Imperial College London

WALL-EE MARS ROVER – GROUP 16

ELEC50008 : ENGINEERING DESIGN PROJECT

DATE : 23 JUNE 2022

LECTURER : ADAM BOUCHAALA

Shaheen Amin
Shaanuka Gunaratne
Indraneel Dulange
Angelo Agathangelou
Joao Pereira
Lau Chun
Liam Browne

Imperial College London

Contents

Imperial College London

Introduction

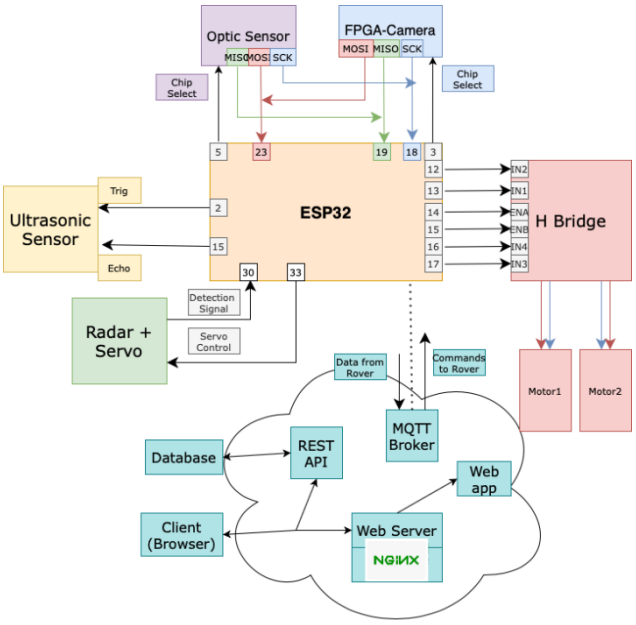
The main goal going into this project was to succeed in making a rover that could be controlled and move in a predictable way, as well as to be able to transmit data from various onboard systems to a webapp and communicate to various physical hardware the routines and functionality implemented.

Group/Project Management

During the initial meeting, each subsystem was allocated to different members depending on the individual's preference and we established that every Monday we would meet and discuss our next steps whilst also aiding one another. We prioritised communication throughout as the final rover requires the tight integration of all our different subsystems and our ability to discuss and cooperate work would reflect how well that would tie together. We regularly updated a Gantt chart that effectively communicated to all members what new changes were being made to the various systems, and this along with enforced internal deadlines allowed steady progression throughout the rover's development. Days highlighted orange are the days that the group decided to meet.

Additionally, we set up a GitHub with branches allocated for each subsystem; every time a new goal was accomplished, we would commit our changes, allowing all systems to have access to complete working versions of other subsystems to use during development. We maintained contact through a WhatsApp group.

We also chose to code in a waterfall methodology in a group, as we felt this would be a wise strategy due to our development be in a very linear manner from research to design to implementation and then integration. This allowed for further sections to be built upon each iteration of code and we could perform a variety of tests systems before these systems were integrated.



High-level overview of full system

Individual Modules

Command

The aim of the command module is to build a fully functional cloud-based web app that can interface with the control module and allow the rover to carry out various functionality such as display its status and to control the rover remotely during autonomous exploration phase.

Tech Stack Implementation

We implemented the cloud-based web app using the MERN stack which consisted of MongoDB, Express JS, React JS and Node JS. Express JS was utilized as a Restful API to query to the serverless MongoDB and respond with JSON objects to the React Frontend. The backend, written in Express JS, was hosted on an AWS EC2 instance and run indefinitely using the PM2 daemon process manager. The JSON data returned by the API queries, was data read from MongoDB Atlas, and it was chosen for its JSON like document data model as this allowed us to have a more streamlined transfer of data without having to change the structure of the data in several locations. There were 5 collections set up which then individually stored the JSON data generated by the rover.

Back-end

Express JS was chosen as it is an extremely popular lightweight web framework which would allow us to write different handlers for requests for different HTTP verbs at different URL paths. The ports of the web app are connected to Nginx which is a high-performance reverse proxy server which acts as a pseudo permission to open a network connection on the specific AWS ports where the app is running. For the backend server our primary goal was to design a RESTful API which conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. Implementing this architecture would allow us to make the API faster with increased scalability.

Justification for a Database

A web API doesn't have a data store; hence we used the backend and client cache to temporarily store any data locally on the client-side. However, this method could not be relied on to create a robust system. There were situations where the data needed to be centralised and distributed to all clients simultaneously. One example is logging discoveries made during the rover's mission. Without a central data store, when a new client loads the web-app mid-way during the mission, all the previous data will be lost. Using a database, we can store each discovered item in a new entry so that all discoveries are continuously stored and updated.

Front-end

The primary goal for the frontend web app was to predominantly use React JSX and avoiding using any custom JavaScript in the root DOM. Despite the difficult implementation, its benefits included instant page rendering, lower website latency and code modularity.

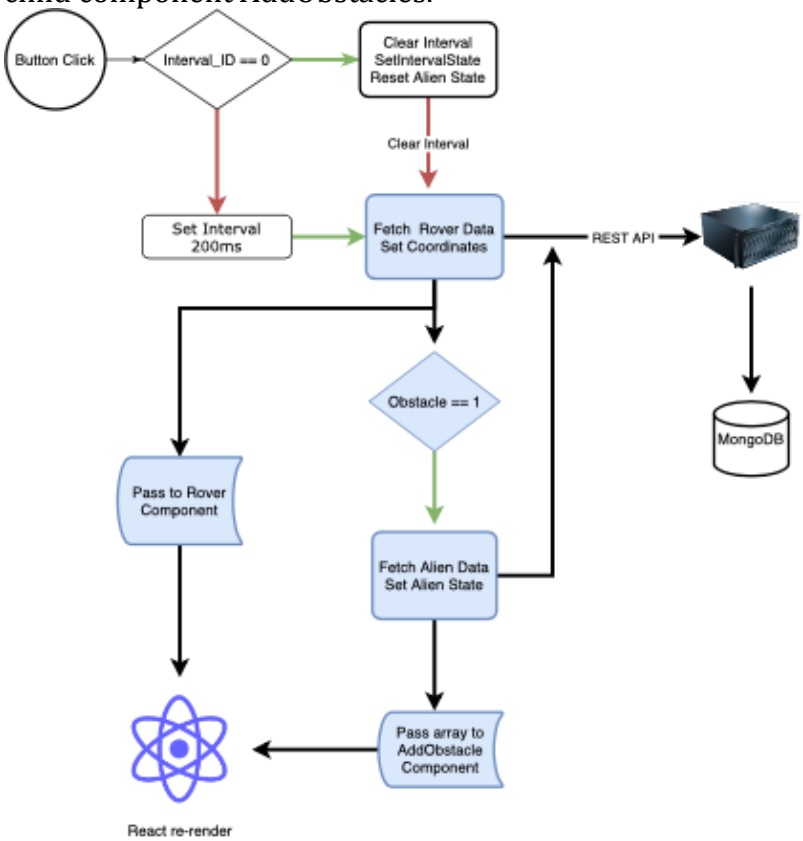
An initial difficulty encountered was building the battery web page, as the data was dynamic. This was an issue as when the user would navigate to that subpage, there would be an "undefined" value displayed, as the battery level could only be shown when the user requested for it by clicking a button. A workaround was found with React Hooks, which extract stateful logic from a component so it can be tested independently and reused. The two main hooks that were used to solve this issue were the State hook which is called inside a function component to add some local state which is preserved between re-renders. The Effect hook adds the ability to perform side effects from a function component like data fetching.

To solve this issue, 2 State and 2 Effect hooks were required in conjunction with each other. When navigating to the battery webpage, the real-time battery level had to be displayed, an Effect hook had to be used to automatically call a JavaScript asynchronous function which would store the data into a State Hook. Another Effect hook was dependent on the battery state and would plug into another state hook to achieve the

current desired battery level icon. This feature provided a significant improvement in UI and much significant performance benefits as we are not passing the battery icon through the JSON object but instead storing a lightweight (30kb) PNG and rendering based on current battery health.

One of the challenges was displaying the rover's position along with the obstacles of different colours with pure React JSX. An inelegant solution would have been injecting JavaScript into the root React DOM, but this would've had significant latency trade-offs. Instead, React Props was utilised to pass data to child components, thereby allowing us to render the aliens as and when they were detected by the rover.

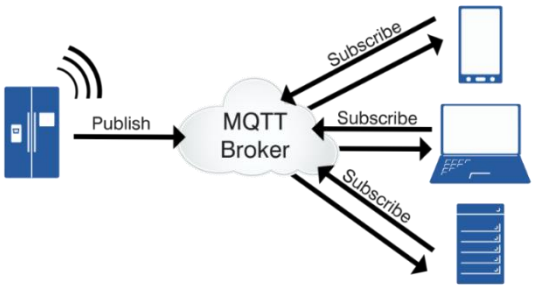
To display the rover's position with precision, we had to request the location every 200ms. This process could be achieve using State Hooks and SetInterval(). To reduce the number of API request, we decided to set up flags inside the JSON coordinates object called isObstacle and isFan. This indicated the detection of an alien and sent an API GET request to obtain the alien's coordinates. The flags inside the object result in improved latency by a factor of 3, increasing overall efficiency. Furthermore, the obstacle object is then passed into a state hook on the frontend and passed as a prop to be rendered by its child component AddObstacles.



When the JSON object is passed to the Obstacle component, it is rendered within the HTML div tag which acts as the map area. The positioning of the obstacles is done relative to the absolute parent div using CSS parameters of top and left. All the boxes in blue in the figure on the left are looped and rerun every 200ms to continuously update the rover's position and obstacles on the map

MQTT

Unlike TCP which is a network level protocol, MQTT is an application layer protocol that makes it easier to handle multiple connections through its publish and subscribe model. The data can be categorized into parts called Topics. An example of a topic we can create is 'battery', in which the current battery data is available for clients to access. Publishing is the act of sending local data to the MQTT server. Subscribing is the act of requesting published data. Data can be published to a certain topic and clients can subscribe to a specific topic to get the information that they desire.



The main interface that handles client interaction is the MQTT broker. We decided to use the Eclipse Mosquitto broker for our project as it's one of the most widely used brokers, hence there's a lot of support for it especially online. The broker handles publish and subscribe requests from connected clients and sends the corresponding data when a client subscribes to a topic. This type of model makes it easier to separate different types of data and manage access. MQTT connections can also be password protected and we can restrict certain topics depending on authentication. This adds a security layer that standard TCP sockets did not have and would be beneficial if the system is expanded for a larger userbase.

Control/Integration

The control subsystem is the main convergence point for all the other, in other words, it performs the integration of all the subsystems, and each has data that can be taken into the ESP32 processor for manipulation or transferring into other systems.

Individual Physical Subsystems

Drive Subsystem:

Using an I2C connection to an H-Bridge, the ESP32 receives commands from the user app and then sends signals to the motors to perform different instructions, such as turning or moving.

Optic Sensor System

The optic sensor tracks the displacement of the rover at every sample interval. This value is modified by 1/40 to approximate each x and y point to one centimetre. The Chip Select signal is then set to high at the end of the use of the optic sensor in order to terminate the SPI communication, allowing the FPGA/Vision system to be communicated with.

Gyroscope

The gyroscope system is implemented with an SEN0142 MPU6050 purchased as an extra component for £x. It tracks the angle of the rover relative to the start angle. The ESP32 samples the gyroscope every loop then multiplies the rotation speed by the cumulative delay to get the change in angle.

Radar System

When the radar detects a fan underground, the current angle of the rover, the radar, the radar's tilt and height above the ground are used to ascertain the general coordinates of the fan.

Ultrasonic Sensor

The ultrasonic sensor solves a specific problem to do with the detection of buildings and walls. The sensor checks for obstacles in its way. If anything is detected, the ESP32 internally evaluates all data from all sources to determine if the detection is an alien, building, or wall, and sends to command.

Vision System

The FPGA sends in data in the form of an SPI bus input to the ESP32 through its SPI pins. This signal is then processed and converted to give the colour of the alien detected, the approximate pixel width and approximate angle (0 to

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bits 0-2: Object specification (alien colours			Bits 3-10: the diameter in pixels of the scanned object								Bits 11-15: the angle of the object relative to the left side of the camera view (0 < angle < 75 degrees)				

Contents of packet from the FPGA over the SPI input bus.

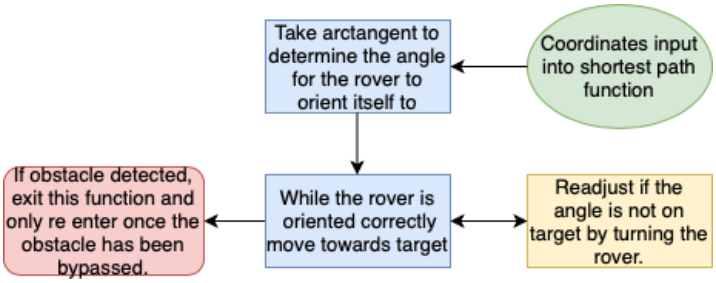
70 degrees). This is then processed further to give approximate distance, and then to calculate using the rover's current position and angle relative to start the approximate location of the alien detected. A logic HIGH is then sent to the Chip select to disable the FPGA's input stream, as the SPI Chip Selects are active LOW.

Integration for Higher Level Functionality

Integration

The displacement measured by the optic sensor and the angle calculated from the gyroscope's angular rotation allows for the rover's X-Y coordinate to be approximated. This is done using simple trigonometry.

Shortest Path



The objective is to be able to pass coordinates to the rover, and it will in turn adjust its orientation and then travel to those coordinates, avoiding obstacles. This is implemented by passing target coordinates into the shortest path function. By taking the arctangent of (x-rover.x/y-rover.y), (where x and y are target coordinates and rover.x and rover.y are the current coordinates of the rover) the angle the rover needs to orient itself to is determined. If an obstacle is encountered at any stage this function is suspended and the avoidance function takes over until the object has been bypassed. The rover will then resume movement to the target coordinates. The rover is programmed to be able to intake a set of coordinates and then go to each one in order of proximity, by evaluating the distance to each coordinate from the current rover coordinates each time one is reached.

Obstacle Avoidance

When an obstacle is detected, either via the Ultrasonic sensor or the Vision System, the rover enters avoidance mode. In this state, the rover suspends normal activity and attempts to manoeuvre around the obstacle(s). It does this by turning 90 degrees (Left or Right), proceeding forward, then turning again (Right or Left), proceeding forward, Right or Left again, and finally arrives at its destination and orients itself again, terminating the sequence. If it encounters another obstacle while doing this, it will start a new avoidance process. A high-level view of the algorithm is described below:

1. An obstacle is detected
2. Turns L/R to orient the rover
3. Turns R/L and checks for the obstacle - moves forward if there are no obstacles seen.
4. If there is an obstacle, repeat from step 1.

Vision

The objective of the Vision Subsystem is to use the terasIC D8M Digital Camera module to observe the environment and discover or identify objects that could be in the path of the rover. These obstacles take the form of multi-coloured balls (aliens) or a striped piece of paper (a building).
After identifying these objects, this Subsystem will determine the rover's approximate distance from the obstacles as well as its relative orientation. This will then be

communicated to the Command and Control subsystems so that they can be safely avoided and displayed on the Rover's web interface. The algorithm used to accomplish this object detection is described as a series of steps below:

HSV Conversion

To make image processing more practical, the implementation of an RGB to HSV converter was necessary – colour itself, or Hue, is represented as a single parameter ranging from 0-360, and intensity and brightness are represented as a combination of Saturation and Value (ranging from 0-255 in this specific implementation to avoid floating point calculations and the performance/space trade-offs that come with it). This representation allows easy manipulation of the image as the object detection algorithm works by simply detecting if the values of the current pixel fall into a defined threshold for each of the known obstacles. [18]

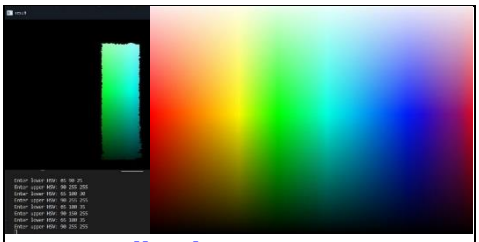
Auto Gain Algorithm

To detect obstacles in the Rover's environment, the parameters of the camera must be set so that it does not cause extreme bright spots in the image whilst maintaining a clear and well-lit image. The parameter to be considered is the camera's gain, which ranges from 0x0-0x800 and directly affects the brightness of the image seen by the rover. During the HSV conversion of the image, the running sum of the values across all pixels is maintained and communicated to the NIOSII processor at the end of the frame – as Value represents the 'brightness' of a pixel. If the average Value is deemed too low or high, the algorithm simply adds or subtracts a small unit to the current camera Gain, which then increases or decreases the Value, eventually stabilising it - this is essentially a negative feedback controller. Using this algorithm, the object detection algorithm can be assured the current image is bright enough to identify any objects and does not require specific tuning for different brightness levels.

Colour Separation and Noise Removal

The method to capture the obstacles simply involved defining HSV thresholds for each obstacle and detecting whether the current pixel falls into the defined threshold – if it does, then it is considered to be part of that obstacle and will be accounted for during distance and relative angle calculation (the ultimate goal of this subsystem).
The values used to define the thresholds for the object detection were obtained through trial and error as well as the usage of our own Python script (named HSVBoundHelper.py) that easily allowed us to view the effect of different lower and upper thresholds on any given image instantaneously, rather than adjusting the values in Verilog and waiting for it to compile to see the effect:

An example of the usage of the HSVBoundHelper.py script that removes pixels that does not fall within the upper and lower HSV values inputted by the user. The original image is shown on the right.



Full Color Spectrum - ColorTools.net |
A colour spectrum image was also used to see exactly what colours the bounds defined.

Technique 1

Originally, a posterization/quantise algorithm was used to reduce the total number of colours available to the object detection algorithm to better 'pick out' different obstacles based on their HSV bounds, with the goal of not requiring any filtering. This was done through the following technique:

1. Separate H, S and V into a discrete number of steps
2. Determine the bands in between which the values for that pixel lie.
3. If the actual value of the pixel is above the midpoint of that band, set it to the upper end, else, set it to the lower end.

An example of how noise in the image was made worse by the posterization algorithm.

Once implemented, the image noise became much more apparent as random pixels would fit into the thresholds defined for the balls, causing the image to look even worse than before (as noisy pixels were now highlighted instead of being filtered out). This made the algorithm redundant and was an oversight in retrospect as the noise is inherently random and would not conform to the pixels neighbouring it.

Additionally, a median filter was designed and implemented to remove noise, however it required constructing a 3x3 window around the current pixel, which consumes an extreme number of resources as 3 rows of 25-bit pixel data (3x640) must be stored in memory. Even after heavy optimisation, this resulted in the Total Logic Elements to be 98% as reported from Quartus Flow Summary, and as further steps were still not complete, this implementation was likely not even possible.

Technique 2

Taking inspiration from this attempt, a new algorithm was devised which by nature performs filtering, and therefore noise removal, whilst also requiring much greater tolerances in the HSV thresholds defined for each obstacle.

The revised technique is described as follows:

1. Allocate a shift register of size n for every object to be detected.
2. If the current pixel activates a pixel threshold defined for any of the obstacles, store a 1 at index 0 into the shift register for that obstacle.
3. For the current pixel, if the number of 1s in the shift register for any obstacle exceeds threshold k (this limit can be tweaked), consider the current pixel to be part of the obstacle, and consider it for distance and angle calculation (described later). Additionally, set the pixel's colour to be that of the obstacle it is detecting – i.e., if there are at least k pixels that are detected as red in the n pixels before the current pixel, the current pixel will be outputted as red. This is the filtering step, and essentially is a mode filter.
4. Shift all shift registers by 1 bit.

The mode filter defined in step 3 is the step that cements this algorithm as being vastly superior to the previous one; no sorting algorithm is required and is easily scalable as only the length of the shift registers is the only parameter that would change – allowing for easy sensitivity adjustments.

Additionally, the mode filtering removes a substantial amount of noise from the image as the other n-k pixels in the window would vastly outnumber the number of noisy pixels – adjustments to the k threshold or the n window would improve noise mitigation but may also introduce false positives, however as described in the next step, noisy

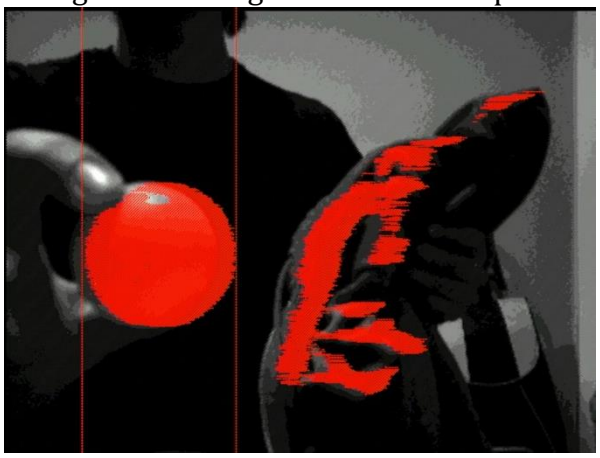
pixels that happen to fall into the HSV bounds for the obstacles will not affect the algorithms ability to detect them, so parameters that optimise entire detection of the obstacles are preferred over ones that mitigate noise.

Bound Drawing

From the mode filtering above, the next step is to draw vertical bound lines around the obstacles to measure the distance of the obstacles from the camera. This bound drawing must only contain the obstacle, which means it must be insensitive to noise pixels that happen to fall within the colour detection ranges for the different obstacles.
This algorithm works as follows:

1. If the mode filter has determined the current pixel to be part of an obstacle, and it is the first pixel to be detected, set it as the temporary left side bound.
2. From now on, if more pixels are detected, keep count of how many pixels have been detected thus far, i.e., the width, and set the temporary right side bound to the most recently detected pixel.
3. If a pixel is no longer detected, assume the entire obstacle has been scanned over. If the width of this specific colour obstacle is larger than anything detected thus far, replace the true bounds with the temporary ones. This step rejects noise, as noisy pixels would only be a few pixels in width and thus cannot override an obstacle if it is within range of the camera.

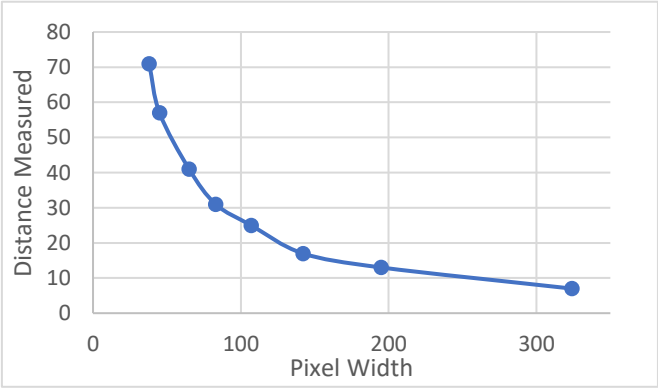
To further combat noise, a minimum distance between the left and right bounds is enforced at 60px.
For the building, a slight adjustment had to be made where the bound is only drawn if enough alternating black and white pixels were found.



The image on the right showcases the algorithms' ability to reject noise – the red jumper (the noise) matches the HSV bounds for the red ball, however the red ball in the image overrides the jumper as it has more red pixels in succession. It can also be seen that the red mask around the ball is offset slightly in the x direction as mentioned previously.

Distance and Angle Calculation

Pixel Width (px)	Distance Measured (cm)
324	7
195	13
142	17
107	25
83	31
65	41
45	57
38	71



Originally, an analytic approach using formulae were considered to determine the distance as well as the relative angle of the obstacle, however lack of knowledge about the parameters defining these equations as well as the fact that they are not static, meant that it would be difficult to implement.
Instead, an approximation was used with exponential regression – we simply measured the actual distance away from the camera the obstacles were and the apparent size of the obstacle in pixels and plotted the data. This allowed us to find a best fit curve to use for the rover's distance estimation.

After using an online exponential regression calculator (see references), the curve had a best fit equation of $y=ab^x$ where a = 68.23 and b = 0.9923. This function was implemented in the ESP32 as it requires the use of exponents and floating-point numbers, the Vision system only communicates the apparent obstacle width in pixels. As the current pixel must have k out of n pixels detected before it, the bounds drawn for the obstacles will be off by at least that same threshold k in the x direction (but the width of the detected obstacle should remain the same) – while not an issue for distance calculation, introduces an error for the angle calculation, however as the angle data lacks precision, the relatively small error (compared to the 480 horizontal resolution) is insignificant and can be ignored.
Angle calculation is fairly trivial – it is defined as the midpoint of the obstacle/20 (which will map the 0-480 horizontal resolution to 0-31, which fits the reserved 5-bit data width at the end of a packet.)

SPI Communication with ESP32

After unsuccessfully trying to use the Intel 3-Wire SPI IP Core, a 3rd party guide that uses the alternative Avalon-ST SPI Intel IP Core was followed, which implemented the necessary hardware required to communicate with the ESP32; however, the core only supported data widths of 8 bits, which meant that the expected 16-bit data packet for each obstacle had to be broken down into 2x8 bit packets. The layout for the packet was described in the Control section.

Radar

The radar subsystem is intended to locate the underground alien infrastructure characterised by subsurface rotating fan blades. The microwave motion sensor module, the HB100, detects the relative speed of moving objects using the doppler effect: The radar emits a 10.525 GHz (microwave) signal, allowing it to pass right through wood or plastic. If an object moves relative to the module, the reflected signal's frequency will be shifted. The output of the module – intermediate frequency (IF) – represents the magnitude of said shift. This is a low-level voltage: it produces a signal of ~10mV amplitude when the radar is next to the target, as well as clutter noise from surroundings. It has a DC offset of -100mV. As the rover moves, stationary and non-stationary objects will have different relative velocities to the rover. Therefore, to accurately relay the location of the alien infrastructure through the ESP32, a circuit was designed to fulfil three main duties:

1. Attenuate the clutter noise while amplifying the signature frequency band of the fan (determined from preliminary tests to be 400 ± 50 Hz).
2. Use a peak detector to smoothen the filtered signal
3. Use a comparator to output a digital high (3.3V) signal when voltage exceeds a threshold.

The resultant circuit is shown in Figure 3.

Section 1 is a three stage, 6th order Butterworth bandpass filter with a 40dB amplified 100Hz passband, centred at 400Hz. This was designed using Analog's filter design tool with the LT1078 op-amp due to convenience and price. As only +5V and 0V supplies are available, Section 2 sets up a 2.5V reference voltage for the filter using the MCP6002 op-amp. The magnitude bode plot of the bandpass filter is shown in Figure 1.
We take the output of the filter and pass it through Section 3 – a peak detector – essentially a diode with a resistor and a capacitor that allows us to smoothen the signal by capturing the maximum point of the waveform. Since the load is varying, a buffer circuit is used to prevents the capacitor from discharging through the load resistor. Capacitor C14 was determined following equation, where f = 400 Hz, and is the forward resistance of the 1N914 diode, which was chosen for its quick response time. can be as low as 100 Hz.

$$C_{14} \geq \frac{1}{10 f R_d} = \frac{1}{10 \times 400 \times 100} = 2.5 \times 10^{-6}, \qquad \text{Thus } C_{14} = 100\mu\text{F}$$

Through trial-and-error with the LT-spice simulation, values for R14 and R15 were adjusted to maximise reliability of circuit. The transient analysis of the output over 500ms:

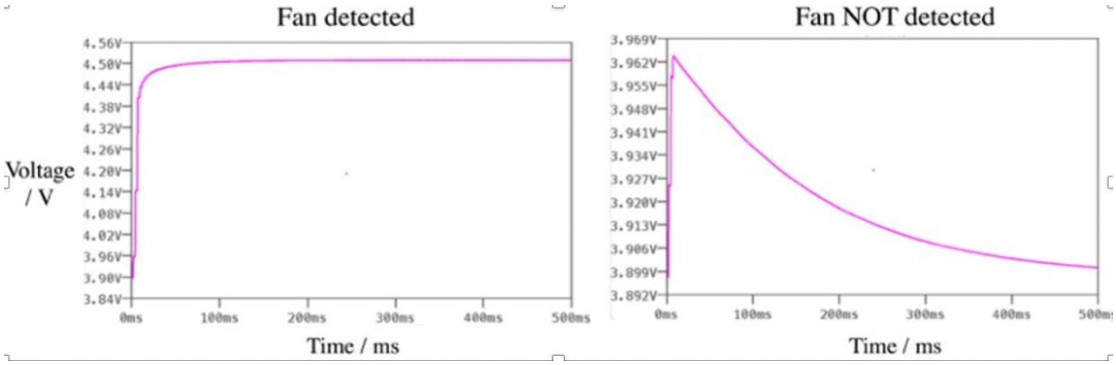


Figure 2 – Transient analysis of the output of sections 1 to 3 for both situations when there is a fan+noise (left) or when there is just noise (right). Done on LT Spice.

As seen in Figure 2, when a fan is detected, the voltage plateaus at 4.5V, whereas if only clutter noise is present, it never exceeds 4V. Therefore, the comparator's threshold voltage was set to 4.16V using a simple potential divider circuit (section 4). The output of comparator swings from 0V when LOW (peak-detector voltage < 4.16) to 5V when HIGH (peak-detector voltage > 4.16). Consequently, another potential divider is added after the output to reduce the output voltage swing to 3.3V for the ESP32 to detect the signal using the digitalRead() function (returns HIGH or LOW depending on the signal into the pin). Therefore, when a signal with frequencies between 350-450 Hz (fan) is detected, Vout sends a constant 3.3V high digital signal, which is then read by the ESP32.

On the ESP32's side, there is a relatively simple code that determines the approximate location of the fan given the angle of the radar subsystem relative to the ground and the current angle of the radar relative to the X-Y coordinates of the Rover, which is calculated by adjusting the angle of the radar by -90, then adding that value to the Rover angle. This code only runs if the radar input goes high. As the radar swivels on a servo by 2 degrees every cycle, it allows it to complete a full 180-degree rotation every x seconds. On the

right is the ESP32-side code that controls the radar subsystem. When called, it will publish data to the server after simple calculations to determine the fan's location.[i]

Drive

The drive subsystem consists of two main sections: the optic sensor and the motor control system. These subsystems are mostly separate to each other. The optic sensor data is used in specific cases to control the motors.

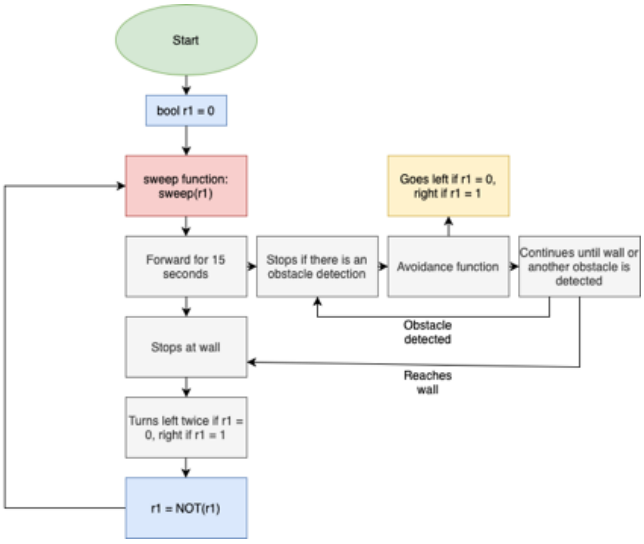
Initially, there were a few issues. The optic sensor first failed to properly adjust and provide useful data. This was resolved after fixing the sample code. Then there was the issue of how to use the code. It was initially configured to measure displacement in the X-Y plane like a mouse. For the rover, which rotates, this would not be ideal. The solution was to take the displacement instead only in the direction the rover was facing at any given time. Combined with the gyroscope angle data, this allowed for tracking the X-Y coordinate of the rover from an initial position 0,0, facing the positive Y direction. This is achieved practically using basic trigonometry. The gyroscope is found out to be inconsistent during testing due to sampling speed and motor speed. This causes the rover to rotate inconsistent angles which are undesired. An alternative is to turn the rover by rotations of the motors. By estimating the time of rotating one full circle, the time for the motors to turn can be determined by ratio between the desired angle and one full circle.

RoverX Coordinate += displacement*sin(angle) RoverY Coordinate += displacement*cos(angle)	Note: displacement is in the axis of motion of the rover. The angle is calculated using the gyroscope.

The limitation of this method is the time for turning would vary if the surface of the field is uneven, which in that case gyroscope is more favourable to use.

Automated drive function

The automated function consists of two modes. One mode for each direction of sweep. On the first sweep, the rover will go from left to right, if it detects any obstacles it will avoid by going left. At the wall, it will turn left twice and then enter the second mode, where it does the same thing in reverse. Each mode runs on a timer where the rover will go forward until detecting an obstacle or a wall, at which point it will avoid or turn depending on the mode. The mode is tracked by the variable r1, a Boolean which switches at the end of the modes to start the next one. A limitation of this algorithm is that the rover might not come across to every obstacle when the rover has avoided two or more obstacles consecutively.



Energy

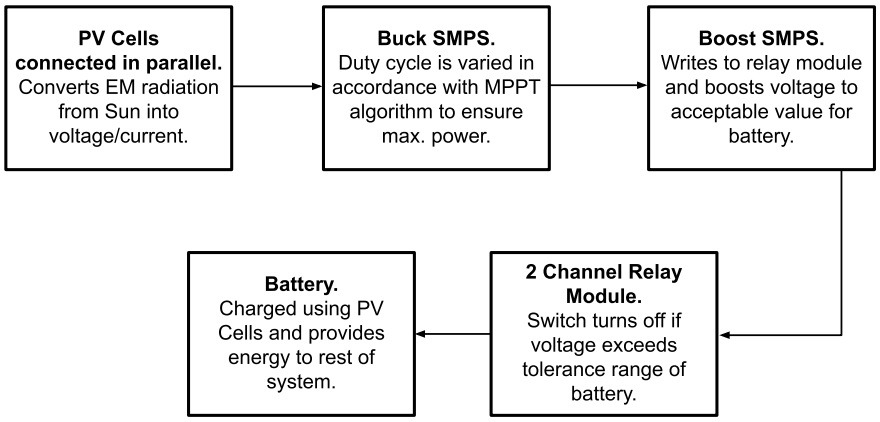


Figure 4: Overview of energy sub-module.

The energy sub-module ensures that the battery is charged most effectively. To achieve this, four PV cells, 2 buck-boost SMPSs, a 2-channel relay module and a battery were used. The entirety of this sub-module was designed upon the assumption that it is stationary. This design choice was made due to the rover being unable to accommodate all this equipment on its journeys and due to the additional energy such weight would require. Hence, the rover's journeys are constrained to the proximity of its charging station. Furthermore, the rover is equipped with code that ensures it has enough charge to return to its charging station.

PV Cells:

To charge the battery, a rather consistent current with an acceptable amount of power is desired. Thus, a consistent current is favoured over a high voltage, such that the four cells were chosen to be connected in parallel rather than series. Due to this, the maximum voltage achieved by the PV cells is around 5V.

MPPT and PV Cell Characterisation:

Mars' conditions are unpredictable and volatile. For instance, one cannot predict how much energy the PV Cells will absorb at an instant due to natural variations in irradiance, cloud cover, dust-covering etc. Hence, it is crucial that no-matter the conditions, the maximum amount of power is being derived from the PV cells. To achieve this, a Maximum Power Point Tracker (MPPT) algorithm is utilised. As ambient conditions vary, the voltage at which the PV cells will output the peak amount of power will vary. Hence, the PV cells are connected to a buck SMPS, which can vary the voltage anywhere from approximately 0 to 5V. The algorithm chosen to implement the MPPT was Perturb and Observe, i.e., P&O, due to its relative simplicity and trial-and-error based nature.

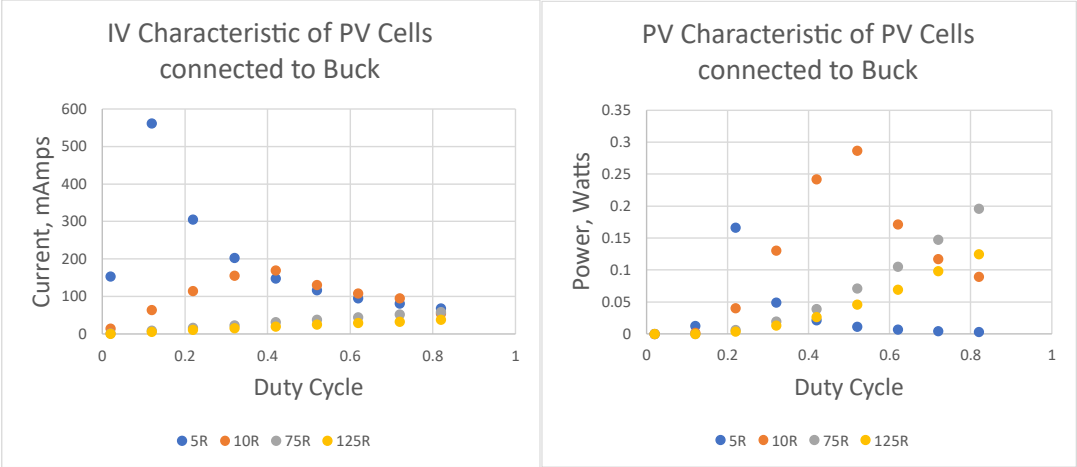


Figure 5: IV and PV Characteristic of PV Cells connected to Buck SMPS for different resistances.

The current and power characteristic of the PV cell-buck sub-system may be seen in *Figure 5*. Note, the independent variable was decided to be the duty cycle rather than voltage, since having a consistent set of incremented voltages for each resistance would have required extensive, trivial guess work. The IV graph obtained deviates heavily from what is expected, as one would expect the current to be constant and then rapidly decrease to zero as the duty cycle approaches 1. This behaviour may be due to the fact that the PV cells are connected to the buck, which would alter the currents behaviour. The power has a peak for the lower resistances of 5 and 10R, yet not for the higher resistances. This may be due to the maximum power not being within the range of 5V.

Boost SMPS:

The battery requires an input voltage in the range of 4.5 to 5.2V. However, according to *Figure 5*, this may not always be feasible as the MPP may be at a lower voltage. Hence, a boost is required increase the buck's voltage such that it is within the acceptable range of the battery. Another issue is incurred by the fact that the battery will draw different amounts of power for different input voltages. For instance, the output of the buck may supply 0.2W of energy, yet the boost is boosting the voltage to 5V, at which the battery attempts to pull 2W of energy. Essentially, if the input voltage of the battery is not mapped correctly to the output power of the buck, more power will be drawn than can be provided, such that the system fails, and no power is drawn at all. Thus, a voltage sweep of the battery's input must be done to determine how much power each input voltage between 4.5 and 5V draws. In practice, this would consist of plotting the input power of the battery against voltage in the range of 4.5 – 5V, then finding the equation of the line of best fit. This equation can then be used to calculate the required input voltage for a certain power. In the case of the 0.2W example, this may mean the input voltage must be 4.6V such that the battery draws 0.2W. Knowing this, the boost can be adjusted to boost the output voltage of the buck to 4.6V, such that the amount of power the PV cells provides corresponds with the amount the battery draws. Unfortunately, we were unable to achieve this in practice due to the boost SMPS not working, even after receiving 2 replacements.

The battery may be damaged if the input voltage is not within the required range. A 2-channel relay module is used to circumvent this. The boost SMPS measures its output voltage and sends this data to the Arduino. If the output voltage is not within the required range, it tells the relay module to switch off, such that the wire connecting the boost and battery becomes open circuit.

Homecoming Code:

The homecoming code monitors how much charge is left in the battery and informs the rover when to return to the charging station to avoid battery-death during a mission. An estimate for the current usage of the rover was obtained experimentally and extrapolated. This estimate was used to determine how long the rover could run on a full charge – this length of time was defined as 100% charge. Thus, to determine the remaining charge, we subtract the run-time of the rover from the full-length the rover can run for. For instance, assume it was found that the rover could run on a full charge for 2 hours. If the rover has been runing for 1 hour, the battery is approximately at 50%. The battery charge is always underestimated as a precaution to ensure the rover always makes it back to the charging station. Although this is effective, it is not efficient.

When the rover returns to the charging station, which is assumed to be at x, y coordinates = 0, it takes a path which cannot be predicted before hand. However, this path can be bounded. Assume a right angle triangle is made between the x, y = 0 and the x,y coordinates the rover is currently at, as seen in *Figure 3*. The shortest path of return is along A and the longest along the sum of B and C. On Mars, it is unlikely for the rover to have an unobstructed path, such that it would not return along A. Instead, it would most-likely travel along a path resembling the red one, where it has to circumvent obstacles. This path is equal to the length of B and C summed. Knowing this distance and the average speed of the rover, the code estimates how long it would take to rover to return. If the run time the battery can supply is less than double the time it takes the rover to return, the rover is informed to return to the charging station immediately

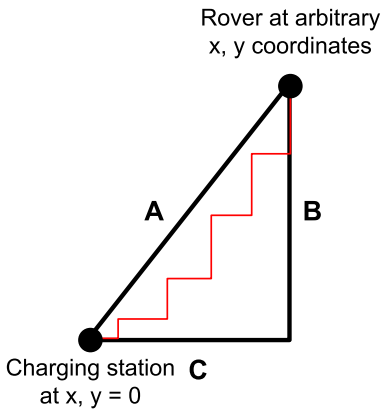


Figure 3: Potential return paths of the rover.

Testing

Our testing suite consisted of a 3-stage process to ensure our components were well tested and bug free.

Before basic integration, each individual subsystem was tested using techniques such as local debugging tools, randomized testing and iterative testing.

Once these modules were thoroughly tested, we decided to integrate the rover with basic functionality such as remote operation this helped us to debug many errors.

Conclusion/Reflections

In the beginning of the project, it would have been ideal to have put more effort and more time per day into learning the new systems for the project (ESP32, platformio, etc) to gain the proficiency needed to do proper work. It would have also been convenient to have understood more rigorously what the specific requirements for the systems were earlier, as it took time to fully comprehend the scope of the project. For the future, I would've used cache balancing on the Nginx webs server and implemented HTTPS support in addition to buying a web domain.

References

[1]"React – A JavaScript library for building user interfaces", *Reactjs.org*, 2022. [Online]. Available: https://reactjs.org. [Accessed: 24- May- 2022].

[2]"MongoDB Community Download", *MongoDB*, 2022. [Online]. Available: https://www.mongodb.com/try/download/community. [Accessed: 13- Jun- 2022].

[3]"Express - Node.js web application framework", *Expressjs.com*, 2022. [Online]. Available: https://expressjs.com. [Accessed: 02- Jun- 2022].

[4]R. Wieruch, "How to use Props in React", *Robinwieruch.de*, 2022. [Online]. Available: https://www.robinwieruch.de/react-pass-props-to-component/. [Accessed: 03- Jun- 2022].

[5]M. Otto, "Bootstrap", *Getbootstrap.com*, 2022. [Online]. Available: https://getbootstrap.com. [Accessed: 02- Jun- 2022].

[6]"How To Deploy a React Application with Nginx on Ubuntu 20.04 | DigitalOcean", *Digitalocean.com*, 2022. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-deploy-a-react-application-with-nginx-on-ubuntu-20-04. [Accessed: 16- Jun- 2022].

[7]I. Sysoev, "nginx news", *Nginx.org*, 2022. [Online]. Available: https://nginx.org. [Accessed: 23- Jun- 2022].

[8]"Icon", *Battery Icons*, 2022. [Online]. Available: https://icons8.com/icon/set/battery/color-glass. [Accessed: 01- Jun- 2022].

[9]M. Aoussiad, "How to Create a Popup Modal in React", *Medium*, 2022. [Online]. Available: https://javascript.plainenglish.io/how-to-create-a-popup-modal-in-react-39315907998e. [Accessed: 21- Jun- 2022].

[10]J. Watmore, "React + Node.js on AWS - How to Deploy a MERN Stack App to Amazon EC2 | Jason Watmore's Blog", *Jasonwatmore.com*, 2022. [Online]. Available: https://jasonwatmore.com/post/2019/11/18/react-nodejs-on-aws-how-to-deploy-a-mern-stack-app-to-amazon-ec2. [Accessed: 23- Jun- 2022].

[11]"Mongoose ODM v6.4.0", *Mongoosejs.com*, 2022. [Online]. Available: https://mongoosejs.com. [Accessed: 15- Jun- 2022].

[12]N. Dsouza, "Image not rendering from Props in React", *Stack Overflow*, 2022. [Online]. Available: https://stackoverflow.com/questions/67606845/image-not-rendering-from-props-in-react. [Accessed: 07- Jun- 2022].

[13]"Tailwind UI - Official Tailwind CSS Components & Templates", *Tailwindui.com*, 2022. [Online]. Available: https://tailwindui.com. [Accessed: 23- Jun- 2022].

[14]"Product example for Bootstrap", *Getbootstrap.com*, 2022. [Online]. Available: https://getbootstrap.com/docs/4.0/examples/product/#. [Accessed: 30- May- 2022].

[15]"ESP32 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials", *Random Nerd Tutorials*, 2022. [Online]. Available:

https://randomnerdtutorials.com/esp32-pinout-reference-gpios/. [Accessed: 07- Jun- 2022].

[16] *Intel.com*, 2022. [Online]. Available: https://www.intel.com/content/dam/support/us/en/programmable/support-resources/bulk-container/pdfs/literature/hb/qts/n2cpu-nii51011.pdf. [Accessed: 07- Jun- 2022].

[17] E. Member 2, "UART and SPI on FPGAs - FINCH - Confluence", *Spacesys.utat.ca*, 2022. [Online]. Available: http://spacesys.utat.ca/confluence/display/FIN/UART+and+SPI+on+FPGAs. [Accessed: 23- Jun- 2022].

[18] "Exponential Regression Calculator", *Omnicalculator.com*, 2022. [Online]. Available: https://www.omnicalculator.com/statistics/exponential-regression. [Accessed: 07- Jun- 2022].

[19] "RGB to HSV conversion", *https://www.rapidtables.com/convert/color/rgb-to-hsv.html*, 2022. [Online]. Available: https://www.rapidtables.com/convert/color/rgb-to-hsv.html. [Accessed: 02- Jun- 2022].

[20] A. Fernandez, "HSV to RGB (and back) without floating point math in Python", *Stack Overflow*, 2022. [Online]. Available: https://stackoverflow.com/questions/24152553/hsv-to-rgb-and-back-without-floating-point-math-in-python. [Accessed: 06- Jun- 2022].

[21] V. Chernov, J. Alander and V. Bochko, "Integer-based accurate conversion between RGB and HSV color spaces", *Science Direct*, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0045790615002827. [Accessed: 07- Jun- 2022].

[22] ”ESP32 – Ultrasonic Sensor” https://esp32io.com/tutorials/esp32-ultrasonic-sensor

Appendix

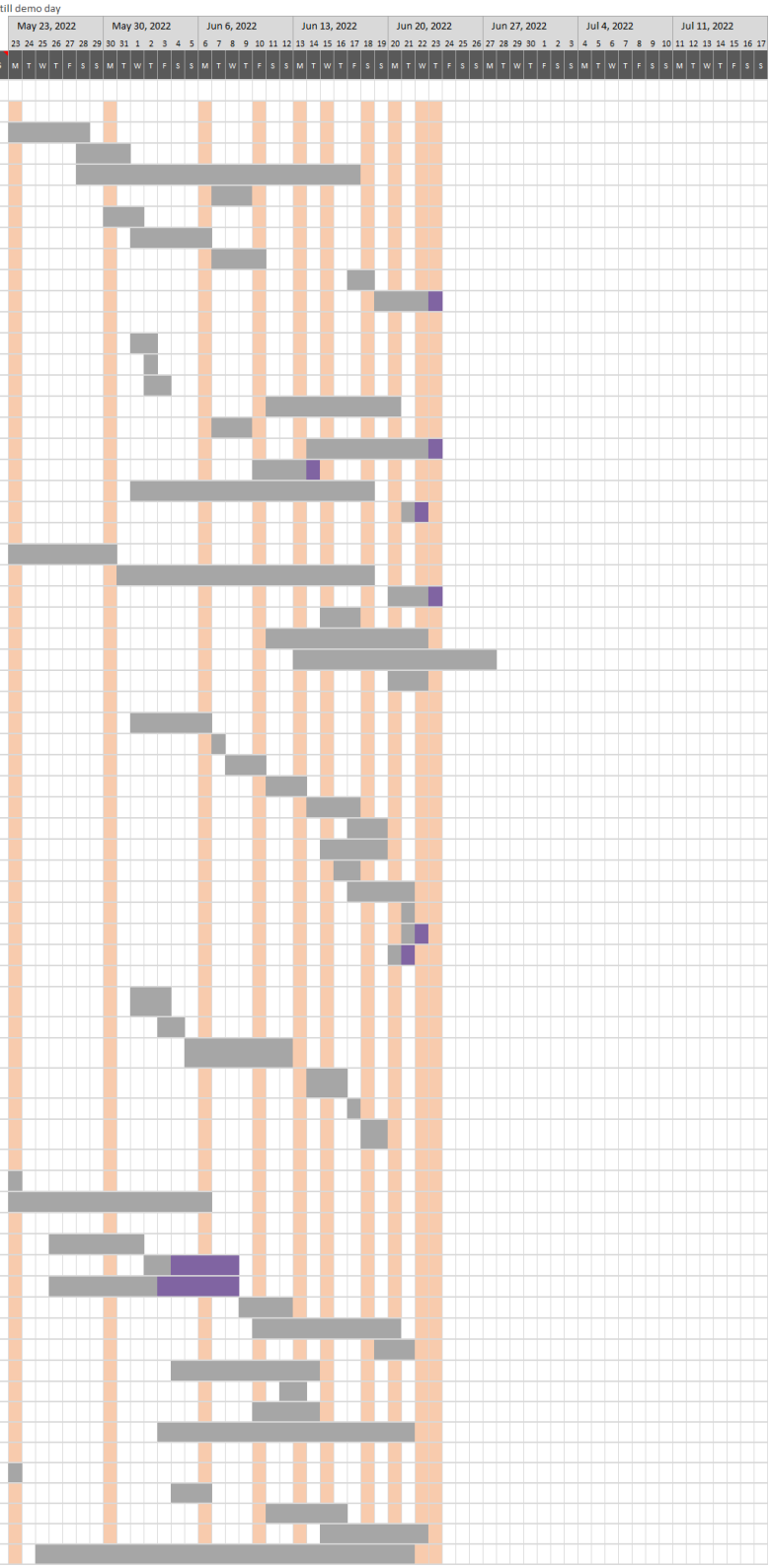
Mars Rover 2022

Wall-EE (16 - JWA2)

Shaankuka - Shaheen - Indraneel - Angelo - Joshua - Liam - Joao

23/05/2022	Start Date
0	Days till report submission
5	Days till demo day
1	Display Week:

TASK	PROGRESS	START	END	DAYS
Command (Shaanuka and Indraneel)				
Learning the basics of NodeJS, Express and React	100%	23/05/2022	28/05/2022	6
Creating a Basic back-end with NodeJS and Expresss	100%	28/05/2022	31/05/2022	4
Creating the FrontEnd UI using React	100%	28/05/2022	17/06/2022	21
Setup AWS Instance and deploy REST API	100%	07/06/2022	09/06/2022	3
Establish working Back-end and Front-end Communication locally	100%	30/05/2022	01/06/2022	3
Setting up a MongoDB Database	100%	01/06/2022	06/06/2022	6
Setup MQTT broker on an AWS EC2 instance and establish 2 way PubSub	100%	07/06/2022	10/06/2022	4
Deploying web app with Nginx proxy server	100%	17/06/2022	18/06/2022	2
Testing full functionality of Web-App	90%	19/06/2022	23/06/2022	5
Control (Angelo)				
Setting up ESP32 with Visual Studio and PlatformIO	100%	01/06/2022	02/06/2022	2
Testing 2-Way Communication between ESP32 and back-end via TCP sockets	100%	02/06/2022	02/06/2022	1
Setup MQTT client on ESP32 and test using MQTT Explorer	100%	02/06/2022	03/06/2022	2
Setup of communication from ESP32 to motors	100%	11/06/2022	20/06/2022	3
Radar function infrastructure setup	100%	07/06/2022	09/06/2022	3
Autonomous functionality	95%	14/06/2022	23/06/2022	10
Setup of communication via SPI between ESP32 and FPGA	95%	10/06/2022	14/06/2022	5
Written Code for communicating with all subsystems	100%	01/06/2022	18/06/2022	
Full implementation of coordinate mode	80%	21/06/2022	22/06/2022	2
Drive (Joao)				
Building the base of rover, including wiring of motors and optical sensor	100%	23/05/2022	30/05/2022	8
Making optic sensor angle tracking code	100%	31/05/2022	18/06/2022	19
Testing motors and optical sensor together	95%	20/06/2022	23/06/2022	4
Rotation and Speed Handling	100%	15/06/2022	17/06/2022	
Using gyroscope to track rover's angle	100%	11/06/2022	22/06/2022	
Devising and Testing Agorithm for Complete Rover movement	100%	13/06/2022	27/06/2022	15
Testing Rover Movement using the webserver	100%	20/06/2022	22/06/2022	
Energy (Liam)				
Testing buck SMPS w/ the Solar Panels	100%	01/06/2022	06/06/2022	6
Investigating MPPT	100%	07/06/2022	07/06/2022	1
Writing/Debugging MPPT Code	100%	08/06/2022	10/06/2022	3
Testing MPPT w/ Buck and Solar Panels	100%	11/06/2022	13/06/2022	3
Testing boost SMPS	100%	14/06/2022	17/06/2022	4
Writing code for and testing 2 channel relay module	100%	17/06/2022	19/06/2022	3
Characterise battery	100%	15/06/2022	19/06/2022	5
Calculate compatible battery voltage for MPPT	100%	16/06/2022	17/06/2022	2
Program boost SMPS to select compatible voltage for battery	100%	17/06/2022	21/06/2022	5
Test SD card	100%	21/06/2022	21/06/2022	1
Ensure all subsystem function together	85%	21/06/2022	22/06/2022	2
SOT code and time-till-recall estimator	90%	20/06/2022	21/06/2022	2
Radar (Joao)				
Preliminary testing of radar (frequency) and understanding of physics behind it	100%	01/06/2022	03/06/2022	3
Building pre-amp of radar to be able to be read by microcontroller (failed)	100%	03/06/2022	04/06/2022	2
Learning Arduino programming to detect fan and servo motor (failed attempt)	100%	05/06/2022	12/06/2022	8
Revising technique - building a circuit that outputs directly a digital high signal when detected rather than using Arduino	100%	14/06/2022	16/06/2022	3
Building bandpass filter with amplification circuit. Simulating on LTspice	100%	17/06/2022	17/06/2022	1
Designing & Building rest of circuit to output high signal when detected (peak-detector, comparator, voltage divider at output)	100%	18/06/2022	19/06/2022	2
Vision (Shaheen)				
Run basic image process test	100%	23/05/2022	23/05/2022	1
Study and understand basic test code	100%	23/05/2022	06/06/2022	15
Add support for all coloured balls (fine tune HSV values)	90%	25/05/2022		
Implement gain loop algorithm	100%	26/05/2022	01/06/2022	7
Implement median filter algorithm - revised	30%	02/06/2022	08/06/2022	7
Implement posterization algorithm - revised	60%	26/05/2022	08/06/2022	14
Implement mode filter algorithm	100%	09/06/2022	12/06/2022	4
Implement bounds detection	100%	10/06/2022	20/06/2022	11
Add support for buildings	100%	19/06/2022	21/06/2022	3
Distance calculation	100%	04/06/2022	14/06/2022	11
Angle Estimation	100%	12/06/2022	13/06/2022	2
Communication of distance and ball data to Control	100%	10/06/2022	14/06/2022	5
Report	100%	03/06/2022	21/06/2022	19
Integration (All)				
Setup GitHub and shared OneDrive	100%	23/05/2022	23/05/2022	1
Integrating MongoDB with backend	100%	04/06/2022	06/06/2022	3
Integrating MQTT into Backend server with Simultaneous PubSub	100%	11/06/2022	16/06/2022	6
Complete Communication between Front-end, Backend and Rover	100%	15/06/2022	22/06/2022	8
Communication across all systems	100%	25/05/2022	21/06/2022	28



Gantt Chart

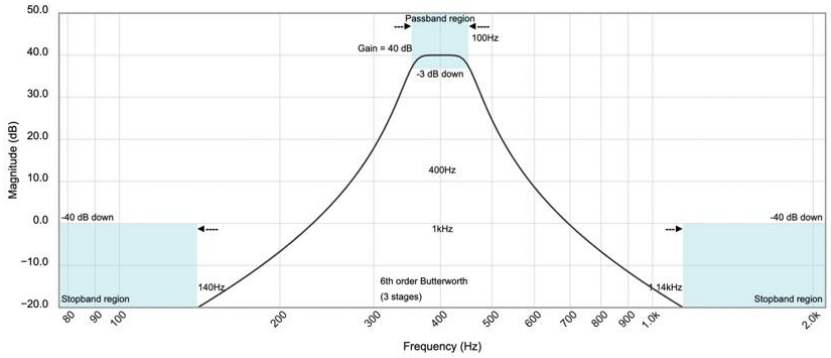


Figure 1 – Magnitude bode plot of bandpass filter

```
void radarDetection(int roverX, int roverY, int roverAngle){
  myservo.write(theta);
  if (digitalRead(RADAR_INPUT) == HIGH){
    //angle calculation will be: roverangle + (theta - 90): theta is from 0 to 180.
    int detectionAngle = roverAngle + (theta-90);
    int distance = RADAR_HEIGHT * tan(RADAR_ANGLE);
    int xcoord = distance*sin(detectionAngle);
    int ycoord = distance*cos(detectionAngle);
    String JSON = "{\"Obstacle\" : \"Fan\"\nX : " + String(xcoord) + "\nY : " + String(ycoord) + "\n}";
    char* name = "Obstacle";
  }
  if (direction == 1)
  {theta+=2;}
  if (direction == 0)
  {theta-=2;}
  if (theta >= 180)
  {direction = 0;}
  if (theta <= 0)
  {direction = 1;}
```

Code that controls servo motor and radar

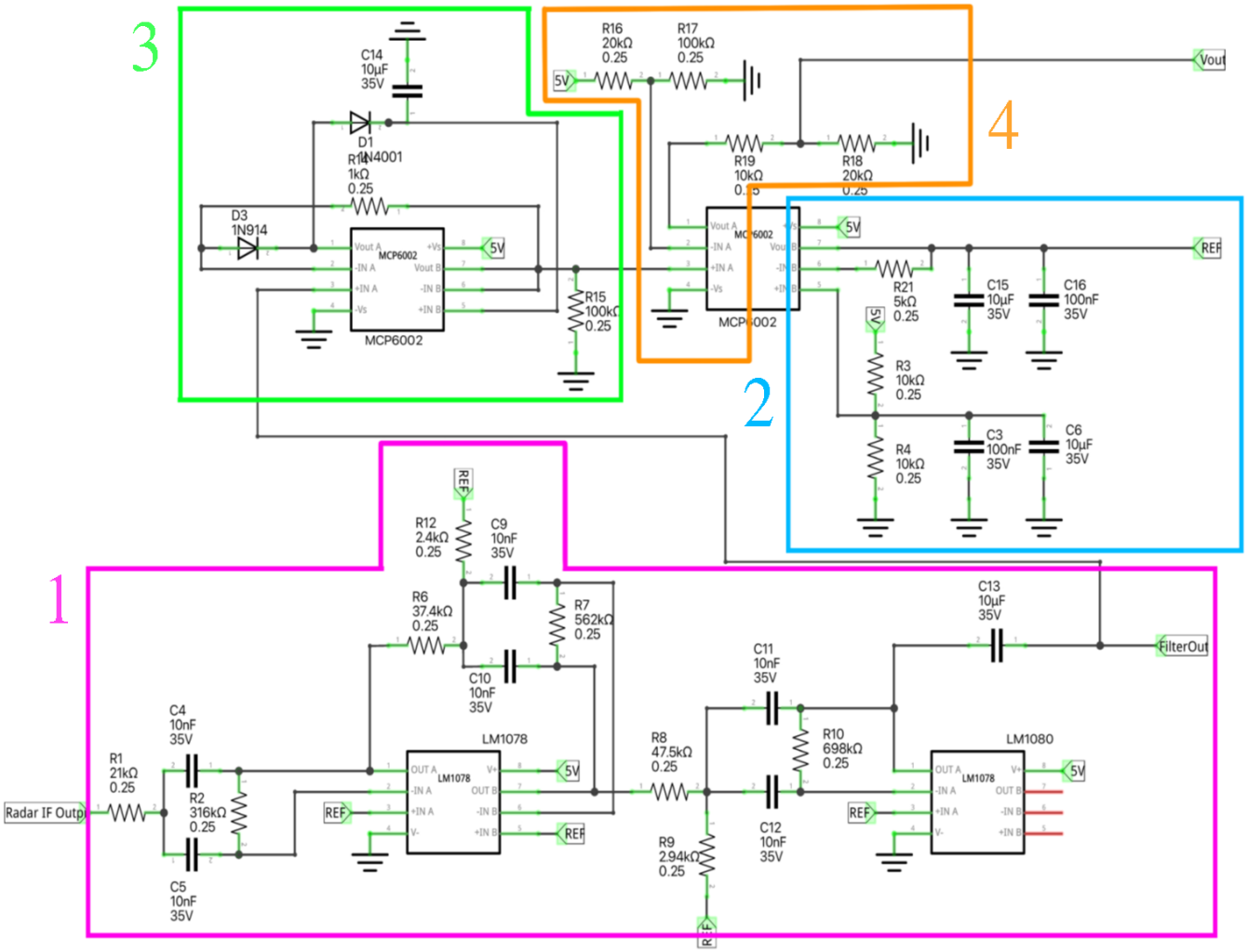


Figure 3 – Radar module circuit diagram with different sections outlined