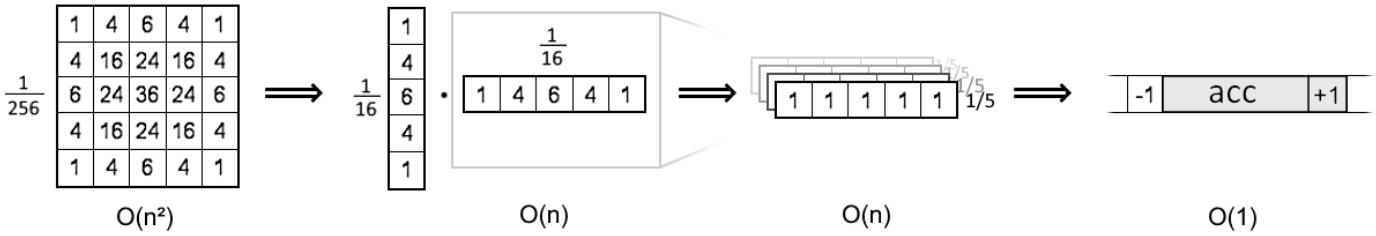


Figure 5.1 – Iterative improvements to the naïve convolution method

In practice, the nature of the noise is singular, saturated pixels, which are directionally independent. This means the vertical blur can be disregarded while still mitigating problematic pixels. Only using horizontal blur also means full rows aren't buffered, which was consuming a substantial portion of the device's resources. The final implemented kernel was a 1×4 mean blur passed over three times. (A width of n=4 allows the use of right shift instead of expensive divides.) The Gaussian σ achieved by the t=3 mean blur passes is ~0.87, which is sufficient.

$$\sigma = \sqrt{\frac{t \cdot n - t}{12}}$$

Equation 5.1 – Equivalent standard deviation for a 1×n mean blur performed 't' times [4]

5.2 Colour Spaces and Bounding Boxes

Pixel data arrives from the blur buffers in RGB format, however the HSV colour representation allows for more precise selection of which colour qualities are accepted, so a transform was performed from one colour space to the other. A white balance was performed by increasing the blue sensor gain. This counteracts the effect of the yellow lighting in the arena and utilises the full dynamic range for each colour. After the colours are identified, bounding boxes are drawn around the targets. Boxes were limited to the lower half of the screen-space, as well as rejecting screen edges, to reduce the chance of false identifications from the ceiling and wrap-around caused by blurring delay.

The boxes' locations in the screen-space are passed to the ESP32, and calculations are performed to calculate the distances and bearings of the targets relative to the rover. The locations of the objects in the arena can then be displayed on the map.

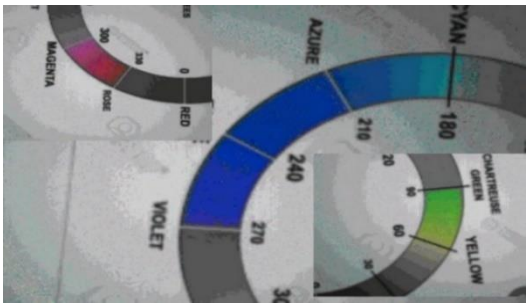


Figure 5.2 - Demonstration of HSV filtering

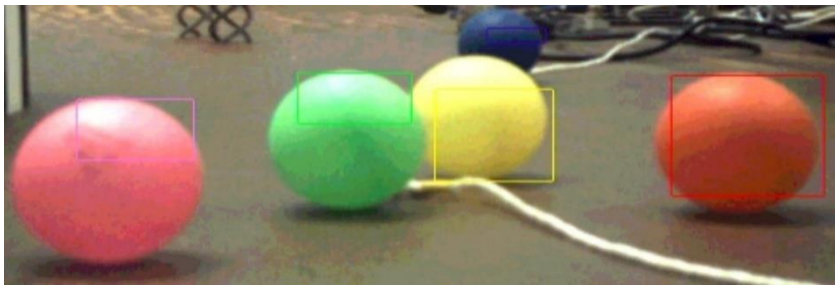


Figure 5.3 – Coloured bounding boxes

5.3 Edge Detection

The alien buildings are different widths; however, the stripes on the buildings are constant width. This makes their distance conducive to being measured from edges detected using a Sobel filter (image derivative) [2]. Through experimentation, it was found that for small kernels, a 1D horizontal kernel had the same effect as the 2D version but was less computationally expensive. The best kernel size was found by testing kernels on a spatial frequency grating. Due to destructive interference of overlapping edges, some larger kernels had blind spots at high frequencies but were able to detect low frequency (blurry) patterns better. Since the buildings had sharp edges, a 1×3 kernel was chosen.

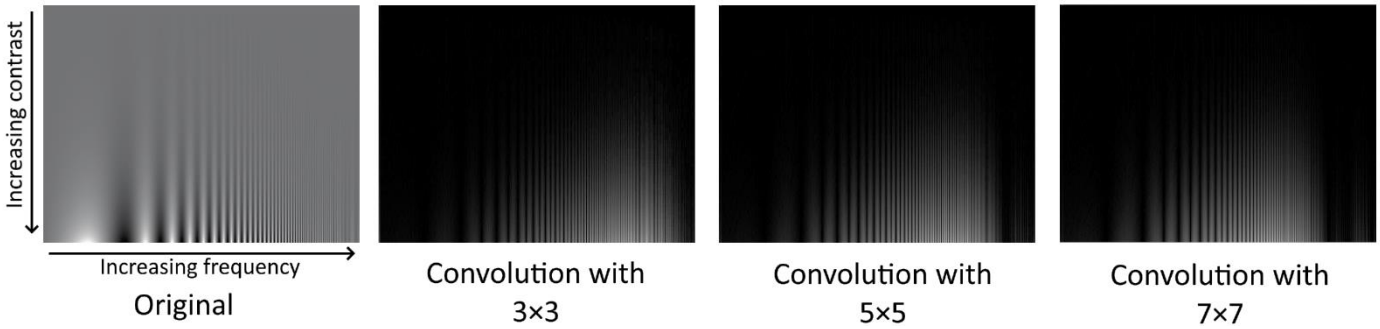


Figure 5.4 – Convolution with a sine grating [5] *to characterise the kernels' frequency responses*

Because each row of the image would produce a different answer for the edge's location (if it was skewed), the program would have to choose the best answer, which is an arbitrary choice. Therefore, this step is skipped by only analysing the central row for edges. The central row also has less interference from ball edges or edges above the arena curtain. Since the buildings are cylindrical, the first few and last few edges are discarded, as the widths of their stripes would be vastly different from the central stripes. The data is then sent to the ESP32. Because multiple buildings can be in view at once, one of the tasks is to determine how many buildings there are in the image. In this implementation, the data is discarded if more than two buildings are in the scene (it is highly unlikely that a building is never mapped due to this). To differentiate between cases where there are one or two buildings, various checks are done. The simplest is verifying that the building's calculated width does not exceed what we know about the largest building. Another technique is to check if the widths of the remaining left-most and right-most stripes are vastly different, which would signify two buildings where one is closer. This works because it is impossible for more than one building to be responsible for the remaining left-most stripe in the scene. The distances and widths are then calculated and sent to the server to be mapped.

This was completed using manual calibration. For instance, by measuring the bounding box size at various distances, a relationship $12000/d$ was found where d is the distance of the ping pong ball from the camera (see Appendix C for testing details). The angle and distance values were provided to the automation script which adjusts the path of the rover accordingly.

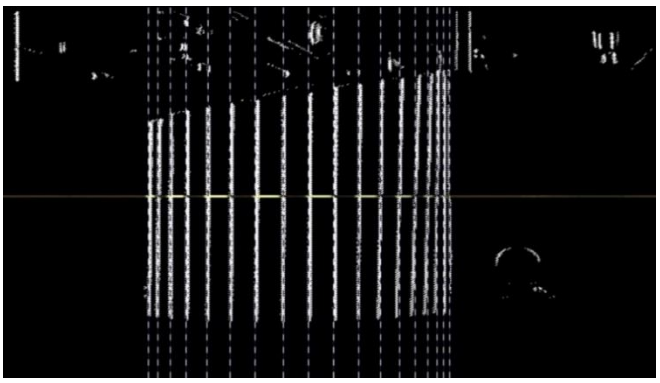


Figure 5.5 – Edge detection of sample obstacle

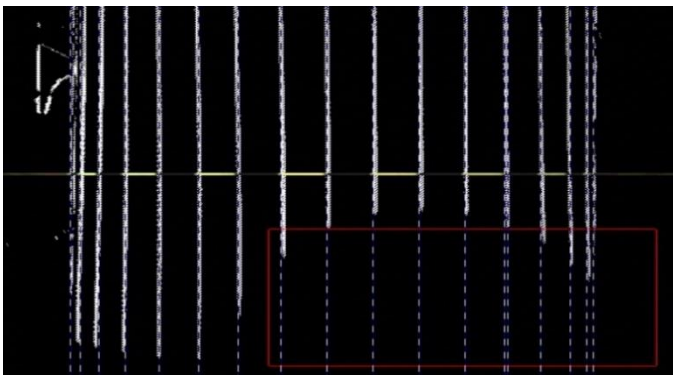


Figure 5.6 – Edge detection robustness in the presence of interfering red ball

6 Energy

The rover requires a charging station to recharge the battery. Solar panels have been used as the energy source must be renewable to ensure an extended and sustainable operational lifetime for the rover. To make the charging station as effective and efficient as possible, extensive testing was undertaken to produce optimal voltage and current control, by using Switch Mode Power Supply (SMPS) modules and optimal solar panel arrangement. The electrical characteristics of the battery were also considered to ensure safe and efficient charging. The energy subsystem is additionally responsible for managing the current battery level of the rover's power bank. The overall charging station diagram is shown below in Figure 6.1.

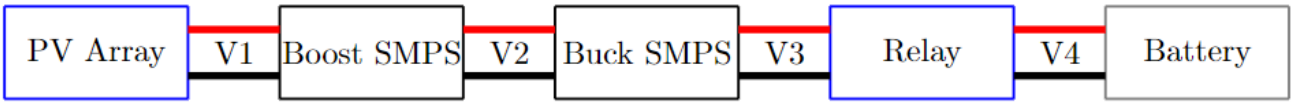


Figure 6.1 - Charging station system diagram

6.1 PV Array and Battery Characterisation/Design

The first task was to characterise the behaviour of four solar panels connected in all possible configurations: four in series, four in parallel, and two sets of two panels in parallel, then connected in series. This allowed for the region of the Maximum Power Point (MPP) of the 3 arrangements to be identified, and for planning of the use of SMPS systems in the charging station. The configuration of four panels in series was decided against as the MPP was above 16V. This would require a significant voltage drop through a Buck SMPS to reach below the maximum voltage of 5.2V needed to charge the battery. Furthermore, the average duty cycle during operation would be very low, such that the resolution of any change in duty cycle would also be low, resulting in control inaccuracies. The provided SMPS in Buck mode cannot handle the voltage required, due to PMOS gate limitations above 8V. Moreover, using multiple cascaded Buck SMPS systems would impact efficiency negatively.

After further experimentation, two sets of two panels in parallel, connected in series, was also rejected for the same reason. As shown in Figure 6.2, even if the Boost SMPS was operating as a voltage follower, the MPP would be outside the operating region of the SMPS systems. This meant that regardless of whether the process of Maximum Power Point Tracking (MPPT) was implemented on a Buck or Boost SMPS, the active region of the SMPS system would be below the MPP of the panels, as the Buck SMPS will always be needed to reduce the voltage to a range that can charge the battery.

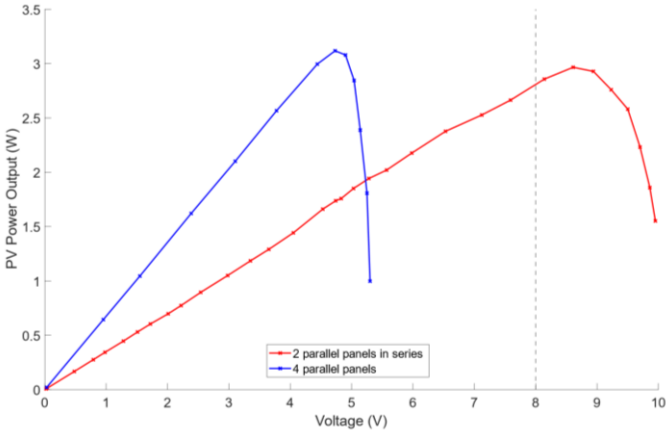


Figure 6.2 – PV array characterisation of relevant configurations

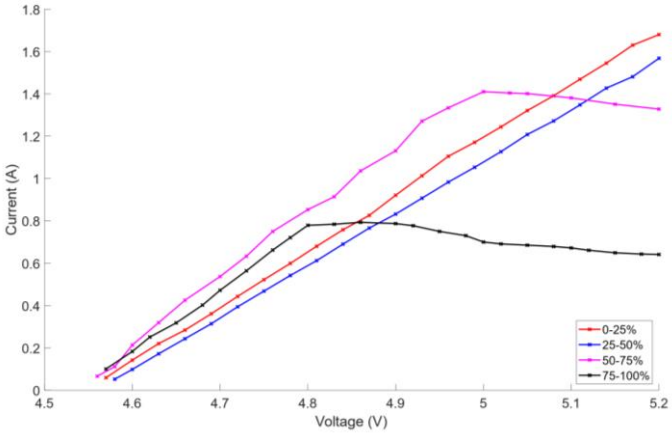


Figure 6.3 – Battery characterisation at quartiles of charge level

The panels provided to us were sub-optimal, as characterisation in Figure 6.2 shows a maximum power produced of 3.2W; however, this had no impact on the efficiency of the entire system. Series panel connections and performance degradation due to partial shading was also considered. To avoid the impact of partial shading, bypass diodes can be used to provide a low-resistance pathway for any current generated by a panel, while other panels in the system are shaded. However, this was not necessary as the configuration chosen was four PV panels connected in parallel. In this arrangement, the MPP would be within the active region of the SMPS system and would not require any additional protection from shading. From characterisation readings, it was found that the voltage output from the chosen configuration was slightly more unstable than the other two arrangements. This was counteracted by fast loop MPPT algorithms correcting instability, as well as forcing the panels to draw the maximum power, as described further in Chapter 6.2.

The battery was then characterised at each quartile of stored charge, shown in Figure 6.3. From observation of this data, it was determined that the battery voltage must be between 4.6V and 5.2V. The battery drew as much current as possible to charge and its behaviour varied depending on its charge level. It was concluded that the battery determined the behaviour of the entire charging system, as shown in Figure 6.7. The battery always acts to increase the charging power, which was a crucial factor contributing to the charging system design, elaborated on in Chapter 6.2.

6.2 SMPS Implementation and Charging Station

A significant role of the charging station control system was to ensure the panel voltage was always biased at its MPP, within the constraints. To implement MPPT, a 'perturb and observe' (P&O) algorithm was used on the Boost SMPS, which measures input voltage and current to calculate power before and after making a change in duty cycle. If power produced by the panel increases as a result of this duty cycle change, this change is repeated, moving up the MPPT curve, until the power starts to decrease. This will cause traversal of the MPPT curve in the opposite direction – an 'over-the-hill' process. Bypass diodes were not required in a '4 in parallel' combination; as a result, there was no need to avoid any decoy MPP peaks. The operation of the MPPT algorithm is shown in Figure 6.4 on a wider scale, with the input and output powers clearly following each other as expected in the P&O algorithm. Figure 6.4 shows that the MPPT algorithm is capable of finding the MPP of the panels by rising up the characterisation curve and settling at the MPP. The Boost SMPS was used for the MPPT as it allowed voltages below 4.6V to be brought up into the range that can charge the battery, which was a limitation of the Buck SMPS due to its restricted operating range. This produced the system shown in Figure 6.1. Due to the linear nature of the Buck SMPS input-output relationship to duty cycle,

controlling the voltage at V2 was realised as the optimal way to maintain an output voltage (V3) within the battery's operating region.

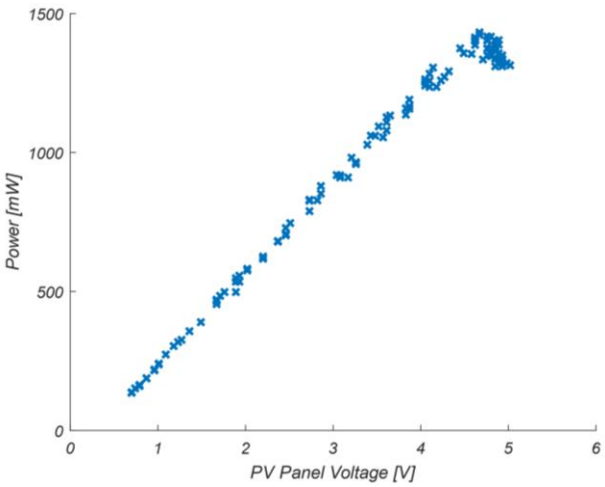


Figure 6.4 – The over the hill character of the MPPT algorithm

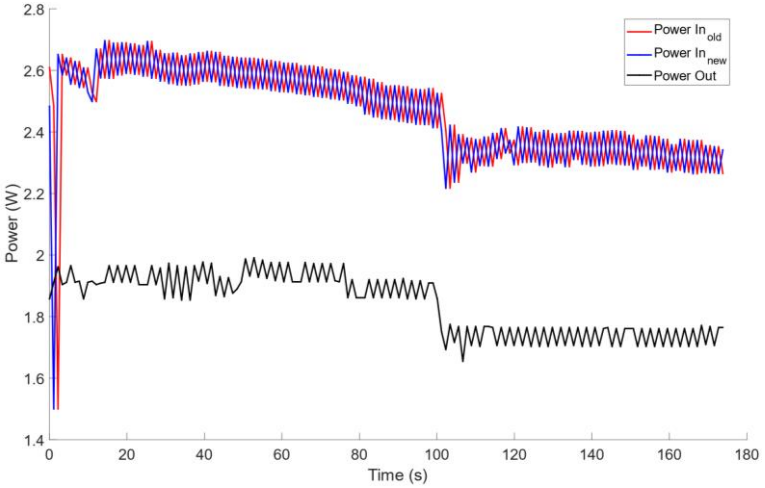


Figure 6.5 – Demonstration of MPPT implementation and evolution of the power point across PV array

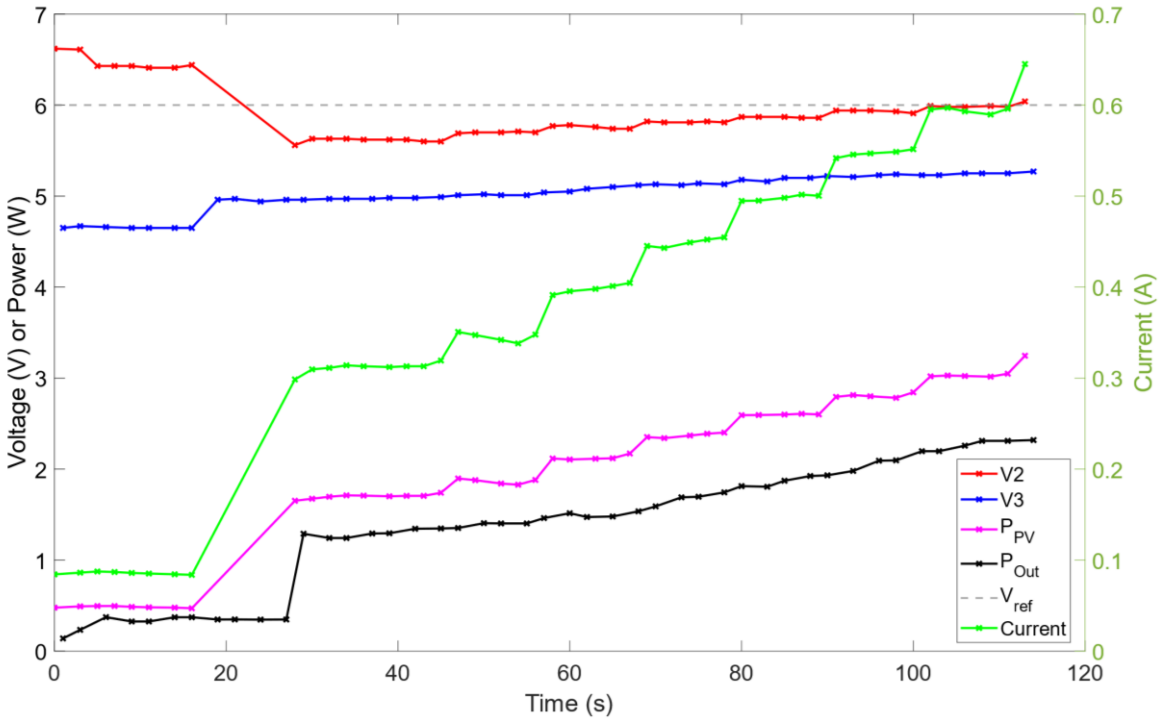


Figure 6.6 – The overall operation of the SMPS system, showing V2, V3, Current and Powers interacting

One major problem was the design choices surrounding this system. The initial idea was to force V1 at a certain voltage using MPPT, giving V2, and forcing the voltage at V3 to be within the battery's nominal operating range using a PID controller. This however produced a maximum current to the battery of less than 25mA, which was unsatisfactory for battery charging.

Upon further investigation, it became apparent that this was a result of a lack of oscillation slack at V2, and a result at V1 and V3. The battery's power circuitry was understood to be forcing the maximum possible current from the system. This is clearly shown in Figure 6.7, where the battery is abruptly connected while the system is running. It immediately draws significant levels of power from the PV panels and dictates V1, V2 and V3; this is unlike the previous line of thought where the PV panel was the driving factor of the system. This process cannot happen effectively if the voltage at which it is operating at is fixed constantly. Voltages are needed to oscillate with a sufficiently large amplitude in

order to draw more current from a certain voltage that it is biased at, effectively giving the battery a larger operating voltage range. Thus, after attempting bang-bang control, a P controller was used to maintain V2 between 6V and 7V, with an intentional error implemented to cause the battery to draw far more current: this range is a consequence of the duty cycle based mapping between V2 and V3. The battery always works within its charging region, which is why the plateaus shown in Figure 6.3 at high powers fail to pose an issue to our system.

The overall principle is illustrated in the final working system, which produced system operation as shown in Figure 6.6. As V2 (represented by the red line) oscillates and approaches 6V, the battery is able to draw more current (represented by the green line) until it eventually stabilises. This is happening while the duty cycle of the buck is close to 0.99: this means that V3 (Blue) increases until 5.2V, leading to an overall increase in power drawn from the PV panels and consequently supplied to the battery.

In the preliminary design, the P controller at V2 was implemented on the Buck SMPS and immediately was found to draw more than 200mA into the battery, an improvement by a factor of 10. The system however was too unstable. If V2 was too small, the duty cycle of the buck would increase. V3 and I3 would therefore increase, in turn bringing up I2 and V2. Frequently, if the oscillation above the 6V reference at V2 was too great, the voltage at V3 would be over 5.2V, causing the relay to disconnect the battery, causing significant undershoot due to the abrupt drop of current and voltage values. This action is overcorrected, causing the relay to switch off again, leading to rapid relay switching that the system was not recovering from.

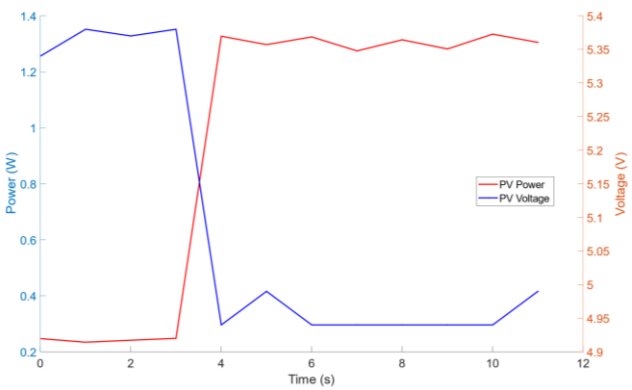


Figure 6.7 – The power drawn from the solar panels before and after the battery is connected

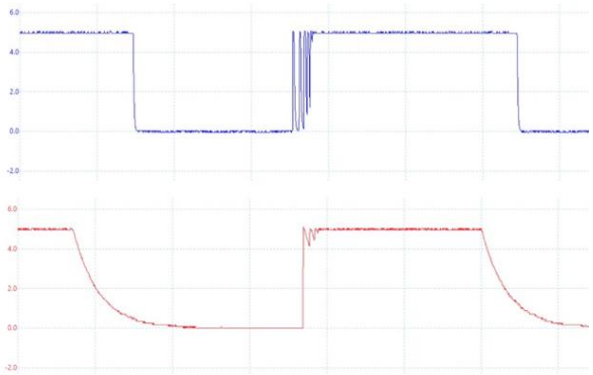


Figure 6.8 - Battery protection relay output voltage before (top) and after (bottom) snubber circuit implementation

The proposed solution was to implement the saturation algorithm, which limited the maximum and minimum duty cycle at which the Buck SMPS could operate at, with variables being passed to the function for maximum and minimum duty cycle. Maximum duty cycle was set as *5.2/V2,* and minimum duty cycle as *4.6/V2*. This allowed for sufficiently restrained oscillations at V2, such that V3 would never go out of range. However, this posed further problems as the changes in duty cycle were not large enough at the extremes, and output voltage would get stuck at either extreme. A term was appended to the minimum and maximum voltages that added on a value proportional to the error between the output voltage, 5.2V or 4.6V respectively. Once implemented, this eliminated the rapid and uncontrolled switching, enabling smooth oscillation. The code is provided in Appendix D.

The role of the relay is to act as a failsafe if the voltage at the output of the Buck SMPS is above 5.2V, disconnecting the battery in such an event. As seen in Figure 6.8, it was observed that the relay oscillated during the transition between connected and disconnected states, which was a common phenomenon. To solve this, a 1nF capacitor and 1kΩ resistor were connected in series across the

relay, known as a snubber circuit, with the resultant smoothed waveform also shown in Figure 6.8. To implement reverse polarity protection, a Schottky diode is connected in reverse across the terminals of the battery. The 1N5822 was chosen due to its high current load capability and low reverse leakage current. Sufficient bandwidth separation was enforced between the three looping systems: MPPT at 1Hz, Buck V2 regulation at 100 Hz, and relay operation at 1kHz.

Overall, from the provided panels which gave a maximum output of 3.1W during characterisation, a maximum of 2.47W was produced at the system to charge the battery, from an input of 3.05W from the solar panels; this resulted in an overall peak system efficiency of 82%. The system efficiency remained around 80% during typical operation. Figure 6.4 shows the output power stabilising (performed in evening sunlight) with the perturb and observe algorithm appropriately illustrated. Power drawn to the battery is working within the 4.6 - 5.2V range at the battery, however the current level is limited by how much sunlight is drawn. Generally, even at sunset, 1.5W out of 3.1W was being drawn by the PV panels.

6.3 Rover Charge Level and Range Estimation

Our rover is instructed to return to a set waypoint, which represents the charging station, when it detects that the battery is lower than 25% when the battery needs replacing. At this level, only one indicator light is lit. To detect this, a light-dependent resistor (LDR) was placed near the LED indicators in a custom enclosure. The LED which indicates the minimum charge level was covered with electrical tape, such that when all other LEDs turn off, the resistance of the LDR increases greatly. The LDR formed part of a potential divider connected across the battery voltage, with the voltage being sent to an analogue pin on the ESP32. A voltage lower than a certain threshold signified low battery level and the rover is immediately sent an instruction to return home.

Rover range estimation was also implemented. The rover performed straight and rotate functions for a set amount of time, after which the charge level was checked by cross-referencing with the battery characteristic graphs, and thus the energy depleted from the battery was found. This allowed for range prediction to forecast how far the rover could go, and when it needs to return to the charging station. Based on linear charge depletion over time, it was calculated that the rover can last a total of approximately 230 minutes on a full charge, when running at maximum speed.

7 Radar

The purpose of the radar module is to detect an underground fan, using the HB100 sensor. The output 'IF' on the HB100 module outputs the change in frequency produced by the Doppler effect due to the rotating targets. However, the signal is very weak, around 10 mV, with a DC offset. Thus, the signal amplification and filtering circuit in Figure 7.1 is used to ensure the signal can be identified by the ESP32. The input to the ESP32 must be a voltage which represents if the target is detected.

The MCP6002 was selected over other op-amps available in the lab such as the TL072 due to its rail-to-rail input/output.

Although the TL072 has a gain-bandwidth product of 3 MHz [6], while the MCP6002 has a gain-bandwidth product of 1 MHz [7], the low-frequency nature of the target signature and small operating gain used means the gain-bandwidth product of the operational amplifier has no impact on the choice.

7.1 Signal Processing

Due to the already heavy computational load on the ESP32, the decision was made to perform as much of the signal processing using analogue methods, ruling out the use of algorithms such as FFT to isolate the target signature. The analogue signal processing circuit is comprised of three stages, shown in Figure 7.1. Stage 1 removes DC offset and amplifies the AC component of the signal from the radar output significantly. It uses a voltage reference to provide a stable 2.5V, whilst removing all battery noise. Stage 2 implements the active band-pass filter. Stage 3 is another amplifier stage which adjusts

the DC offset and AC amplification to best fit the range of the analogue pin on the ESP32, while ensuring the maximum voltage at the output of Stage 4 does not exceed 3.3V. Stage 4 is a peak detector which outputs a DC voltage which increases based on the proximity of the radar.

To create Stage 2, an online filter designer [8] was used to produce an active bandpass filter to isolate the signal arising from the target signature. The bandpass filter has unity gain to allow straightforward adjustment of total gain using the first stage, allowing simple adjustment of gain based on testing. Since the fans output a Doppler frequency of 366 Hz, the centre frequency of the filter is set as such to isolate the target signature frequency. The stopband gain was set as -40dB outside of a bandwidth of 1000 Hz from the centre frequency, as it was found to be sufficient for cluster rejection while allowing implementation in only two stages, due to the space constraint on the rover's breadboard. This is because the main sources of unwanted signals arise from the rover's motion, and high frequency noise from the battery and components. A Butterworth filter was chosen to ensure the constant gain across the passband. Also, the passband bandwidth was set as 50 Hz to account for component tolerances and changes in target Doppler frequency, due to rover motion.

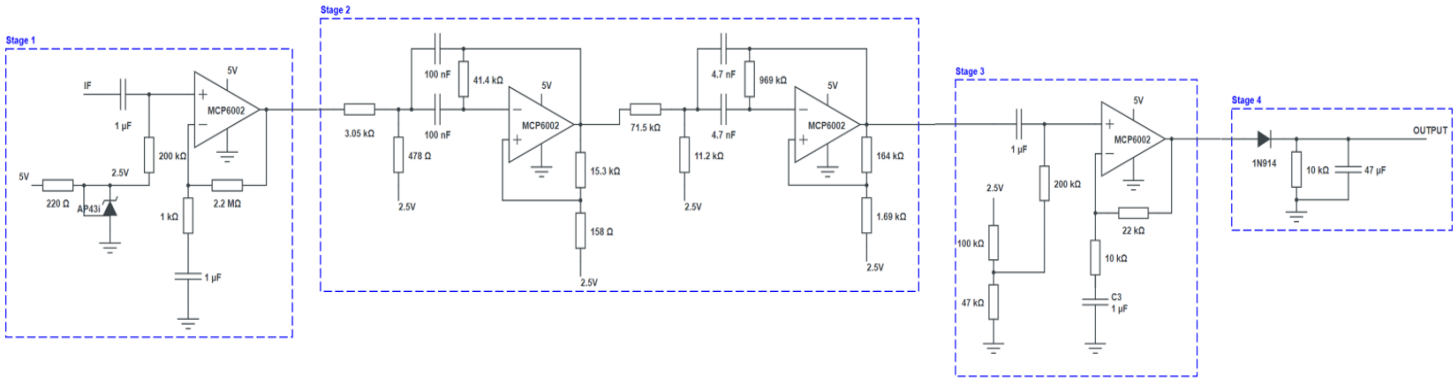


Figure 7.1 – Signal Filtering Circuit

7.2 Target Detection and Radar Range

Equation 7.1 shows the relationship which governs how the signal power at the output of the radar module is dependent on the target's distance from the radar, as well as the target and radar characteristics.

$$P_r = \frac{\eta^2 A^2 \sigma}{4\pi\lambda^2 R^4} P_t$$

Equation 7.1 – Radar Range Equation

From this, the important effect of distance can be observed—the received power scales with distance as $1/R^4$. This requires high transmitting power and a high-gain antenna to achieve large detection distances. However, the power and gain are limited by the radar module used. Therefore, the radar has a limited range, which has to be mitigated by the path taken by the rover around the arena.

The effective area, A, is dependent on the value of directivity at the location of the target relative to the radar. Therefore, the module must be mounted in a manner that ensures the reflector passes through the radar's main lobe. The method to ensure this occurs is described in Chapter 7.3.

7.3 Implementation and Performance

The initial idea to implement the radar detection was to create a 'heat map' overlay on the map, which would be continuously updated with the signal processor output voltage, corresponding to a reflector being near the rover, and the location of the largest voltage value would be the location of the reflectors. However, this was found to be computationally expensive, and the poor sensitivity, range, and accuracy of the module made this method unusable.

Instead to implement the radar subsystem within the Mars Rover, an ESP32 analogue pin is used as an input. Once a voltage above a set threshold, determined by testing, was passed, a marker is placed on the map, displaying that the reflector is in that position. Using the datasheet [9], the radiation patterns show that the reflector has the largest effective area directly along the axis of the HB100 module, and this was confirmed by testing. Hence, the radar is mounted parallel to the arena ground. A modification was made to reduce the distance from the radar module to the ground, increasing range.

Overall, the radar subsystem allowed the detection of reflector targets up to a radius of 0.2 metres. This hampers the automation path choices described in Chapter 3.3, to ensure the target is detected.

8 Conclusion

Several improvements could potentially be implemented if time and resources allowed. For example, the radar signal processing circuit could instead be made on a custom PCB. This would improve performance by increasing reliability and reducing circuit parasitic behaviour due to the breadboard. Radar performance could also be improved by using a radar with a larger gain, increasing detection range.

An attempt was made to implement a livestream from the camera on the front-end. Real-Time Messaging Protocol (RTMP) was considered to send data from the server to the website by converting the video output to a usable digital format. However, this is limited by the 9-bit UART data frame size and maximum 5 Mbps data rate; therefore, Serial Peripheral Interface (SPI) would be required.

Another improvement would be to complete the top-down automation algorithm in the Python simulation, illustrated in Figure 8.1. Through ray-casting in an internal model of the arena (on the server), paths without obstacles are found, and smooth, Bezier-like motion is iteratively calculated to reach the next waypoint, as shown in Figure 8.2. A complex algorithm for radar detection is also used, utilising weighted probability densities. The simulation also supports ultrasound for more accurate object avoidance.

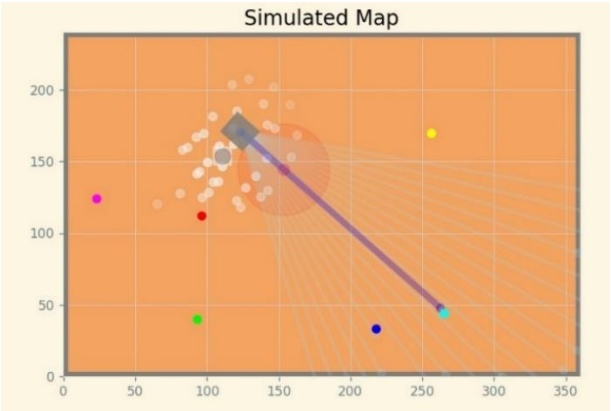


Figure 8.1 – Python Simulation showing ray-casting and radar probabilities

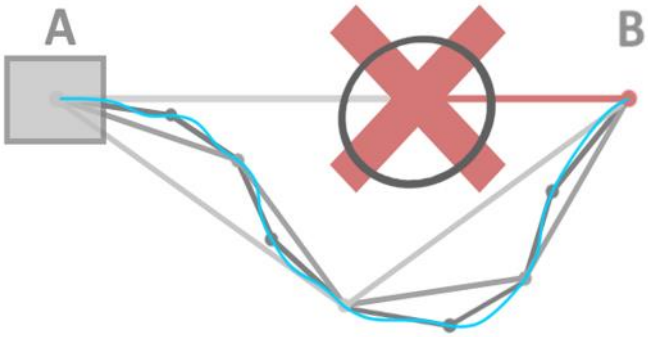


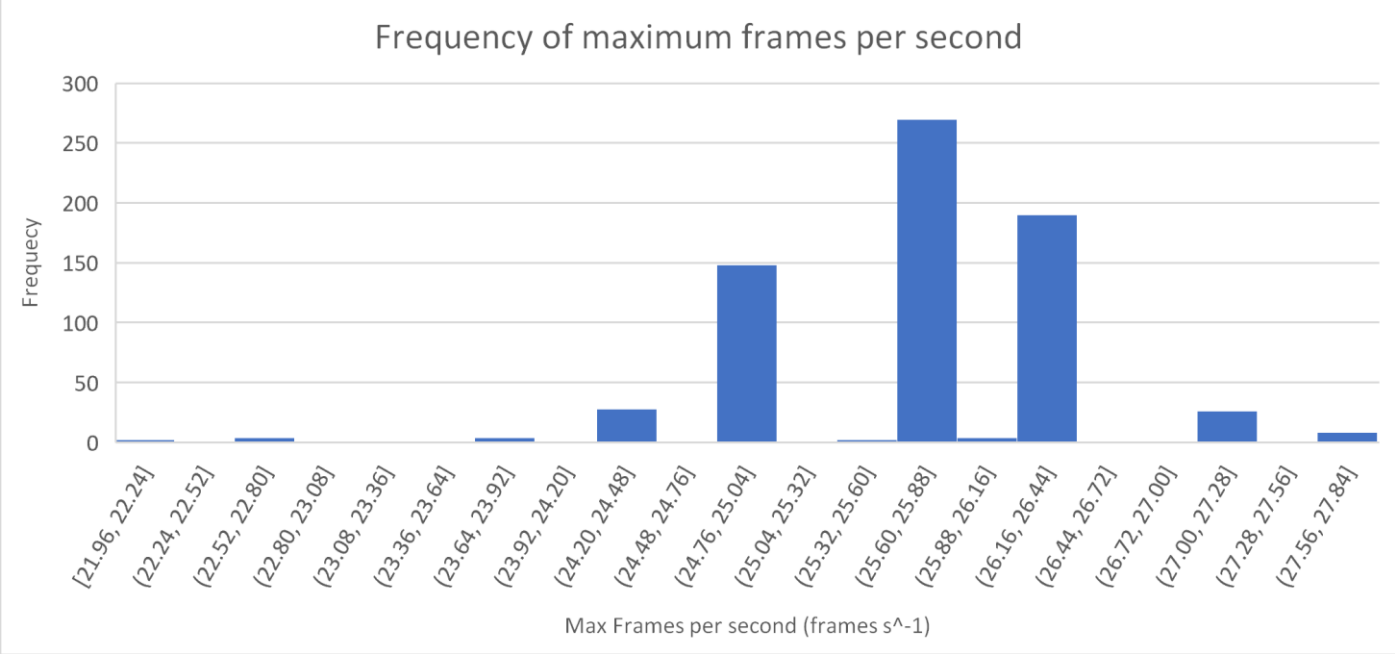
Figure 8.2 – Macro-level waypoints shown in grey, with micro-level PID control overlayed in blue

9 Appendix

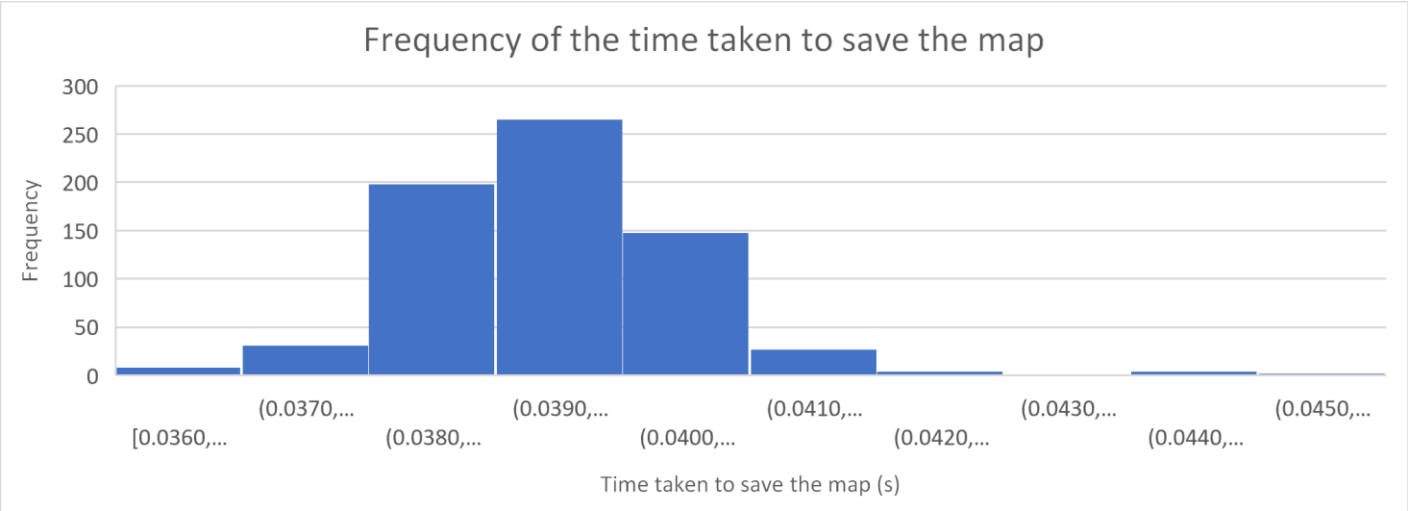
Appendix A:

HTTP/TCP Benchmark Test: Time taken for ESP to send 10 messages (each 20 bytes long) to the server was measured using the Arduino micros() function, and the mean time to send one message by dividing by 10. Over 29 samples, the average time taken for an HTTP post was 112.4ms. The HTTPClient library was used. Over 72 samples, the average time taken for an TCP message was 0.73ms. The WifiClient library was used.

Appendix B:
Measuring FPS of client-side vs server-side map rendering: the total time taken for 20 frames to be rendered was measured using the JavaScript date() function and logged to the console. Over 687 samples, the server-side rendering method achieved a minimum of 22 FPS and a maximum of 27 FPS. Over y samples, the client-side rendering method achieved a maximum of 142.6 FPS.

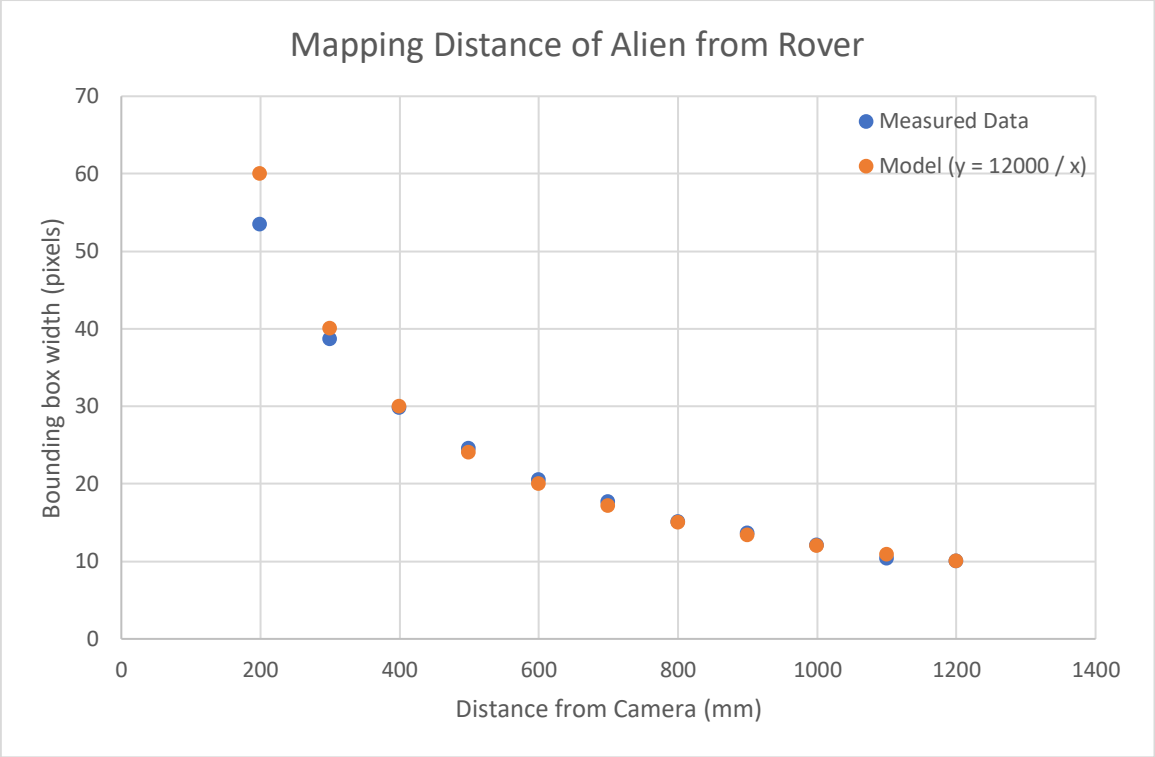


Measuring image transmission time from server to client over HTTP: The image was transferred from server to client using the Fetch API. The client JS program measured the time between the fetch request and the arrival of the image using the JS date() function. Over x samples, the average time of transmission was found to be 0.043ms. The cache functionality of the API was toggled off to ensure that the image was sent from the server every time.



Appendix C:

Determining Distance of Obstacle from Rover: The relationship $d = \frac{k}{w}$ (d = distance from rover, w = width of bounding box) was empirically determined. The image processor was tuned to detect only a green ball against a white background. The ball was placed at 100 mm increments from the rover camera, and the mean bounding box width returned by the image processor was noted.



Appendix D:

```
float saturation( float sat_input, float uplim, float lowlim) { // Saturation function
    if (sat_input > uplim) sat_input = uplim;
    else if (sat_input < lowlim ) sat_input = lowlim;
    else;
    return sat_input;
  }
//va is the input voltage to the buck, vb is the output voltage from the buck.
errorv = 6 - va;

      open_loop = open_loop + k_p*errorv;

      min_duty_error = 4.6 - vb;

      min_duty = 4.6/va +min_duty_error*0.2;

      max_duty_error = 4.9 - vb;

      max_duty = 5/va + max_duty_error*0.1;

    pwm_modulate(open_loop); // and send it out

    open_loop = saturation(open_loop, max_duty, min_duty);

    open_loop = saturation(open_loop, 0.99, 0.01);
```

Appendix E:
https://github.com/OrkunAliOzkan/MarsRover

10 References

[1] N. Akki, “Express vs FastAPI,” 2021. [Online]. Available: https://gochronicles.com/benchmark-restful-apis/#:~:text=Express%20(NodeJS)%20is%20~1.5,Python%20or%20NodeJS%20based%20frameworks..

[2] J. Johnson, “Image filtering II,” 2020. [Online]. Available: https://web.eecs.umich.edu/~justincj/slides/eecs442/winter2020/442_WI2020_lecture08.pdf.

[3] W. Jarosz, “Fast Image Convolutions,” 2001. [Online]. Available: http://www.acm.uiuc.edu/siggraph/workshops/wjarosz_convolution_2001.pdf.

[4] P. Kovesi, “Fast Almost-Gaussian Filtering,” [Online]. Available: https://www.peterkovesi.com/papers/FastGaussianSmoothing.pdf.

[5] J. Pillow, “Vision: From Eye to Brain,” 2015. [Online]. Available: http://pillowlab.princeton.edu/teaching/sp2015/slides/Lec07_EyeToBrain_Chap3B.pdf.

[6] R. T, “TL07xx Low-Noise FET-Input Operational Amplifiers datasheet,” 2014. [Online]. Available: https://www.ti.com/lit/ds/symlink/tl074h.pdf?ts=1655379915115&ref_url=https%253A%252F%252Fwww.google.com%252F.

[7] microchip.com, “MCP6001/1R/1U/2/4 - 1 MHz, Low-Power Op Amp,” 2020. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MSLD/ProductDocuments/DataSheets/MCP6001-1R-1U-2-4-1-MHz-Low-Power-Op-Amp-DS20001733L.pdf.

[8] Analog Devices, “Filter Wizard,” [Online]. Available: https://tools.analog.com/en/filterwizard/.

[9] Agilsense, “HB100 Datasheet,” [Online]. Available: https://www.limpkin.fr/public/HB100/HB100_Microwave_Sensor_Module_Datasheet.pdf.