Imperial College London

Second Year Project 2023: EEEBalance Bug

Group 2

Andres Martinez Llorente, Anish Narain, Awais Khawaja, Janvid Wu, Pengyuan Shao, Riya Chard

Department of Electrical and Electronic Engineering
Imperial College London
2023

Contents

1 Introduction **1**

1.1 Abstract 1

1.2 Acknowledgements 1

1.3 System Design 2

1.4 Teamwork 3

2 Implementation and Results **5**

2.1 Computer Vision 6

2.1.1 Hardware Description 6

2.1.2 Design Choices 6

2.1.3 Video Input and Buffering 7

2.1.4 Colour Segmentation and Beacon Detection 8

2.1.5 Beacon Distance Measurement 11

2.1.6 Front Wall Detection 12

2.1.7 Message Communication 13

2.1.8 Beacon Recalibration 14

2.2 Application 16

2.2.1 Client Application 16

2.2.2 Communication 17

2.2.3 Maze Traversal 20

2.3 Energy System 28

2.3.1 PV Panel characterisation and Emulation 28

2.3.2 PV Power Output Optimisation 31

2.3.3 Testing MPPT Performance 33

2.3.4 DC grid Management 34

2.3.5 LED Drivers 36

2.3.6 Optimisation 38

2.3.7 LED Driver Testing 39

2.3.8 WiFi Considerations 39

2.3.9 Comparison 40

2.4 Balancing 41

2.4.1 Controller Design and Implementation 41

2.4.2 Sensor Fusion 41

2.4.3 Microstepping 42

2.4.4 Multiprocessing 43

2.4.5 Cascaded Control 43

2.4.6 Tuning Parameters 44

2.5 Chassis Design 45

3 Conclusion **49**

3.1 Conclusion 49

3.2 Improvements 50

A Appendix **51**

Bibliography **53**

1 Introduction

Contents

1.1 Abstract 1
1.2 Acknowledgements 1
1.3 System Design 2
1.4 Teamwork 3

1.1 Abstract

The aim of this project was to produce an autonomous rover that: moves through an illuminated maze, surveys the layout, produces a map of the discovered layout, finds the shortest path through the maze and balances on two wheels. This required successful implementation of a robust mechanical design with a controller to maintain balance; computer vision to detect LED walls and beacons for navigation and dead reckoning; an energy system for powering the beacons; a traversal algorithm to navigate through the entire maze; designing a communication system from the client application to the onboard ESP32, using a webserver deployed on AWS; and a user interface which displays the maze traversal.

1.2 Acknowledgements

Dr Hakan Merdan, Course Lecturer: Teaching Fellow, Electrical and Electronic Engineering, Imperial College London.

1

Dr Edward A. Scott, Course Lecturer: Senior Teaching Fellow, Electrical and Electronic Engineering, Imperial College London.

Mrs Esther P. Borobio, Course Lecturer: Principal Teaching Fellow, Electrical and Electronic Engineering, Imperial College London.

Dr Philip Clemow, Course Lecturer: Senior Teaching Fellow, Electrical and Electronic Engineering, Imperial College London.

And to all lab assistants as well as undergraduate and postgraduate teaching assistants.

1.3 System Design

The project was broken down into the following self-contained parts, as shown in 1.1:

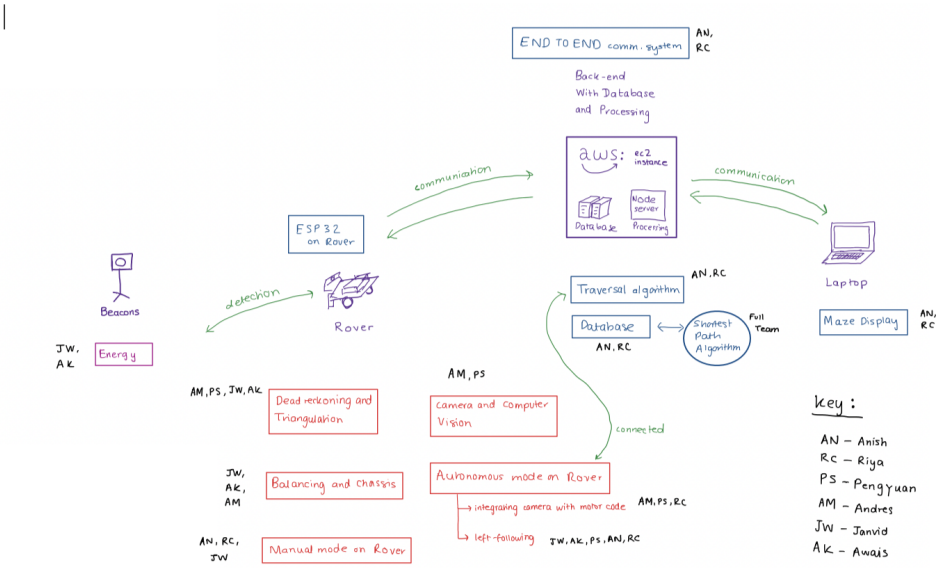


Figure 1.1: Division of project into submodules and the team members assigned to work on them

The project consisted of four main teams: Application, Energy, Computer Vision, and Control. In the first weeks, the group divided the work amongst these teams:

The Computer Vision team focused on wall following and beacon detection. The camera sensor was integral in extracting information about the local region around the rover for collision prevention, as well as detecting beacons for verifying the rover's coordinates. Advanced techniques such as colour segmentation and colour space mapping had to be applied.

The Application team developed the interface, communication, and navigation functionalities. Their goal was to establish communication between the ESP32 and client app. An application was built using react.js and a webserver was coded in node.js and deployed on an ec2 instance. They also aimed to develop an efficient traversal algorithm.

The Energy team worked on the PV panels and SMPS systems. The beacon and energy system was designed to calibrate the rover's position as it traversed the maze. The main challenge was making the system robust enough to maintain power throughout the duration of traversal and in varying lighting conditions.

Lastly, the control team was in charge of producing a controller to balance the chassis and traverse the maze. The controller for the stepper motor drivers had to be implemented digitally on the ESP32. The capabilities of the drivers and the microprocessor board were stretched through microstepping, multiprocessing and filtering.

As the project progressed into the final weeks, the team members began to integrate the sub-modules. Regular communication was maintained to ensure consistency across the entire project. The teams coordinated their efforts to ensure that the different components worked together seamlessly.

1.4 Teamwork

The project utilised Notion, a project management tool, to facilitate collaborative teamwork. Sprint planning was conducted on a weekly basis, a task list for each individual was created for the upcoming "sprint", representing one week of work. This approach ensured that all team members had a clear understanding of the tasks that needed to be accomplished and allowed them to coordinate their efforts effectively.

The sprint planning method also promoted equal contributions from all team members. Once a team member completed their assigned tasks, they had the opportunity to either start working on tasks scheduled for the next sprint or provide support to their fellow team members. This ensured a balanced distribution of workload and fostered a collaborative environment.

1

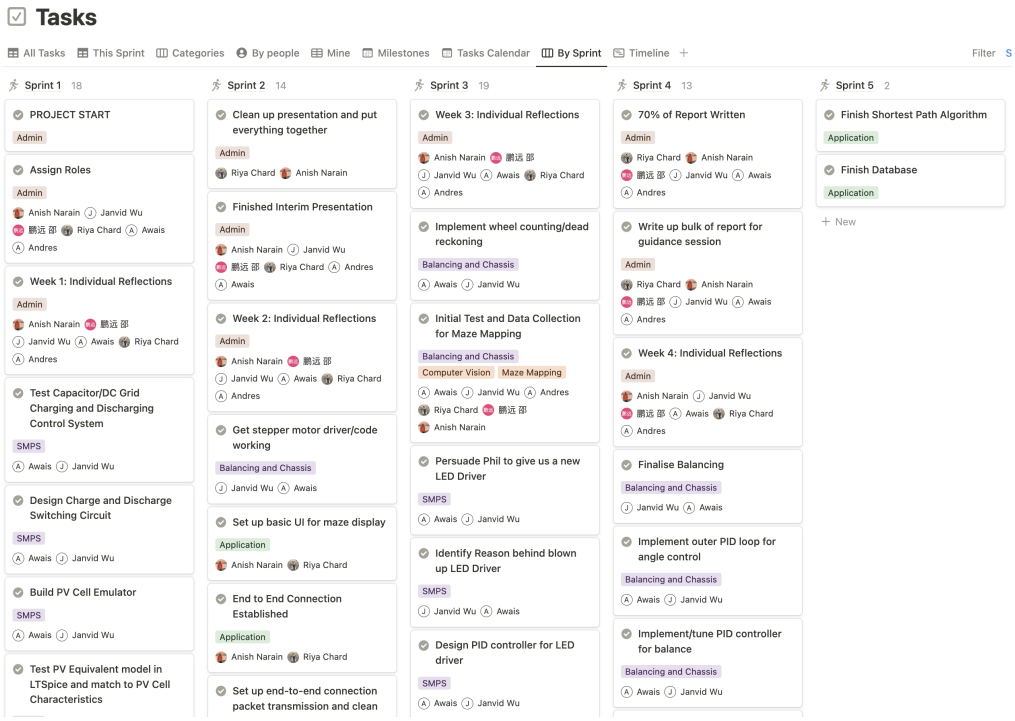


Figure 1.2: Notion app task manager

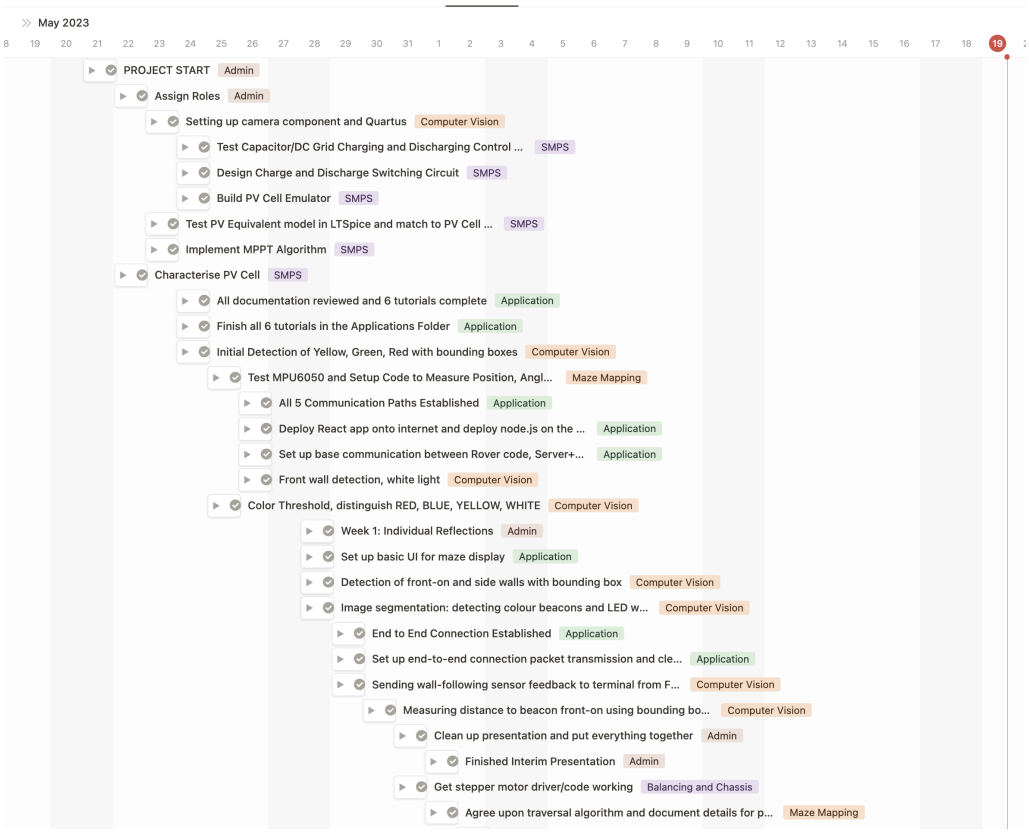


Figure 1.3: Task Gantt chart

2

Implementation and Results

Contents

2.1 Computer Vision **6**
2.1.1 Hardware Description 6
2.1.2 Design Choices 6
2.1.3 Video Input and Buffering 7
2.1.4 Colour Segmentation and Beacon Detection 8
2.1.5 Beacon Distance Measurement 11
2.1.6 Front Wall Detection 12
2.1.7 Message Communication 13
2.1.8 Beacon Recalibration 14
2.2 Application **16**
2.2.1 Client Application 16
2.2.2 Communication 17
2.2.3 Maze Traversal 20
2.3 Energy System **28**
2.3.1 PV Panel characterisation and Emulation 28
2.3.2 PV Power Output Optimisation 31
2.3.3 Testing MPPT Performance 33
2.3.4 DC grid Management 34
2.3.5 LED Drivers 36
2.3.6 Optimisation 38
2.3.7 LED Driver Testing 39
2.3.8 WiFi Considerations 39
2.3.9 Comparison 40
2.4 Balancing **41**
2.4.1 Controller Design and Implementation 41
2.4.2 Sensor Fusion 41
2.4.3 Microstepping 42
2.4.4 Multiprocessing 43
2.4.5 Cascaded Control 43
2.4.6 Tuning Parameters 44
2.5 Chassis Design **45**

2.1 Computer Vision

The vision module of the rover serves as a crucial component for recognising different coloured beacons within the arena, as well as identifying the white LED lights that function as the maze's walls. This module utilises live streaming from a camera, processing each frame to extract RGB values, and implementing image processing algorithms to detect and categorize various objects based on their colours. The resulting meaningful outputs, such as distance to beacons and coordinates recalibration, are then transmitted to other modules for further processing.

2.1.1 Hardware Description

The rover's vision system incorporates several hardware components. It consists of a DE10-lite FPGA board, which serves as the main processing unit. The FPGA board is connected to the Terasic D8M-GPIO camera package, featuring an OV8865 image sensor. Communication between the image sensor and the FPGA board is established using GPIO pins. Additionally, the DE10-lite FPGA board is equipped with a 4-bit VGA output, allowing for real-time visualization and debugging of object detection. Furthermore, the FPGA board includes a connector compatible with an Arduino Uno R3 shield, which facilitates seamless integration with the ESP32 used for control purposes.

2.1.2 Design Choices

A frame-buffered image-processing project was already provided. Therefore, the project began by setting up the camera and using the provided RGB colour thresholding and bounding boxes to capture different colour beacons. The three beacons provided are red, blue, and yellow.

The initial design choice used RGB values to distinguish between all three beacons, using the demo code from the starter project. However, the red and yellow spectrums overlap in RGB so the pure RGB threshold was not enough to distinguish between beacons. It is necessary to do some pre-processing on the incoming RGB values to better distinguish the colours. The decision was made to change the RGB colour space into a HSV colour space (hue, saturation, value). HSV colour space calculates difference in RGB values and spreads different colours into a 360-degree spectrum, making red and yellow further apart in the spectrum and easier to differentiate. Saturation and value are also useful to determine the brightness and purity of the colour.

In the context of wall detection, the initial design choice was to apply an edge detection filter to differentiate the bright and dark regions. A convolution matrix or kernel in image processing was useful for this purpose. Specifically, a 3x3 matrix was convolved with a similar 3x3 range of pixels on the graph, resulting in the creation of edge detection images. It was subsequently determined that this method was not applicable to our starter project due to the line-by-line output of pixel values, which posed significant challenges in accessing a 3x3 area of pixel values simultaneously. As a result, the decision was made to employ colour detection (RGB) as a means of wall detection, similar to beacon detection.

To utilise the camera in maze wall detection, the initial consideration was to position the camera vertically with a slight downward tilt. This setup aimed to enable the rover to capture both the maze and the beacons while minimising the detection of the background environment. However, it was quickly realised that placing the camera at a high position above the ground cannot accurately detect the wall directly in front of the rover. Testing showed that when the camera was positioned 15cm above the ground, it could only perceive walls located approximately 0.3m away from the rover. This discrepancy posed challenges in designing an effective maze traversal method since the information about the maze walls at the current position was delayed by 0.5m. To address this issue, the decision was made to lower the camera and invert its orientation, placing it only 3cm above the ground. This adjustment proved to be beneficial for both front wall detection and beacon recognition. However, this method does come with a drawback: the camera loses the ability to capture information about side walls. To compensate for this limitation, other sensors are employed to capture the necessary data regarding side walls.

2.1.3 Video Input and Buffering

The video input and buffering are based on the starter project “EE_IMGPRO.v” provided by Dr Ed Stott. The system uses the D8M Camera as the video input source and various IP components from the Intel Video and Image Processing Suite. One of the key components used is the Clocked Video Output, which facilitates the conversion of raw data captured by the camera in Bayer filter form into RGB video packets. These video packets have a resolution of 640x480 pixels, ensuring compatibility with standard video formats. Additionally, the system incorporates the Frame Buffer Core, which plays a crucial role in buffering the frames. This buffering mechanism enables the camera and output components to operate at different frame rates, ensuring smooth and synchronized processing.

2

2.1.4 Colour Segmentation and Beacon Detection

To ensure accuracy in colour segmentation, the decision was made to utilize the HSV (hue, saturation, value) colour domain instead of the RGB colour domain. This choice necessitates a conversion between the two-colour spaces. The traditional HSV colour space has H value defined as 0-360 degrees, S and V values defined between 0 and 1. However, to avoid complexity and maintain consistency with the RGB colour space. The decision was made to convert all HSV values into 8-bit numbers ranging from 0 to 255 [1]. This choice avoids the need for floating-point calculations, which are generally less efficient for FPGA implementations. By employing this conversion equation, it ensures that the HSV values align with the 8-bit integer range used for RGB colour representation. The equation of converting from RGB to HSV colour space is shown below.

$$M = \max(R, G, B)$$

$$m = \min(R, G, B)$$

$$\Delta = M - m$$

$$H = \begin{cases} M = 0, & 0 \\ \Delta = 0, & 0 \\ M = R, & 0 + \frac{43 \times (G-B)}{\Delta} \\ M = G, & 85 + \frac{43 \times (B-B)}{\Delta - B)} \\ M = B, & 171 + \frac{43 \times (n-G)}{\Delta} \end{cases}$$

$$S = \begin{cases} M = 0, & 0 \\ \Delta = 0, & 0 \\ \text{otherwise} & \frac{255 \times \Delta}{M} \end{cases}$$

$$V = M$$

RGB to HSV Equations

```
module RGB_to_HSV (R,G,B,H,S,V);

input wire [7:0] R;
input wire [7:0] G;
input wire [7:0] B;
output wire [7:0] H;
output wire [7:0] S;
output wire [7:0] V;

wire [1:0] fmax;
wire [7:0] cmax;
wire [7:0] cmin;
wire [7:0] C;
wire [15:0] t0;
wire [15:0] t1;

assign cmin = (R < G & R < B) ? R : (G < B) ? G : B;
assign cmax = (R > G & R > B) ? R : (G > B) ? G : B;
assign C = cmax - cmin;

assign H = (cmax == 0 | C == 0) ? 8'b0 : (cmax == R) ? (43*(G-B)/C)
         |: (cmax == G) ? ((43*(B-R)/C)+85): ((43*(R-G)/C)+171);

assign S  = (cmax == 0 | C == 0) ? 8'b0 : ((255*C)/cmax);

assign V = cmax;

endmodule
```

Figure 2.1: Hardware implementation of RGB to HSV conversion

Figure 2.1 is a hardware module created based on the equations stated, which performs the conversion of RGB colour values to HSV colour space. It takes 8-bit inputs R, G, and B, representing the red, green, and blue colour channels respectively. The resulting HSV values, H, S, and V, are also 8 bits each. The division operation is undesirable in FPGA operations as it is computationally expensive. However, division is unavoidable when performing HSV colour space conversion. Although the division operation is present, it is worth noting that it is performed between two integer values and the result is subsequently rounded down to the nearest integer, as the output is an 8-bit value. This allows for a more efficient implementation compared to using floating-point operations directly. Actual testing shows that the FPGA can effectively calculate the values.

The starter project includes some initial lines of code that can adjust the overall gain and exposure. Upon referring to the OV8865 image sensor datasheet, the separate RGB gains can be adjusted. A post-adjusting picture is then sent through the 8-bit VGA port. The subsequent image is processed using the MATLAB Colour Thresholding tool in the Image Processing Toolbox [2], shown in figure 2.2. This tool enables users to apply a mask for colour segmentation and calculate thresholds for different channels to filter out background colours.

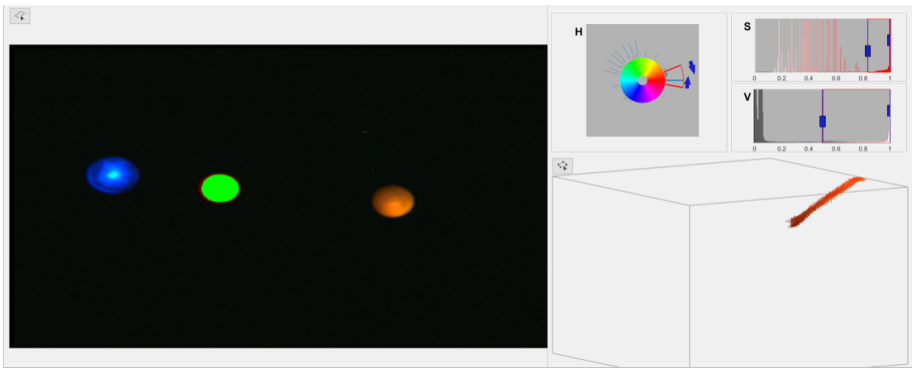


Figure 2.2: Matlab colour thresholding tool

The accuracy of the colour thresholding is not optimal due to several reasons. Firstly, the picture is captured from a 4-bit VGA port, whereas the actual camera stores 24-bit colour. This results in a significant disparity in colour resolution between the actual camera and the captured image. Secondly, the environment where the beacons are placed also affects the thresholding process. Bright and dark environments require different sets of thresholds. Sunlight and white walls will also affect the white light detection, giving false readings in the background. In our case, the lowest exposure and overall gain were chosen to simulate a dark environment, filtering out the background, while the beacons and walls are still sufficiently bright for detection. Table 2.1 shows

the HSV values set for each colour.

Colour	Hue (0-255)	Saturation (0-255)	Value (0-255)
Red	0~10	120~255	50~255
Yellow	10~35	100~255	50~255
Blue	132~212	120~255	50~255

Table 2.1: HSV threshold

After applying the HSV colour thresholding, most colour segments are accurately recognised. However, there are instances where noise is present within the detected areas, leading to errors. Specifically, certain pixels that should be identified as a specific colour, such as a region of blue, maybe falsely detected as another colour, like yellow or even no colour at all. These discrepancies introduce inaccuracies in the overall colour detection process. What is even more critical is that these falsely detected pixels affect the size of the bounding box. Due to certain pixels being wrongly classified as the incorrect colour, the size of the bounding box may become inaccurate or excessively large. This situation makes it difficult for the bounding box to accurately enclose the target region, posing challenges for subsequent distance calculation.

A model filter was proposed to mitigate the impact of falsely detected pixels. The filter employs a counter mechanism for each colour pixels. The maximum and minimum pixel coordinates of each colour's bounding boxes are only updated when a consecutive sequence of eight pixels of the same colour is detected.

For example, as shown in figure 2.3, the camera detects three consecutive blue pixels followed by two yellow pixels, and then an additional ten blue pixels. In this case, the two yellow pixels are ignored since there are only two of them. The bounding box is then drawn based on the first blue pixel and the last one. By implementing this model filter, anomalies are successfully filtered out, resulting in the generation of highly accurate bounding boxes. As shown in figure 2.4, the blue beacon has red and yellow noise amongst the blue pixels. Due to the implementation of the model filter, only the blue bounding boxes is drawn on the screen.

An optimal balance has been achieved between the number of consecutive pixels (threshold) and its impact on the sizes of the bounding boxes. Setting the threshold too high would result in the exclusion of the top and bottom pixels from the bounding box, thereby compromising the accuracy of distance measurements to the beacons. Furthermore, beacons located at a greater distance from the camera would exhibit fewer consecutive pixels than the threshold, leading to their detection as noise. To mitigate this issue, a threshold of 8 pixels was chosen, as it effectively

filters out most of the noise while still enabling the detection of beacons located up to 4 meters away.

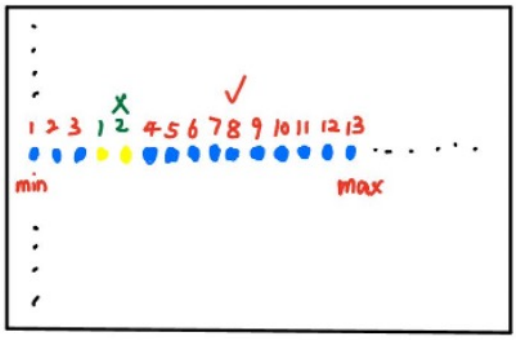


Figure 2.3: Model filter illustration

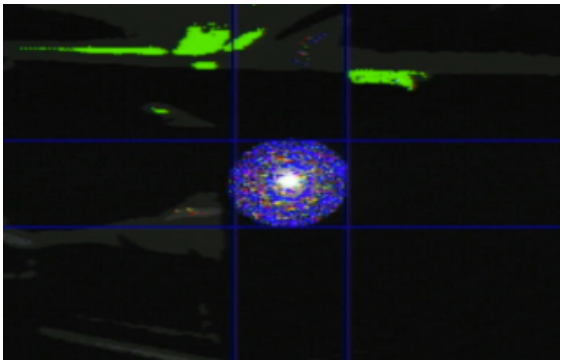


Figure 2.4: Model filter applied on blue beacon

2.1.5 Beacon Distance Measurement

After accurately defining the bounding boxes, an algorithm was proposed to determine the distance to the beacons. Due to perspective, the size of the bounding box gets smaller as the distance to the beacon increases. Thus, the distance can be measured using the size of bounding boxes. Initially, the design choice was to use the width of the bounding box as a reference. However, a drawback was encountered when the camera was not adequately aligned with the beacon, leading to inaccuracies. To address this issue, the height of the beacons was now utilized, as it offers greater accuracy from all viewing angles.

To successfully recognize beacons located at significant distances, the camera incorporates a zoom function. When the "measure beacons" action is triggered, the camera zooms in to level 2. After the action is completed, the camera zooms back out to level 1. The reference height of all three beacons is taken at 60cm away from the camera. Table 2.2 presents the reference values for each beacon.

Beacon	Reference height at 60 (pixels)	Height shifted up by 10 bits
Red	79	80896
Blue	75	76800
Yellow	62	63488

Table 2.2: Reference height of beacons

Subsequently, a straightforward inverse function is employed to compute the precise distance from the beacons. It is important to emphasize that this calculation takes place within the Nios2

processor, which is not optimized for handling floating-point operations. To ensure accurate distance measurements, a strategy is implemented wherein the reference height is shifted upwards by 10 bits (equivalent to multiplying by 1024 in decimal) while performing the calculation. Finally, the calculated result is shifted downwards by 10 bits to restore the accurate distance value. The code snippet is shown in figure 2.1.5. This adjustment compensates for the limitations of the Nios2 processor in dealing with floating-point operations and maintains the accuracy of the distance calculation. The measurement was tested to be fairly accurate, with an error within 5% of the actual distance.

$$\text{dist} = (\ (\text{height } 60/\text{h}) * 600) \gg 10$$

Figure 2.5: Code for distance calculation

2.1.6 Front Wall Detection

In addition to beacon detection, the detection of the front wall is crucial for subsequent traversal along the left wall. The camera plays a vital role in identifying the presence of a front wall by detecting the colour white using the RGB colour space. A designated region of the screen is used as the detection area, represented by a bounding box. If any white light is detected within this bounded area, the rover can infer the existence of a wall in front of it. 2.6 and 2.7 illustrates the mechanism for front wall detection, with white light being visualized as green for clarity. The bounding box is positioned at the top of the screen, and an orange line is displayed to indicate the highest white pixel within the bounding box. The bounding box is specifically set 120 pixels down from the top of the screen. With these settings, the rover can detect the presence of a wall when it is approximately 5cm away, providing sufficient time for the rover to execute a turn and avoid collision.

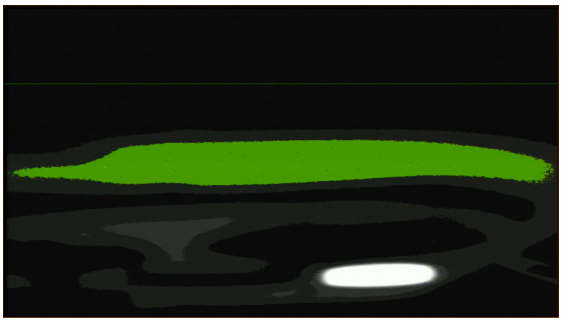


Figure 2.6: Front wall not detected

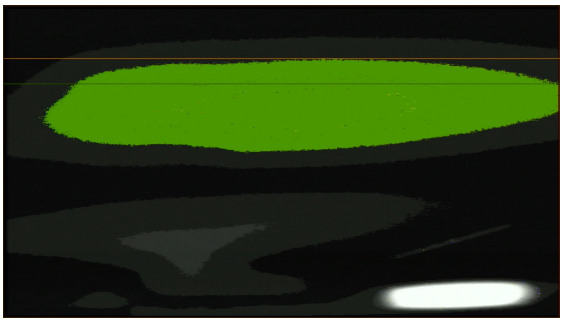


Figure 2.7: Front wall detected

2.1.7 Message Communication

The communication between the FPGA hardware and software sides, as well as between the FPGA and ESP32, involves the exchange of messages. Each message comprises 4 bytes, equivalent to 8 hexadecimal digits. The messages transmitted from the FPGA hardware and software sides consist of the following (Table 2.3): a start of message indicator (RBB), three bounding box coordinates conveyed in 6 messages, and one message indicating the presence or absence of a front wall, denoted by a value of 0 or 1. These eight messages are then processed within the FPGA software core to calculate the distance to the beacon. As only one beacon will be detected in each frame, the software core sends a single message containing the closest beacon's distance to the ESP32. This message is designed to utilise the highest 1 byte to represent the serial number of the detected beacon, while the lowest 3 bytes are allocated for the distance measurement. Therefore, three messages are transmitted from the FPGA to the ESP32 (Table 2.4): the start of message indicator (RBB), the distance to the beacon, and the front wall detection status.

Start (RBB)	Red xmin,ymin	Red xmax,ymax	Blue xmin,ymin	Blue xmax,ymax	Yellow xmin,ymin	Yellow xmax,ymax	Front wall detection
00524242	00db0060	02440099	0133007b	015800a5	00e20065	010e0095	00000001

Table 2.3: Messages outline from FPGA hardware to software

Start (RBB)	Beacons distance (yellow, 840mm)	Front wall detection (detected)
00524242	02000348	00000001

Table 2.4: Messages outline from FPGA to esp32

Since zoom in and out are needed for beacons recalibration, a two-way communication between the FPGA and the ESP32 is needed. Messages are transmitted from the FPGA to the ESP32 using the UART ports on the FPGA. The FPGA sends 4 bytes of data at a time. The ESP32, in turn, reads 4 bytes and decodes the raw byte messages into meaningful integer values. These values are then utilised for further traversal operations. A similar method is implemented for transmitting messages from the ESP32 to the FPGA. However, it was soon discovered that the "fread" function was blocking the main loop and sending when there was no data available to be read. To address this issue, a solution was found by setting the file opened to the UART port to null blocking mode using the "fcntl" function. This adjustment ensures that if no data can be read, the "fread" function will return 0, allowing the while loop to continue running without interruptions.

2

2.1.8 Beacon Recalibration

The rover's position is determined using the dead-reckoning method, which combines wheel count and bearing information. This method provides correct initial x and y coordinates for the rover. However, as time progresses, the position may deviate due to various factors, such as wheels getting stuck against a wall or bearing drift caused by noise. To mitigate these issues, a recalibration system was implemented to update the x and y coordinates.

The recalibration process involves the rover performing a 360-degree turn to locate all three beacons positioned around the three corners of the arena. The distances to these beacons are then calculated. With knowledge of the height and width of the arena, two triangles can be constructed using three known side lengths. By applying the cosine rule, the system can determine the angles within the triangles and subsequently calculate two sets of x and y coordinates. These coordinates are then averaged to yield the current x and y position output. Figure 2.8 shows the equations used.

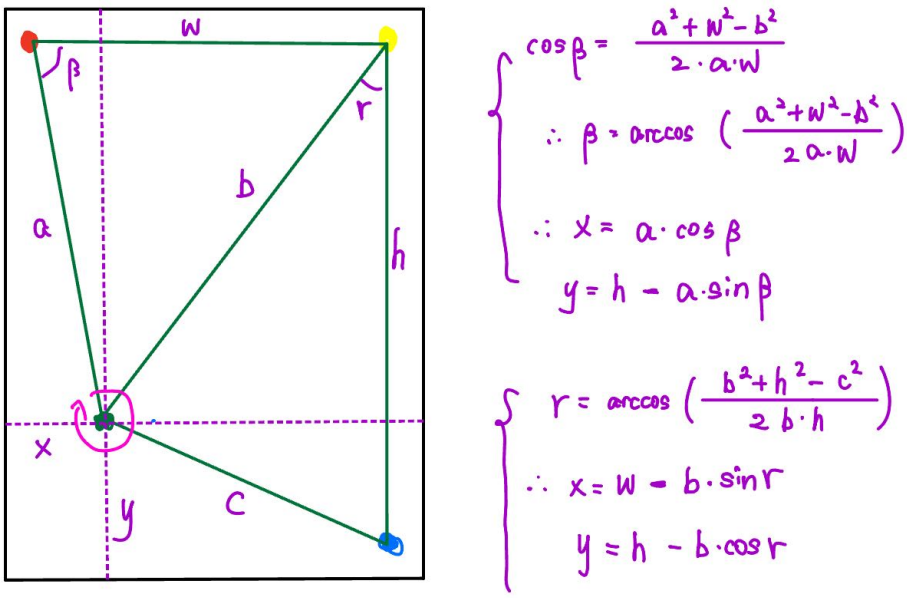
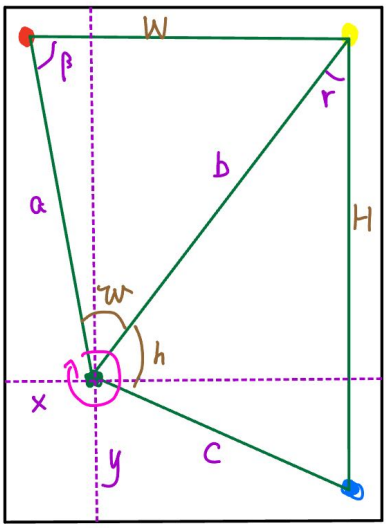


Figure 2.8: Illustration of beacon recalibration (old)

However, due to the inaccuracy in measurements, the calibration outputs x and y are sometimes NAN as the method is trying to arccos an invalid value. A new method was proposed later to use two measured side and the turning angle to calculate coordinates (figure 2.9). This method avoids NAN values and always gives out a set of coordinates when two beacons are detected.

Based on current testing, the optimal frequency for beacon recalibration is every 2 minutes. This interval maintains a good balance between accurate positioning and minimising unnecessary recalibration. The error in the x and y coordinates after recalibration is within 3%, ensuring reliable and reasonably precise position updates.



$$\begin{cases} W = a^2 + b^2 - 2ab\cos w \\ \frac{\sin\beta}{b} = \frac{\sin w}{W} \quad \therefore \beta = \arcsin\left(\frac{b \cdot \sin w}{W}\right) \\ \therefore x = a \cdot \cos\beta \\ y = \text{height} - a \cdot \sin\beta \end{cases}$$

$$\begin{cases} H = b^2 + c^2 - 2bc\cos h \\ r = \arcsin\left(\frac{c \cdot \sin h}{H}\right) \\ x = \text{width} - b \cdot \sin r \\ y = \text{height} - b \cdot \cos r \end{cases}$$

Figure 2.9: Illustration of beacon recalibration (new)

2

2.2 Application

2.2.1 Client Application

React App

To display the maze and control the rover, a react app was implemented. The react app connected to a node.js server, which was used as a backend to send coordinates of the rover from the ESP32 to the webapp. The coordinates calculated from the gyroscope and accelerometer were sent to the server, which then forwarded the data to the webapp. React was chosen to display the maze system as it is a dynamic website, which meant that it could quickly and efficiently update while the rover was discovering more of the maze. This is due to React's virtual DOM (Document Object Model) which only re-renders modified areas of the UI.

Buttons

Buttons to control the rover's direction manually were added to the website. Though the intention was for the rover to move autonomously, manual control was added so that other elements of maze mapping and control could be tested. For example, before autonomous movement, manual movement was used to test the quality of the camera's wall detection. The webapp received data for movement based on button clicks, this information for next direction was sent to the server. The server then sent the next direction based on button clicks to the ESP32 which used the next direction received to drive the motors.

Coordinate System

When deciding how to display the maze, the first idea was to use a grid based on a matrix to fill the walls of the maze using 1's to represent walls and 0's to represent paths. However, after seeing the initial layout of the maze, which included curved and slanted walls, an alternative display was implemented. To display the maze, a coordinate system would be used, it displayed the coordinates the rover could traverse to show the paths of the maze. This design was chosen as it would accurately display curved and slanted lines. Figure 2.10 shows the initial and current designs.

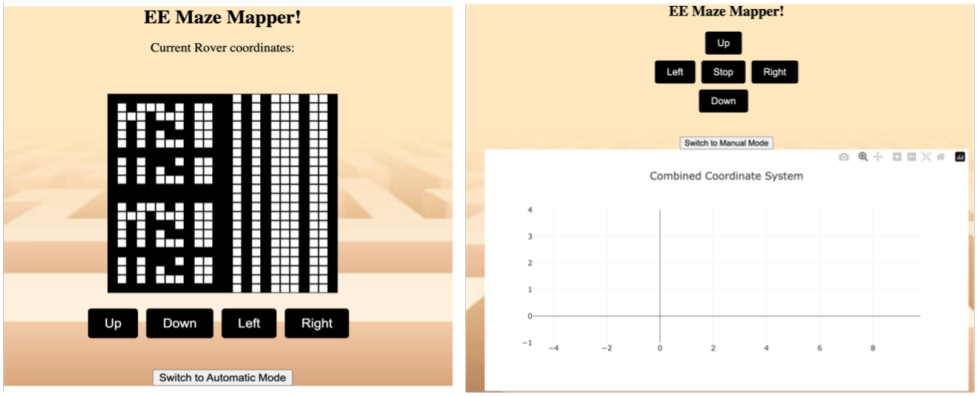


Figure 2.10: Webapp layout before and after

2.2.2 Communication

Webserver and Node.js

Having chosen react.js for the client app, the webserver was coded in the accompanying node.js for ease of development. The server was used to enable communication between the ESP32 on the rover and the client laptop. This was used to make decisions for the rover's traversal of the maze. Since node.js operates on an asynchronous, event-driven architecture, it could handle multiple concurrent requests coming from the rover and client app without blocking the execution of other tasks. Furthermore, the unthreaded nature of node.js allows it to handle a larger number of these multiple concurrent requests by eliminating the overhead associated with creating and managing multiple threads.

Polling, WebSocket and Event Driven Architecture

In the project, various communication methods were considered to enable communication between the ESP32, client application, and web server. Both WebSockets and polling were evaluated for their suitability, but neither were a good fit.

WebSockets were deemed unsuitable due to their limitations in meeting the rapid communication needs of the project. Given the objective of maximizing the speed of the rover as it traverses the maze, it was crucial to have a communication method that could respond quickly to next direction requests. WebSockets, although capable of providing bidirectional and real-time communication, still introduce some overhead due to the underlying protocol and the need for constant message framing. This overhead could potentially hinder the speed required for the rover's operation.

Similarly, polling was found to be inadequate for the project's requirements. Polling involves making frequent requests to the server at regular intervals to check for updates. However, this approach did not align with the need for rapid communication in real-time. The repeated requests introduced delays and increased network traffic, which were not optimal for the desired speed and responsiveness of the system. Additionally, frequent polling would not offer the desired real-time plotting of coordinates traversed by the rover on the client application's coordinate system.

Considering these limitations, a HTTP GET/POST based, event-driven architecture was chosen as the most suitable communication approach for the project. The event-driven architecture excelled in meeting the requirements for rapid and real-time communication. By leveraging an event-driven model, the communication system could respond immediately to events, such as next direction requests from the rover or updates in the rover's position. This approach eliminated the delays and inefficiencies associated with polling and allowed for rapid and responsive communication.

Optimisations on ESP32

The ESP32's communication code was implemented similarly to the react.js client app, with the difference being that it was written in Arduino C code and required the inclusion of specific libraries for json parsing. Initially, the communication code utilized multiple GET and POST requests. However, after establishing an event-based architecture for the communication system, it was decided that GET and POST requests should be triggered by events. This led to a design where a single GET/POST request was made after each if-else condition.

To test the communication code, it was combined with the motor code, and the rover was operated in manual mode using button clicks on the client app. During testing, it was observed that the rover's smooth continuous movement was replaced by jittering and inconsistent motion. The wheels would rotate slightly, pause, and then rotate slightly again. This highlighted two areas for improvement:

1. Multithreading: By running the motor code and communication code on separate cores, the inconsistent movement was resolved. This change ensured that the cost of executing the communication code no longer impacted the motor code's loop.

2. Single GET and POST endpoint: Instead of utilizing five different endpoints, the communication between the rover and webserver was streamlined to only two endpoints. While testing, it was discovered that each GET and POST request incurred a substantial overhead in completing

the loop. By reducing the number of endpoints, the speed of the loop more than doubled.

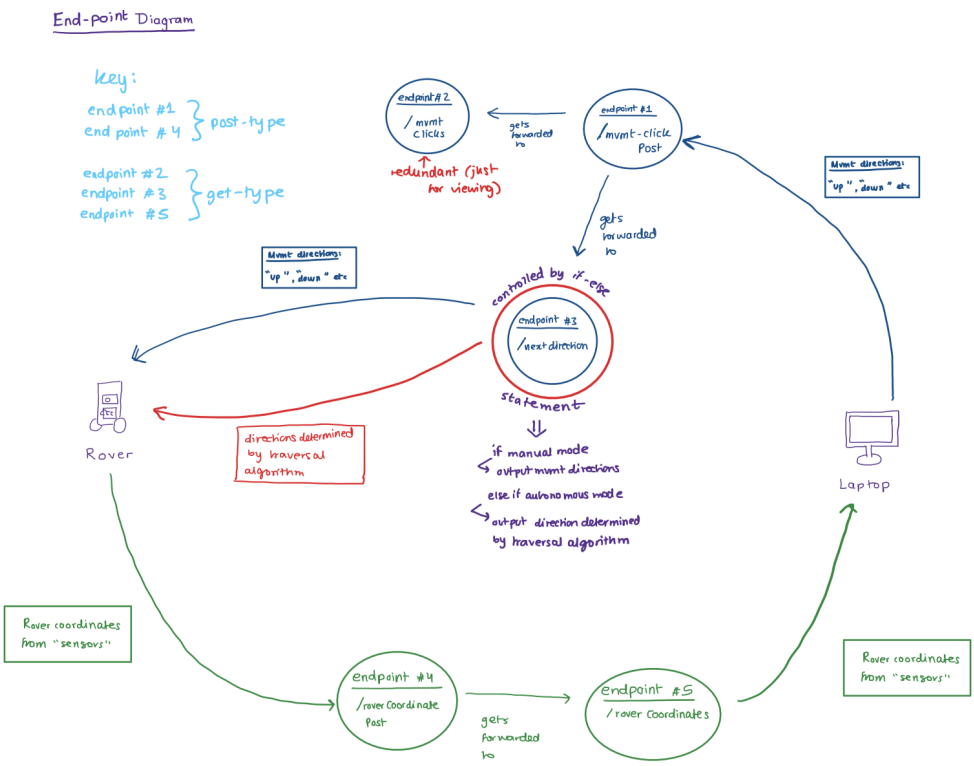


Figure 2.11: 5 GET/POST endpoints

Database

A DynamoDB database was chosen to store coordinates received by the server for two primary purposes: to facilitate information recovery in case of connection failures and to retain past traversals. Initially, a MySQL database was considered, but DynamoDB's NoSQL schema offered greater flexibility, making it a more compelling choice. Communication between the ESP32, webserver, and client application relied on JSON packets. Here, DynamoDB's native support for JSON data structures proved highly advantageous. Storing the JSON packets in DynamoDB required minimal manipulation, thanks to its seamless integration with JSON data. This convenience further reinforced the decision to opt for DynamoDB over MySQL in this specific use case.

2.2.3 Maze Traversal

2

Phototransistors

During the testing of the camera feed on the maze, an issue was discovered related to the camera's angle and its ability to detect both beacons and the LED "walls." Specifically, there was a range problem with wall detection. The camera had a minimum distance of 6cm for detection, which meant that anything closer than that would go undetected. This posed a problem in tight turns where the front wall would fall within the camera's blind spot. Additionally, when examining the maze traversal from the camera's perspective, it became evident that implementing left wall detection would be challenging in regions where the range problem could cause gaps between successive walls to be missed, mistakenly identifying them as continuous walls as shown in figure 2.12

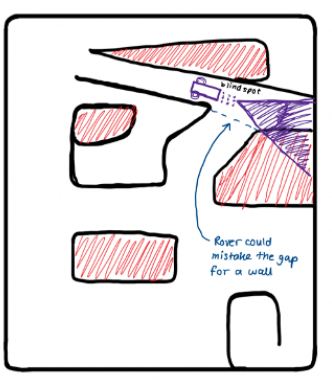


Figure 2.12: Camera left following error

To address the range issue, alternative methods of light detection were explored. Initially, an LDR (Light Dependent Resistor) was considered, but it was found that the distance from the light strip had to be impractically close for any detection to be made by the ESP32. Phototransistors emerged as a better option, as they exhibited higher sensitivity. When light is incident on the phototransistors, current can flow and a resistor converts this to a voltage that can be sensed. A capacitor was also added to form a RC high-pass filter to provide further consistency and stability to the readings by filtering out any high frequency noise. To make the readings robust against ambient light and effective at detecting the LED strips, the phototransistors were positioned low to the ground.

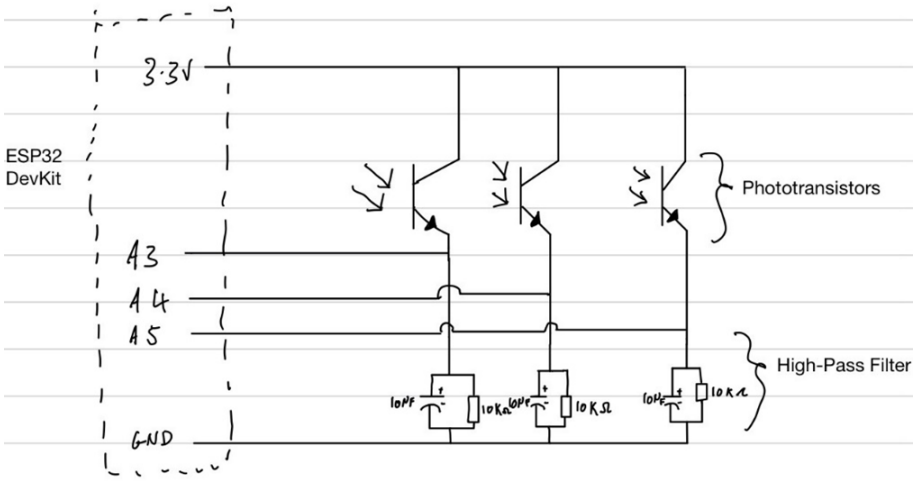


Figure 2.13: Phototansistor circuit

Left Wall Following

To achieve left wall following, the rover was equipped with phototransistors on its left side. Initially, two phototransistors were used, one in the front and one a short distance behind, as shown in figure 2.14. The original idea was to use the difference between the detected light to determine whether the rover should move in a straight line or make a turn. However, it was later discovered that the back transistor was unnecessary for gathering the required information.

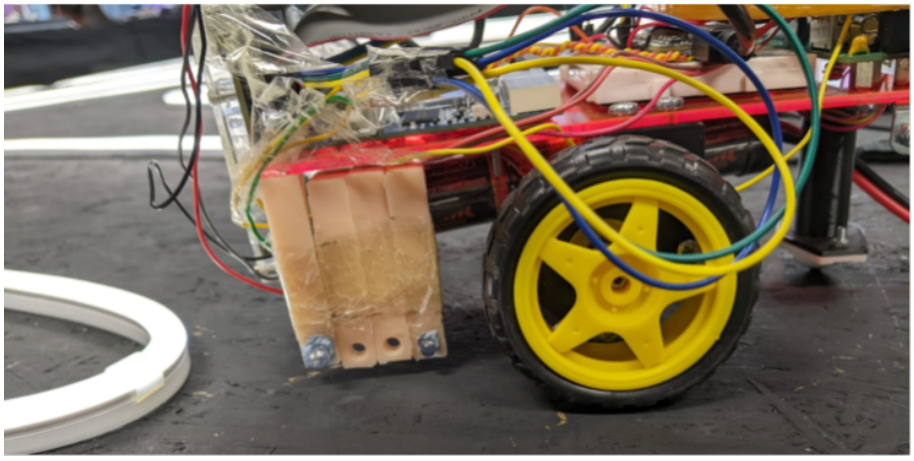


Figure 2.14: Phototansistors mounted on chassis with filter circuit

Based on this finding, the design was revised to use only the front phototransistor. This modification allowed the rover to traverse straight lines effectively. However, during turns, especially tight ones, a problem arose. The front phototransistor would detect less light when passing by a turn, causing the wheels to turn towards the wall. This resulted in the wheels crashing into, or running over the walls. To solve this issue, a secondary sensor was added close to the front of the

wheel, pointing downward, as shown in figure 2.15. This additional sensor would block the rover from moving left if the wheel's trajectory during a turn was heading towards a wall, forcing the rover to turn right instead, thus avoiding a collision that would skew the dead-reckoning .

Another significant issue that emerged was the impact of ambient light on the rover's performance. The high sensitivity of the phototransistors caused problems. The rover would work correctly in the evening when ambient light was minimal but fail to traverse properly in the morning. To overcome this, a third phototransistor was added above the front phototransistor. This additional sensor allowed for recording ambient light changes, which could then be subtracted from the reading of the front sensor. As a result, only the incident light from the LED strip was used to determine the rover's movement, minimizing the influence of ambient light on its performance.

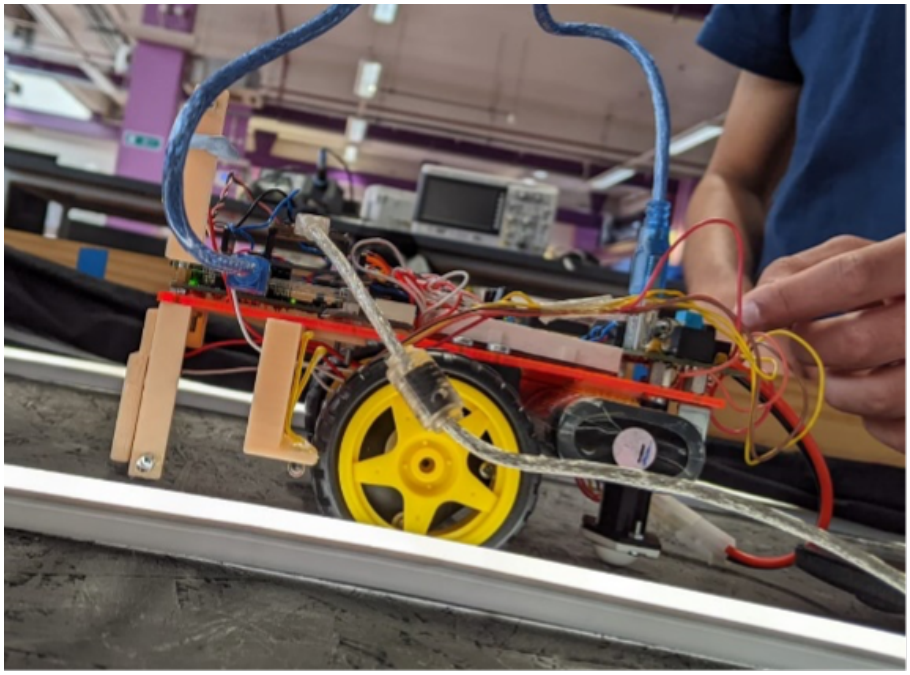


Figure 2.15: Finalised phototransitor set-up

Algorithm

Before seeing the maze layout, the initial idea was to use a depth first approach, relying on making decisions at junctions and exploring every possible route. However, once seeing the maze layout this idea was changed to a left wall following algorithm, as there were slanted and curved walls with no definitive junctions. When the rover would traverse along the left hand side of every new wall it encountered until it returned to its starting position. This relied on the phototransistors to

detect the left side of a wall, as the camera could not detect this at close range.

Once the rover returned to the starting position following left wall traversal an additional algorithm was used to continue traversal through the wall until another new wall was detected which could then be followed. The second algorithm involved splitting the maze up into a grid, so that once a grid had been traversed the rover could look for grids that hadn't yet been traversed to ensure the whole maze had been travelled through. The blue markers on the outside of the maze were used as reference and the layout was split into 54 blocks, 9 by 6. These measurements were chosen as they were similar size to the rover, so that the grids weren't too small that the rover would try to do unnecessary traversal but also not too big so that it would avoid key areas thinking they were already traversed. These ideas are illustrated in figure 2.17.

A second algorithm involved checking each possible direction (up, then right, then left, down and finally diagonals) for walls and if that grid had already been traversed. The algorithm prioritized traversing directions that led to non-traversed areas to discover more of the maze, while avoiding directions that pointed to a wall. The checking was implemented with the intent that the rover would turn on the spot to pick a direction then once an orientation was chosen it would move straight until another wall was detected. This method relied on the front wall detecting camera to navigate the maze until a new wall was found, at which the rover could then follow it along the left wall to map the island. The flowchart in figure 2.16 shows the logic behind this algorithm.

Solving Infinite Loops and Dead Regions

Additionally, the traversal approach had to overcome two problems: the rover entering an infinite loop, and mapping dead regions (parts of the maze that could never be reached). To avoid looping, the server keeps track of coordinates already traversed, so if the rover is in left-following mode and it returns to the starting position, the left-following ends. Furthermore, when the rover has recognised it is in an already traversed grid, it checks for directions in a random order to escape. This avoids the pitfall of being trapped in pre-determined consecutive directions (e.g., right to left to right and so on).

To identify dead regions, it was observed that some grids may not be possible to traverse but the grids around it would be. Two main types of dead regions were identified: dead regions at the boundaries of the maze, and dead regions within islands on the inside of the maze. When left following the outer loop for the first time, if, due to obstacles, boundary grids were unreachable, the algorithm recognise them as dead regions. For dead regions in islands, it was assumed that

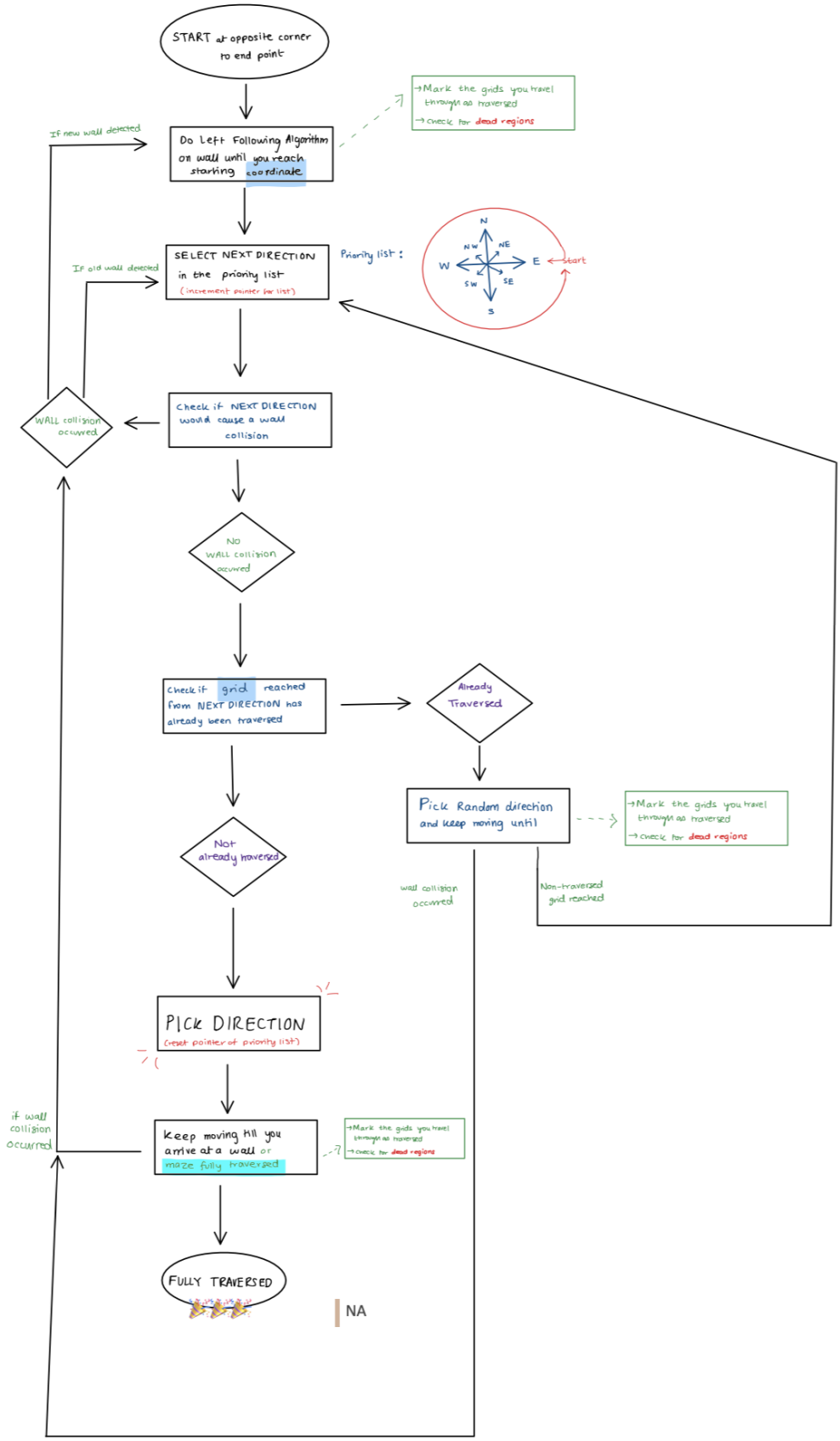


Figure 2.16: Algorithm flow diagram

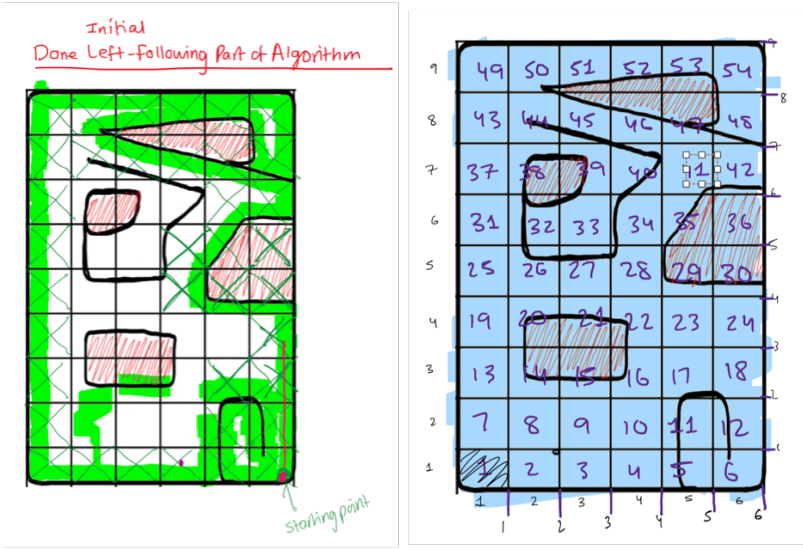


Figure 2.17: Left wall following and grid diagram

because the grids were strategically chosen to be large enough to encompass a majority of islands and loops, it is unlikely for multiple grids neighbouring grids to be in a dead region in islands and loops. Hence cells that have all neighbouring cells traversed are also marked as traversed on the map.

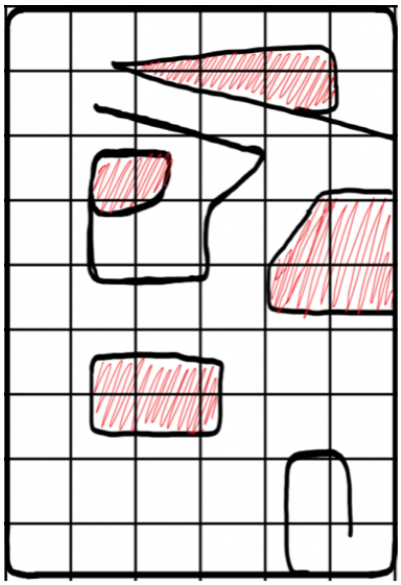


Figure 2.18: Dead regions marked in red

Dead Reckoning

Dead reckoning was achieved through a combination of wheel counting and gyroscope angular velocity measurements. Since stepper motors were used, the ESP32 has to directly send a signal

to cause them to turn, meaning it is able to count the total number of step-signals it has sent. A single step is 1/200 of a full revolution, and given the diameter of the wheel, the distance travelled per step, d_s, can be calculated. The bearing of the rover, θ, is obtained by integrating the measured angular velocity, and is relative to the y-axis. Equations 2.1 and 2.2 show how the step distance and rover bearing can be used to calculate the change in 2-dimensional coordinates.

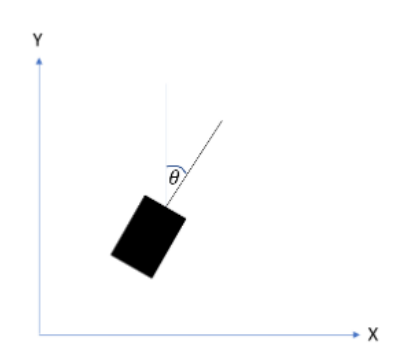


Figure 2.19: Coordinate system

$$\Delta x = \sin(\theta) \cdot d_s \tag{2.1}$$

$$\Delta y = \cos(\theta) \cdot d_s \tag{2.2}$$

The coordinates produced from dead reckoning were then transmitted to the client application and plotted on the screen. This allowed testing of the system which revealed errors in the maze plotting. It was found that the coordinates tended to drift as time went on because error in the bearing measurement of the gyroscope accumulated over time. Furthermore, when the rover collided with the walls, the motors continued moving in the direction of travel so the rover would think it is moving forwards and update its coordinates, even though it was stationary. Regions with sharp turns like the single LED strip were prone to the most wall collisions and hence had the least accurate representation of the traversal.

Shortest Path

Since the traversal of the rover was represented as coordinate points, the shortest path has to be found with only this information. Hence standard shortest path algorithms like Dijkstra's or A* search cannot be implemented in their true form as that would require a graphical representation

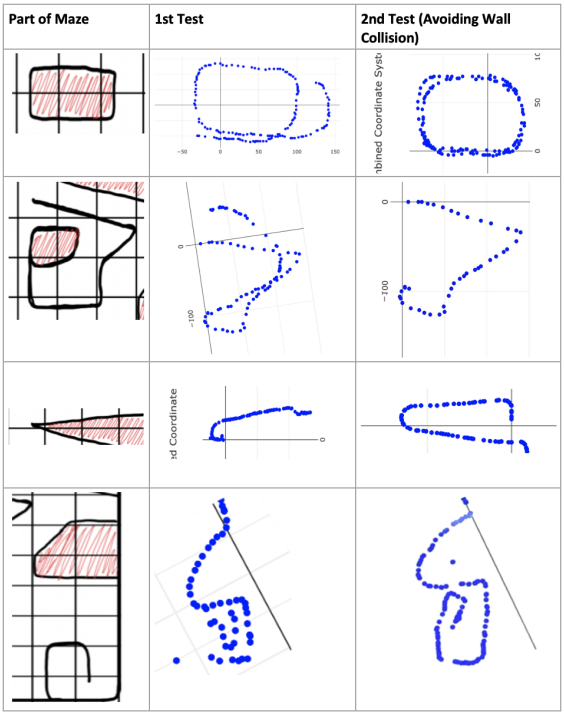


Figure 2.20: Results of plotting tests

of the information. Hence, an algorithm was devised, which draws curves to mark the possible paths of traversal in lines. It does this by connecting coordinates in close proximity. Once the dots are outlined with lines, the image is processed using a MATLAB script which uses a flood-fill algorithm to find the shortest path. The script requires the user to specify the start and end points on the image.

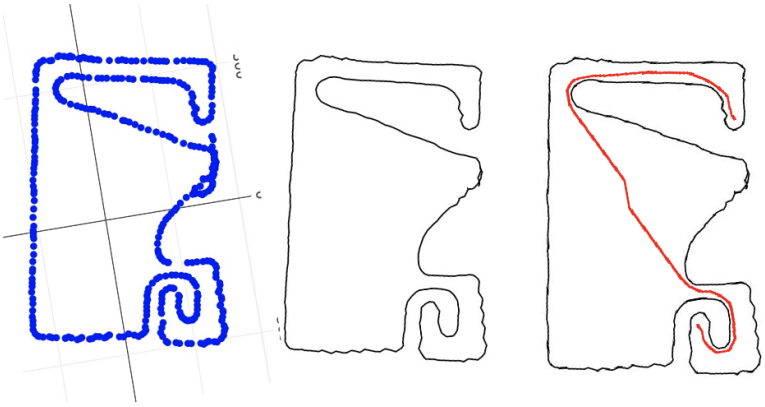


Figure 2.21: Plot of coordinates replaced with line and then flood-fill algorithm applied to image

2

2.3 Energy System

The role of the energy system is to manage and distribute the incoming energy from three 1.15W photovoltaic (PV) panels [3] such that three 1W coloured beacons are always powered in times of high and low sunlight. To help us achieve this a 0.25F 18V supercapacitor is used to store any excess energy not used to power the beacons when incident light is high, and output energy when light is low, thus maintaining a steady flow of energy to the beacons. Due to the I-V characteristics of the PV panels, an algorithm must be implemented in the SMPS interfacing the panels to the grid [4], to maximise the energy extracted. Each beacon will also have their own buck SMPS driver to safely manage the voltage and current delivered to each beacon. The role of the beacons is to recalibrate the onboard sensor, specifically the MPU6050, which suffers from drift and thus increasing inaccuracy as the maze is traversed. It also supports dead-reckoning by recalibrating the wheel counting if the rover is stuck on a wall. All the SMPS utilised have their own microcontrollers which are powered by a voltage regulator that requires a minimum grid voltage of 7V. Therefore, the three Buck SMPS drivers also play a vital role in maintaining a minimum grid voltage by controlling exactly when to allow power to flow to the beacons. Figure 2.22 shows an overview of the various devices involved and their interconnections.

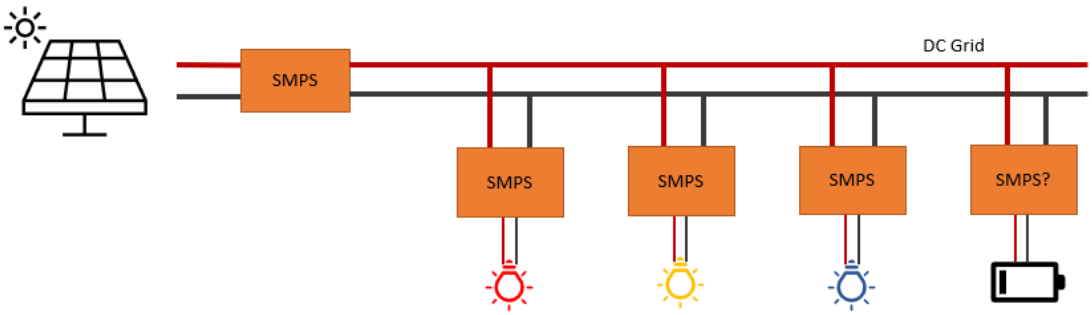


Figure 2.22: Overview of energy system

2.3.1 PV Panel characterisation and Emulation

To understand the performance of the PV panels they must first be characterised. The output of the PV panel was connected to the input of the SMPS. When connected to the SMPS, duty cycle can be varied to draw different currents from the panel and voltage can be measured. From this, an I-V plot can be produced. To acquire the different sections of the I-V curve, the load

resistance was changed to alter the maximum amount of current that can be drawn. To simplify the collection of measurements, a decision was made to use the SMPS in boost, non-synchronous mode. The microcontroller has a built-in sensor to measure the current drawn from the solar panel at the input side of the SMPS. The microcontroller is also able to measure the voltage of the PV panel by measuring the input voltage. As the duty cycle was varied, the inductor current and input voltage were monitored.

An issue encountered when characterising the panels outside was variations in sunlight intensity between readings. This posed an issue as the PV panels have different I-V characteristics for different light intensities. A shift to a different I-V curve makes characterising the panel for a specific intensity much more difficult. A solution to this was to add code to the SMPS microcontroller which sweeps the duty cycle and takes measurements of the output current and voltage automatically. This allowed values to be recorded before the light intensity had the chance to change. Values for the open circuit voltage and short circuit current were extrapolated from the graph.

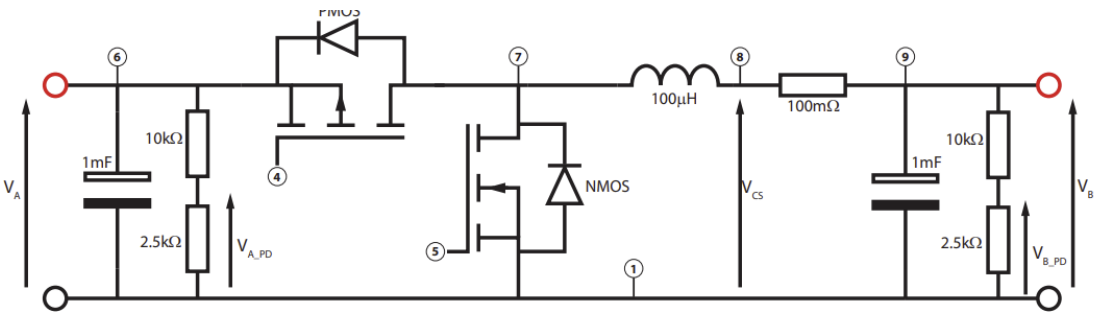


Figure 2.23: Merged buck and boost SMPS circuit

To simplify testing of the energy system in the lab and also to emulate the setup used in the demo, a circuit with the same I-V characteristics of the panels must be created using the bench PSU to act as a constant current source, and also two resistors to represent the shunt and series resistance of the panel.

LTSpice was then used to simulate the equivalent model and the resistor values were tuned so that the characteristic curves of the emulator were as close as possible to the one of the panel. The final values are shown in figure 2.25. The ideal diode was used to form a constant current source. Figure 2.26 is a plot of the experimental and simulated IV characteristic of the cell.

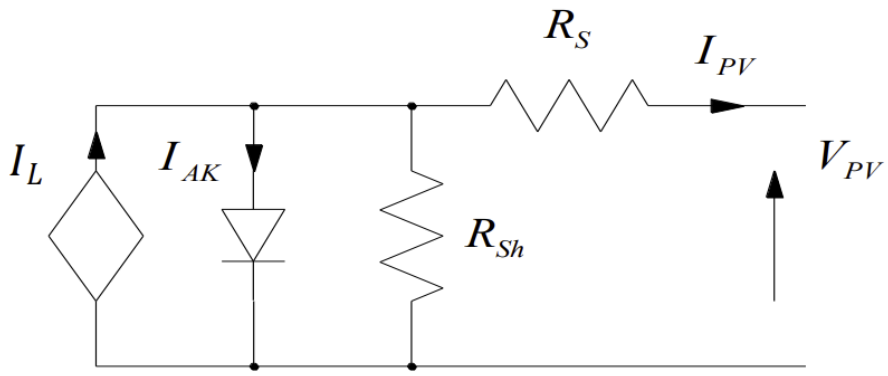


Figure 2.24: PV cell equivalent circuit

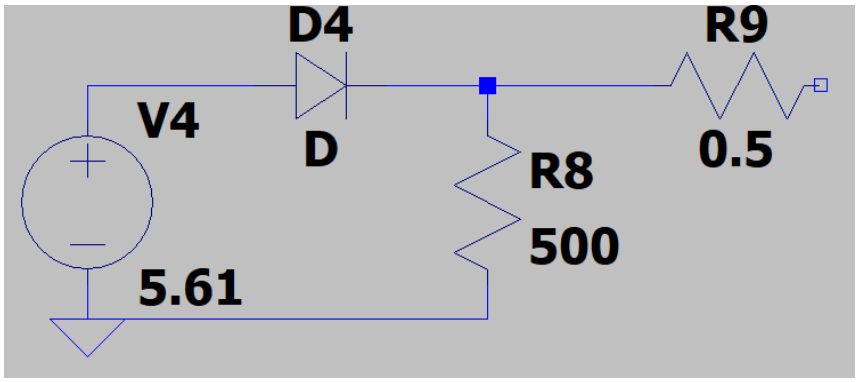


Figure 2.25: PV cell emulator circuit

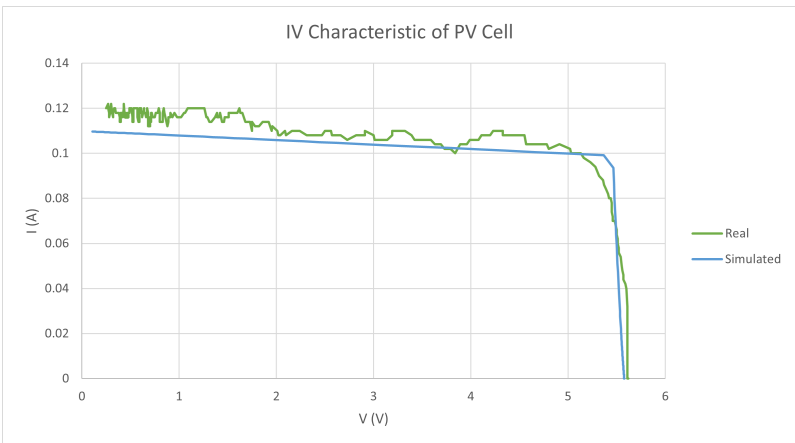


Figure 2.26: I-V characteristics of a simulated PV cell and Emulator

2.3.2 PV Power Output Optimisation

If a plot of the product of the PV Panel's voltage and current was produced, a point will exist where power is maximised. It was found that the power output is maximised when the voltage of the PV panel reaches approximately 5.6V under the tested light intensity. This is shown in figure 2.27.

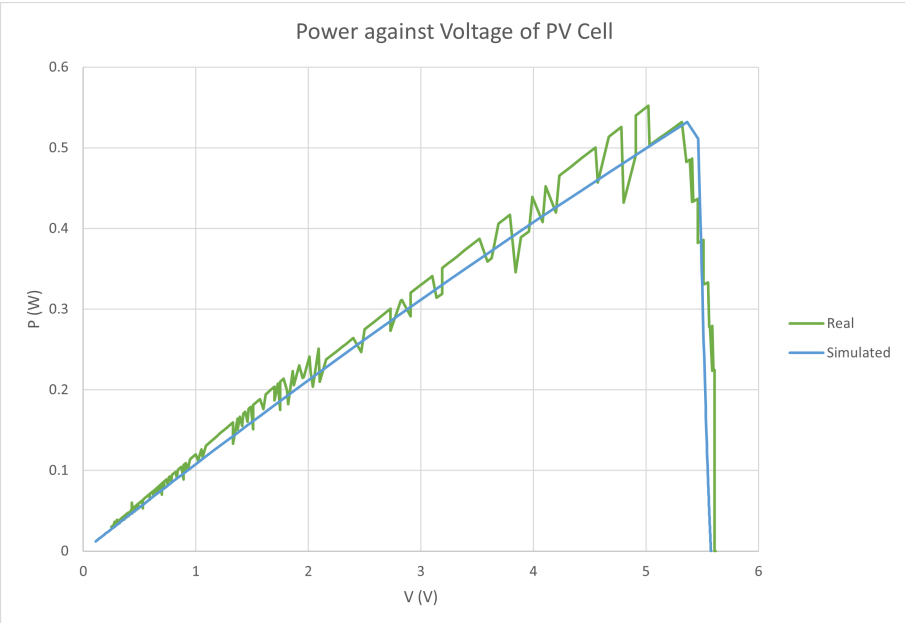


Figure 2.27: Power graph of real and emulated PV cell

The aim is to locate the point at which power output is maximised and stay at this point under any light intensity. Using the current and voltage sensors of the SMPS, an algorithm could be implemented to perform tracking of this maximum point. This is commonly known as Maximum Power Point Tracking or MPPT. The method implemented with the SMPS is a variation of the standard 'Perturb and Observe (P&O)' algorithm. The flowchart is shown in figure 2.28.

The microcontroller can calculate the current power drawn from the PV cells as the product of the input current and input voltage (V_b). It makes a small adjustment to the duty cycle in a certain direction, causing the voltage to change and therefore the current being drawn to change. It then re-samples the sensors and works out the new power being drawn. If this is greater than the previous time step, the algorithm will carry on increasing the duty cycle in the same direction, under the assumption that it is travelling towards the maximum power point. If at any point the power is less than the previous sample, it changes direction, and the duty cycle begins changing in the opposite direction. This is because it assumes it has now passed the maximum power point,

2

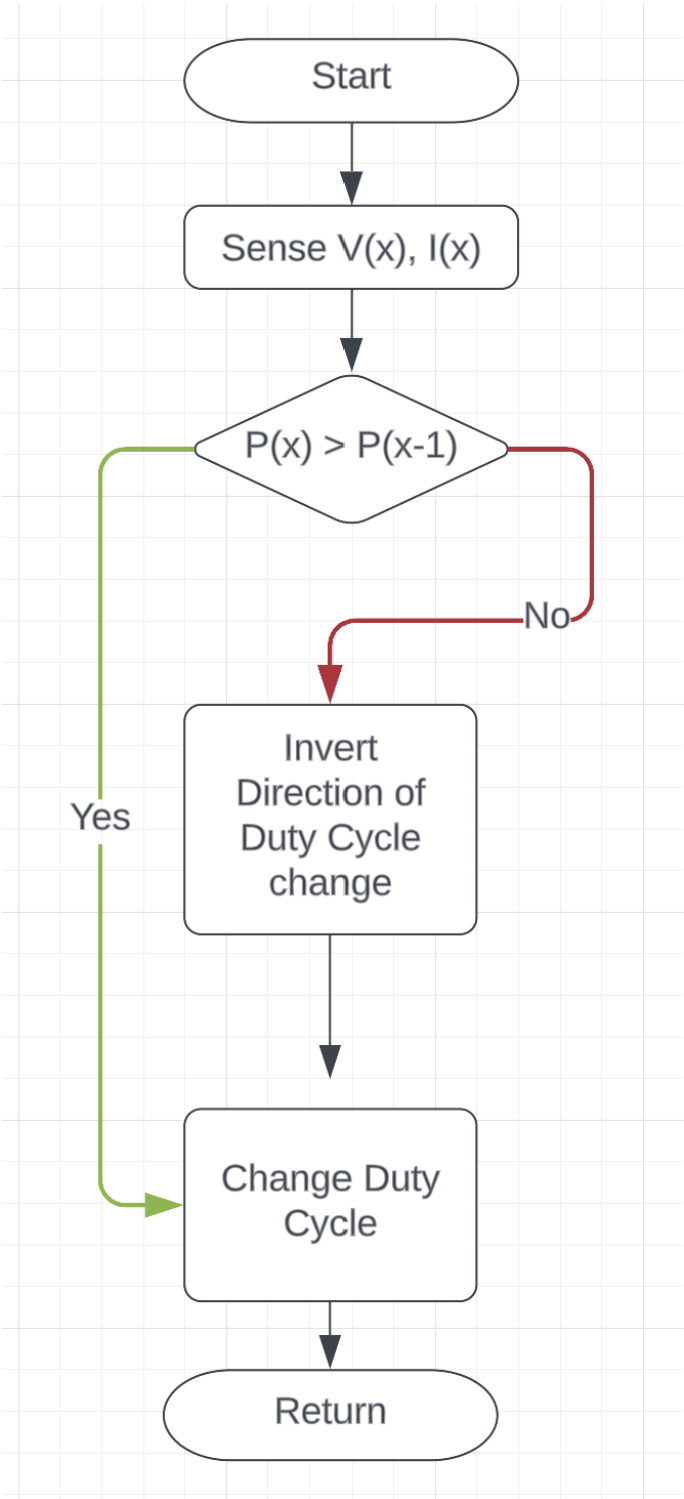


Figure 2.28: P&O algorithm

and now reverses to get back to it. The algorithm causes the power output to move towards the maximum point, and then oscillate around, constantly checking for changes.

2.3.3 Testing MPPT Performance

An issue was encountered when testing the original implementation as the MPPT algorithm would be extremely slow at finding the MPP (Maximum Power Point). This was due to the sensing being performed too frequently and perturbation of the duty cycle was also being fast, making it more susceptible to noise. Noise causes the measured power to randomly go up and down over a brief period of time, even if there is an overall trend in one direction. If sampled too fast, noise could lead the algorithm to believe that voltage and power is decreasing and thus reduce the duty cycle. To mitigate this, the code only takes a sample and perturbs the output once every 200 cycles. This means that while noise will still be present, given enough time, the voltage will change enough between each sample to observe the overall direction that the power is changing in.

```
16:14:30.140 -> Va: 0.50        Vb: 0.00        Inductor Current: 0.00  Power: 0.00     Duty: 0.55
16:14:30.653 -> Va: 0.50        Vb: 0.00        Inductor Current: -0.00 Power: 0.00     Duty: 0.50
16:14:31.123 -> Va: 0.86        Vb: 0.82        Inductor Current: 0.33  Power: 0.27     Duty: 0.49
16:14:31.642 -> Va: 1.30        Vb: 0.98        Inductor Current: 0.33  Power: 0.32     Duty: 0.44
16:14:32.111 -> Va: 1.68        Vb: 1.04        Inductor Current: 0.33  Power: 0.34     Duty: 0.39
16:14:32.624 -> Va: 2.04        Vb: 1.22        Inductor Current: 0.33  Power: 0.40     Duty: 0.42
16:14:33.137 -> Va: 2.44        Vb: 1.54        Inductor Current: 0.33  Power: 0.51     Duty: 0.47
16:14:33.644 -> Va: 2.88        Vb: 1.92        Inductor Current: 0.33  Power: 0.63     Duty: 0.52
16:14:34.116 -> Va: 3.36        Vb: 2.36        Inductor Current: 0.33  Power: 0.77     Duty: 0.57
16:14:34.628 -> Va: 3.88        Vb: 2.88        Inductor Current: 0.33  Power: 0.94     Duty: 0.62
16:14:35.144 -> Va: 4.42        Vb: 3.46        Inductor Current: 0.32  Power: 1.12     Duty: 0.67
16:14:35.654 -> Va: 4.98        Vb: 4.12        Inductor Current: 0.00  Power: 0.00     Duty: 0.70
16:14:36.125 -> Va: 5.51        Vb: 4.18        Inductor Current: 0.32  Power: 1.35     Duty: 0.65
16:14:36.639 -> Va: 6.01        Vb: 4.56        Inductor Current: 0.32  Power: 1.46     Duty: 0.68
16:14:37.154 -> Va: 6.53        Vb: 5.29        Inductor Current: 0.32  Power: 1.69     Duty: 0.73
16:14:37.624 -> Va: 6.93        Vb: 5.47        Inductor Current: 0.21  Power: 1.14     Duty: 0.70
16:14:38.138 -> Va: 7.37        Vb: 5.41        Inductor Current: 0.29  Power: 1.57     Duty: 0.65
16:14:38.651 -> Va: 7.71        Vb: 5.55        Inductor Current: 0.18  Power: 1.00     Duty: 0.66
16:14:39.163 -> Va: 7.95        Vb: 5.45        Inductor Current: 0.24  Power: 1.29     Duty: 0.61
16:14:39.628 -> Va: 8.35        Vb: 5.41        Inductor Current: 0.28  Power: 1.50     Duty: 0.58
16:14:40.143 -> Va: 8.69        Vb: 5.35        Inductor Current: 0.32  Power: 1.71     Duty: 0.55
16:14:40.657 -> Va: 9.05        Vb: 5.29        Inductor Current: 0.32  Power: 1.69     Duty: 0.54
16:14:41.126 -> Va: 9.33        Vb: 5.31        Inductor Current: 0.32  Power: 1.70     Duty: 0.51
16:14:41.642 -> Va: 9.63        Vb: 5.39        Inductor Current: 0.32  Power: 1.73     Duty: 0.50
16:14:42.156 -> Va: 9.89        Vb: 5.39        Inductor Current: 0.31  Power: 1.65     Duty: 0.49
16:14:42.669 -> Va: 10.15       Vb: 5.41        Inductor Current: 0.30  Power: 1.61     Duty: 0.48
16:14:43.137 -> Va: 10.41       Vb: 5.23        Inductor Current: 0.32  Power: 1.67     Duty: 0.47
16:14:43.646 -> Va: 10.67       Vb: 5.23        Inductor Current: 0.00  Power: 0.00     Duty: 0.46
16:14:44.160 -> Va: 10.89       Vb: 5.23        Inductor Current: 0.32  Power: 1.67     Duty: 0.45
16:14:44.632 -> Va: 11.11       Vb: 5.41        Inductor Current: 0.29  Power: 1.56     Duty: 0.44
16:14:45.146 -> Va: 11.33       Vb: 5.41        Inductor Current: 0.29  Power: 1.57     Duty: 0.43
16:14:45.658 -> Va: 11.55       Vb: 5.39        Inductor Current: 0.32  Power: 1.72     Duty: 0.42
16:14:46.171 -> Va: 11.75       Vb: 5.37        Inductor Current: 0.32  Power: 1.72     Duty: 0.43
16:14:46.640 -> Va: 11.95       Vb: 5.31        Inductor Current: 0.32  Power: 1.70     Duty: 0.42
```

Figure 2.29: P&O Algorithm Tracking Result

Figure 2.29 shows the measurements acquired by the SMPS microcontroller. The emulated PV

circuit was used which has the benefit of allowing the maximum current and voltage to be set, and thus control the maximum power that is outputted. To emulate the real PV panel, the PSU was configured with a current limit of 330mA at 5.6V, producing a theoretical maximum power output of 1.848W. It can be seen that the power of the PV panel is maintained at 1.70W, showing effective tracking by the algorithm with an error of only approximately ±30mW. The discrepancy between the set input power of the PSU and actual maximum power sensed by the SMPS is due to the shunt and series resistors used in the emulators which both consume power. This also shows that the emulator is accurate.

2.3.4 DC grid Management

The decision was made to connect the 0.25F supercapacitor directly to the output of the Boost SMPS, in other words, the DC grid. The full energy system is shown in figure 2.30. The benefit of this method is that a secondary SMPS between the DC grid and supercapacitor is not required, increasing efficiency and also simplifying the overall energy system design. Though disadvantages include the fact that the capacitor must reach the LED drivers' minimum input voltage before turning on and also any remaining energy below this minimum voltage cannot be used. However, due to the low power consumption of the beacons and high output current from the parallel PV Panels, the energy system is able to rapidly charge above the LED drivers' threshold voltage and maintain a higher voltage for an adequate amount of time.

The Boost SMPS is configured to increase the DC grid voltage to as close to the maximum rated voltage of the supercapacitor as possible since the energy stored in a supercapacitor is proportional to the square of the voltage. However, it is critical that the capacitor is not charged past its rated voltage as this could lead to damage and breakdown of the dielectric material, and in the worst case, an explosion. Therefore, within the code for the SMPS, the duty cycle was decreased as soon as the voltage at the output reaches 17.9V. A 100mV margin of error was included in case of overshoot, which could occur and cause damage if the threshold was set to 18V [5]. A snippet of the code used, and resulting capacitor voltage is shown in figure 2.31

Ideally, the Boost SMPS should be in synchronous mode in order to maximise efficiency, but the current design of the SMPS only allows a maximum output voltage of 9V in this mode. This is due to the use of a second synchronous MOSFET at the output side. When the output is greater than 9V, V_{gs} of the PMOS will always be lower than the threshold and therefore will always be saturated. This means that when the NMOS is active there is a direct path to ground which

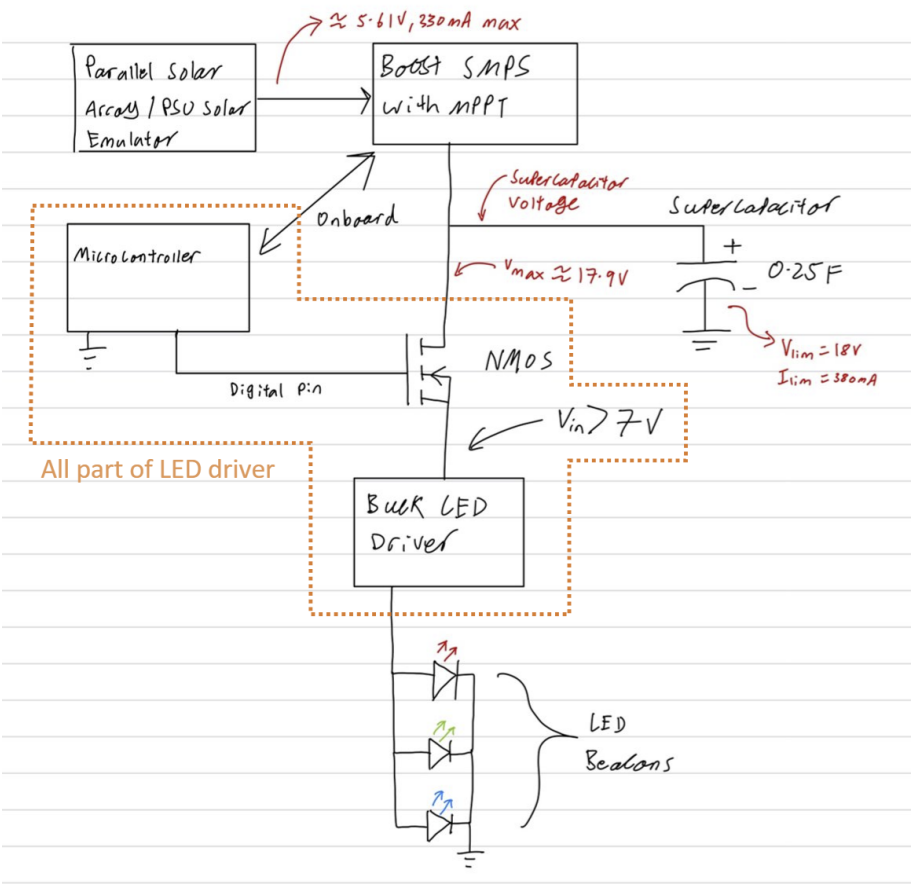


Figure 2.30: Energy system

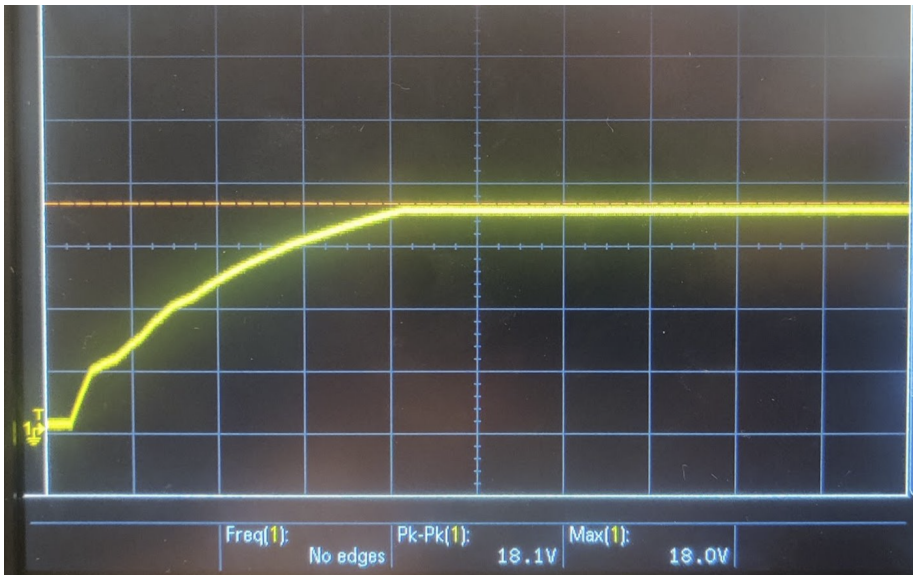


Figure 2.31: DC grid voltage management

reduces the voltage of the capacitor.

2

2.3.5 LED Drivers

The LED drivers are SMPS circuits configured in buck mode. Their purpose is to provide a constant flow of power to the LED beacons, even under changing conditions on the DC grid. They are controlled by an onboard RaspberryPi Pico W. Built-in resistors allow the Pico to measure voltages and currents at key points across the circuit. Pin 27 allows the input voltage to be measured, while pin 28 allows the output voltage to be obtained. These pins are connected via potential dividers to scale down the voltage to a range the Pico can read. Pin 26 measures the voltage across a small resistance, which can be used to calculate the current flowing at the output. All of these measurements are useful for controlling the behaviour of the device.

Initially, the LED beacons were powered by a PSU and the voltage was varied to observe the change in brightness. The camera and computer vision software were then used to determine the minimum brightness that could be detected. This information would later be utilized to ensure the beacons did not consume more power than was necessary. The results of this experiment are summarized in table 2.5.

Beacon Colour	Voltage (V)	Current (mA)
Yellow	1.9	26
Blue	2.6	28
Red	1.9	31

Table 2.5: Power consumption of coloured beacons

The Pico was programmed to control the MOSFET driver through a PMW output pin. By varying the duty cycle of this PWM signal, the output voltage could be controlled approximately using equation 2.3, where δ is the duty cycle of the MOSFET control signal.

$$V_o = \delta V_i \tag{2.3}$$

The SMPS has to hold the output voltage constant at the voltages previously found. Since the input voltage (connected to the DC grid) would vary with time, an algorithm to control the PWM duty cycle had to be implemented.

One initial solution is to assume ideal conditions and use 2.3, along with a measurement of the input voltage, to work out the required duty cycle. However, there are many uncertainties that could result in unexpected values. If the output voltage is too low, the beacons may not be bright enough and the calibration algorithm would not be able to work. If the voltage went too high, the beacons would begin to consume too much power, and could potentially be damaged. An alternative solution, that could take into account changing uncertainties was required. A PID controller was chosen as it is able to hold the output at a defined setpoint whilst also responding to changes in conditions. Equation 2.4 shows how the control output is determined. $e(t)$ is the error, the difference between the current output and the intended output. By changing the gains, the behaviour of the controller could be modified to obtain ideal performance. The output of this PID equation is the required change in duty cycle to track the intended output voltage.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \tag{2.4}$$

This controller had to be implemented digitally on the Pico. Instead of an integral, a sum of previous errors was kept. Each error was multiplied by T_s, the difference in time between successive executions of the main loop, as an approximation of dt. Instead of differentiation, the difference between the current and the last error was found, and this was divided by T_s. This gives an approximation of the gradient at the current point in time. The discrete version implemented is shown in equation 2.5

$$u[t] = K_p e[t] + K_i \sum_{\tau=0}^{t} (e[\tau] T_s) + K_d \frac{e[t] - e[t-1]}{T_s} \tag{2.5}$$

The gains were tuned until the controller gave satisfactory performance in terms of response speed and damping of oscillations. The final values are shown in table 2.6.

Gain	Value
Proportional, K_p	300
Integral, K_i	5
Derivative, K_d	10

Table 2.6: Gains of SMPS PID controller

Extra precautions to ensure the safety of the LEDs were implemented. If the output current ever rose above 250mA [6], the Pico would restrict the duty cycle to help bring it back down. If the

output voltage ever rose above the stated LED limit, the PWM signal would switch off, essentially disconnecting the LED from the DC grid. This prevents damage from voltage spikes on the DC grid. If the input voltage were to unexpectedly and rapidly increase, the PWM duty cycle would take time to change, but in the meantime the output voltage would still rise to match equation 2.3. Although extreme spikes of this nature are not expected, this feature was simple to implement with no real disadvantage.

The buck SMPS can only operate when the input voltage is greater than 7V, so the duty cycle was programmed to drop to 0% when this occurred, turning the LED beacons off. As the beacons consumed power, the voltage on the DC grid would drop. When it reached 7V, the beacons would turn off and the DC grid voltage would begin to rise again. This would cause the beacons to turn back on almost immediately again. To prevent the voltage oscillating around 7V, and the beacons rapidly flashing, hysteresis was implemented. The beacons would turn off when the input voltage dropped to 7V, but would only turn on again when it rose above 8V.

2.3.6 Optimisation

To preserve power, when the beacons were not running, the Pico goes into light sleep mode [7]. This causes its power consumption to drop from 0.16 W to around 0.065 W. The Pico wakes from light sleep every second to check the input voltage to see if it has risen above 8V. If not, it goes back to sleep for another second. This is particularly useful when the PV cells have been outputting little power for extended periods of time. Since the Pico is powered from the circuit's input voltage (connected to the DC grid), if the DC grid ever drops below 5V, the Pico will shut off. Hence, when the voltage drops below 7V and the beacons turn off, this sleep mode allows the system to remain functioning for longer.

One final optimisation choice was to flash the beacon LEDs. They would turn on for one second, then off for one second, and repeat this cycle. This meant in the long run they would only consume around half the power as if they were constantly on. The main drawback is that the camera would potentially have to wait for the beacons to turn on, however the relatively short time of this cycle meant it would only have to wait at most one second.

2.3.7 LED Driver Testing

```
V_in: 8.153119,  V_out: 2.599077,  I_out: 0.05371176,  Duty: 0.3200654,  pwm_ref: 20975.49,  Vmax: 2.647573, E: 0.0009231567
V_in: 8.157161,  V_out: 2.599077,  I_out: 0.04660285,  Duty: 0.3196336,  pwm_ref: 20947.19,  Vmax: 2.647573, E: 0.0009231567
V_in: 8.165243,  V_out: 2.586952,  I_out: 0.05292188,  Duty: 0.3197199,  pwm_ref: 20952.84,  Vmax: 2.651614, E: 0.01304746
V_in: 8.161201,  V_out: 2.603118,  I_out: 0.05292188,  Duty: 0.319917,   pwm_ref: 20965.76,  Vmax: 2.651614, E: -0.0031178
V_in: 10.50189,  V_out: 2.607159,  I_out: 0.05450164,  Duty: 0.2521443,  pwm_ref: 16524.28,  Vmax: 2.675862, E: -0.007159233
V_in: 12.75745,  V_out: 2.607159,  I_out: 0.05529151,  Duty: 0.2084786,  pwm_ref: 13662.64,  Vmax: 2.683944, E: -0.007159233
V_in: 14.7382,   V_out: 2.595036,  I_out: 0.05134212,  Duty: 0.1806862,  pwm_ref: 11841.27,  Vmax: 2.70011, E: 0.004964352
V_in: 11.31419,  V_out: 2.57887,   I_out: 0.04186357,  Duty: 0.2265417,  pwm_ref: 14846.41,  Vmax: 2.70011, E: 0.02112961
V_in: 11.9853,   V_out: 2.611201,  I_out: 0.04897248,  Duty: 0.2235944,  pwm_ref: 14653.26,  Vmax: 2.70011, E: -0.0112009
V_in: 13.67104,  V_out: 2.599077,  I_out: 0.05371176,  Duty: 0.1944324,  pwm_ref: 12742.13,  Vmax: 2.70011, E: 0.0009231567
V_in: 13.7155,   V_out: 2.595036,  I_out: 0.04897248,  Duty: 0.1929839,  pwm_ref: 12647.2,   Vmax: 2.70011, E: 0.004964352
V_in: 12.64025,  V_out: 2.607159,  I_out: 0.04739273,  Duty: 0.2074821,  pwm_ref: 13597.34,  Vmax: 2.70011, E: -0.007159233
V_in: 10.61504,  V_out: 2.582911,  I_out: 0.04897248,  Duty: 0.2443651,  pwm_ref: 16014.47,  Vmax: 2.70011, E: 0.01708841
V_in: 8.626206,  V_out: 2.590994,  I_out: 0.04976236,  Duty: 0.3018737,  pwm_ref: 19783.29,  Vmax: 2.70011, E: 0.009006023
V_in: 7.219321,  V_out: 2.611201,  I_out: 0.05134212,  Duty: 0.3620289,  pwm_ref: 23725.56,  Vmax: 2.70011, E: -0.0112009
V_in: 9.75803,   V_out: 2.603118,  I_out: 0.05845103,  Duty: 0.2741106,  pwm_ref: 17963.84,  Vmax: 2.70011, E: -0.0031178
V_in: 12.06613,  V_out: 2.603118,  I_out: 0.05134212,  Duty: 0.2185116,  pwm_ref: 14320.16,  Vmax: 2.70011, E: -0.0031178
V_in: 13.41214,  V_out: 2.595036,  I_out: 0.05134212,  Duty: 0.1966019,  pwm_ref: 12884.31,  Vmax: 2.70011, E: 0.004964352
V_in: 14.11559,  V_out: 2.590994,  I_out: 0.04976236,  Duty: 0.1881791,  pwm_ref: 12332.32,  Vmax: 2.70011, E: 0.009006023
V_in: 15.17871,  V_out: 2.57887,   I_out: 0.05292188,  Duty: 0.1760934,  pwm_ref: 11540.28,  Vmax: 2.70011, E: 0.02112961
V_in: 16.12463,  V_out: 2.599077,  I_out: 0.05292188,  Duty: 0.16669,  pwm_ref: 10924.03,  Vmax: 2.704151, E: 0.0009231567
V_in: 16.41157,  V_out: 2.590994,  I_out: 0.04739273,  Duty: 0.1630218,  pwm_ref: 10683.63,  Vmax: 2.704151, E: 0.009006023
V_in: 13.12117,  V_out: 2.562705,  I_out: 0.04581297,  Duty: 0.1987581,  pwm_ref: 13025.61,  Vmax: 2.704151, E: 0.0372951
V_in: 8.517092,  V_out: 2.586952,  I_out: 0.05055224,  Duty: 0.3074396,  pwm_ref: 20148.06,  Vmax: 2.704151, E: 0.01304746
V_in: 8.521132,  V_out: 2.586952,  I_out: 0.04581297,  Duty: 0.3072858,  pwm_ref: 20137.97,  Vmax: 2.704151, E: 0.01304746
```

Figure 2.32: LED driver output measurements

Figure 2.32 shows various measurements made by the Pico during testing. The input voltage was varied between 8V and 16V to observe how well the control system held the output at the target voltage of 2.6V. The performance of the design can be evaluated on a few different metrics. When the input was held at 7.99V for an extended period of time, the output hovered around 2.607V, so the system has good static precision. There is also minimal oscillation in the output, which varies by less than 100mV across the testing range. The design has a fast response speed, being able to track the output setpoint even when the input voltage was rapidly changed.

2.3.8 WiFi Considerations

The RaspberryPi Pico W microcontroller has WiFi capabilities. In theory, it is possible to communicate between the three LED drivers, the PV boost SMPS, and the ESP32 on the rover itself. One potential use of this would be to allow the rover to control when the beacons turn on. By default, the beacons would be off. When the rover wanted to recalibrate, it would send a message to the drivers, telling them to turn the LEDs on. Once calibration was completed, it would send another message to cause the LEDs to turn off. The key advantage of this system is that the beacons only consume power when required. The current system will turn the beacons on, even if the rover doesn't currently require them, thus wasting power. However, the major disadvantage of this design is the increased power consumption of the Pico caused by the networking components. This is further exaggerated by the fact that the Pico is unable to enter sleep mode whilst WiFi is enabled, meaning it always draws this increased power. It is possible to disable WiFi to enter sleep mode, and then reconnect upon wakeup, however this was determined to be too slow at responding

to requests.

State	Power Consumption (W)
Idle	0.16
WiFi Enabled	0.21
Making GET Request	0.25

Table 2.7: Power draw of Pico in various states

2.3.9 Comparison

Experimentation shows the combined power consumption of all three LED beacons is around 109 mW. If they were to flash on only half of the time, this drops to approximately 54 mW. If instead the Pico is WiFi enabled and the rover is able to dictate when the beacons turn on, an initial estimate of their average power consumption is 11 mW. This assumes the beacons are on for 10% of the time as an upper bound.

Using these estimates, along with the measured current draw of the Pico, approximations of the average power consumption of the system can be calculated. Table 2.8 shows these approximations for the various design choices considered.

Design	Average Power (mW)
Constantly on	221
Flashing	166.8
WiFi controlled	220.9

Table 2.8: Estimated power consumption of various designs

The most power efficient design was the flashing version. Although the beacons consumed less power overall with the WiFi enabled design, the increased power usage of the Pico, and the fact it was unable to enter sleep mode, completely counteracted this, being only marginally better than just leaving the beacons constantly on. Furthermore, the increased complexity of the WiFi system would make it less reliable as it creates more points of failure. To further reduce the power consumption of the flashing design, the on-off ratio could be changed so that the beacons are off, and the Pico in sleep mode, for a greater period of time.

2.4 Balancing

2.4.1 Controller Design and Implementation

Balancing the robot on 2 wheels required a controller in a feedback control scheme. The controller was designed in continuous time, as shown in figure 2.33, then implemented digitally on the ESP32. Said controller follows the very common PID controller design, its coefficients are tuned experimentally until the system satisfies qualitative requirements.

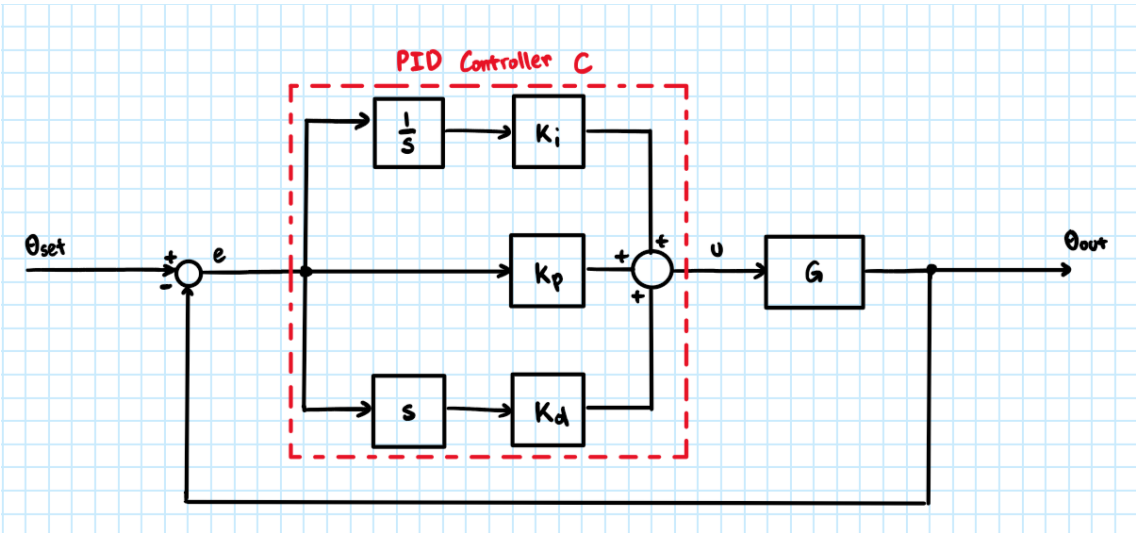


Figure 2.33: Continuous-time representation of control system

The control system consisted of 3 main modules: pitch sensors, ESP32 controller software and stepper motor actuators. Design choices taken in each are covered and justified below.

2.4.2 Sensor Fusion

The MPU6050 is an inertial measurement unit which includes a gyroscope and an accelerometer. It is used to determine the orientation of the rover along two different axes. The first use was to keep track of the direction the rover was facing, a value needed to implement dead-reckoning. The second use was to determine the pitch of the chassis, necessary for the balancing control system.

The MPU6050's gyroscope measures angular velocity $\dot{\theta}$. By integrating $\dot{\theta}$ over time, the current pitch can be obtained. Since a digital micro-controller was used, a sum was used to approximate

integration.

$$\theta_1[t] = \theta[t-1] + \dot{\theta}[t]T_s \tag{2.6}$$

The accelerometer, on the other hand, measures linear acceleration a. This sensor can detect acceleration due to gravity, so orientation of the sensor can be inferred from the acceleration components in each axis. Pitch, the desired angle, is a function of $a_x[t]$ and $a_y[t]$, as shown in equation 2.7

$$\theta_2[t] = \arctan\left(\frac{a_x[t]}{a_z[t]}\right) \tag{2.7}$$

The gyroscope's pitch readings suffer from integration error which accumulates and generates drift, a form of low-frequency noise. The accelerometer's pitch is influenced by high-frequency noise from the rover experiencing accelerations not caused by gravity. Since noise from the accelerometer and gyroscope lies in different frequency bands, a complementary filter [8] was chosen to fuse both readings into a more accurate estimation of pitch. The complementary filter equation 2.8 is a weighted sum of the two measurement methods, which filters the noisy bands of each sensor's readings.

$$\theta[t] = \alpha\theta_1[t] + (1-\alpha)\theta_2[t] \tag{2.8}$$

Complementary filtering is a less computationally expensive alternative to Kalman filtering, which was appropriate given the limited computational power of the ESP32. As discussed below, it was very important to operate control processes at high frequencies, so reducing load on the ESP32 was crucial.

The Arduino library 'MPU6050_light' was employed due to its fast read access to the MPU6050 and automatic complementary filtering of both sensor data when the function $MPU6050.getAngleY()$ is called [9].

2.4.3 Microstepping

When testing the stepper motors, operating in full stepping mode generated a considerable amount of vibration. This prevented the rover from maintaining equilibrium for more than a few seconds.

Increasing the stepping resolution allowed smaller adjustments to be made when near equilibrium. The motor drivers can operate in various "microstepping" modes [10]; $\frac{1}{16}$ stepping was chosen to maximise precision.

2.4.4 Multiprocessing

The STEP and DIR signals sent to the A4988 stepper driver has to be periodically updated for the motors to keep rotating. The stepper signal was originally set by a line of code in the ESP32's main loop routine. However, having other processes, such as reading from the MPU6050, in that loop reduces the maximum frequency at which steps can be taken, hence limiting maximum angular velocity of motors.

A solution is to process the controller processes and the motor processes in parallel. The variable control_speed exchanges information between them. The ESP32 library 'FreeRTOS' allows the user to assign tasks to each core with a chosen priority; *loop*() is priority 0 by default [11]. A separate task was created for sensor reading and controller calculations; it was designated to core 1 and given lower priority. Running the stepper motor process in the main loop without other processes significantly increased the maximum stepping frequency, allowing the system to respond faster to disturbances and support higher motor speeds.

2.4.5 Cascaded Control

Once the rover was able to balance, making the rover move required an additional outer acceleration loop that varies the pitch setpoint. The MPU6050 is able to provide raw acceleration data, hence, a fairly accurate measure of acceleration could be used to calculate acceleration error. The same PID controller was suitable for acceleration control, but new coefficients had to be found through tuning.

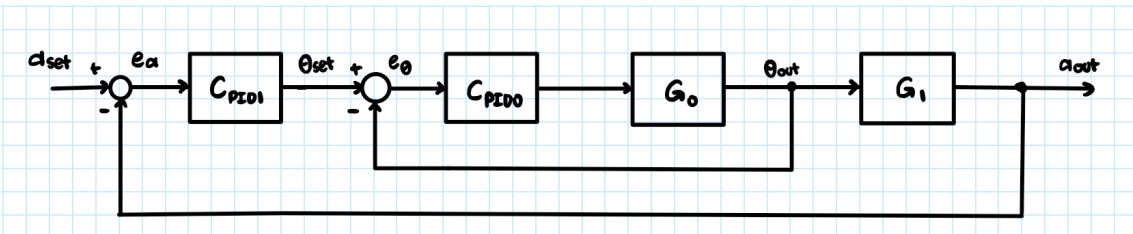


Figure 2.34: Continuous-time representation of cascaded control system

By applying a set acceleration for a short period of time, a set displacement was achieved and the rover was able to follow a routine of moving forwards, stabilising, moving backwards, stabilising, and so on.

2.4.6 Tuning Parameters

Through extensive testing, the outer and inner loops were tuned until the rover was able to hold its pitch at 0 in an idle state, making it stationary, and then move with a given, gentle acceleration. The tuned coefficients are shown in table 2.9.

Controller	K_p	K_i	K_d
Pitch	1000	50	400
Acceleration	3	0.1	0.5

Table 2.9: Cascaded control parameters

Refer to the appendix for videos of the balancing and moving rover. An LED was attached to the rover during debugging, which flashes blue when the rover is attempting to move. This acted as a great method of checking whether the rover was following its intended behaviour.

2.5 Chassis Design

The current traversal strategy relies on the camera being low to the ground, hanging off the front of the chassis. Furthermore, the transistors that aid with wall detection need to be mounted close to the LED strips for optimal performance and sensitivity. A mount was required that would be able to securely attach to the chassis, and hold all of these peripherals in place.

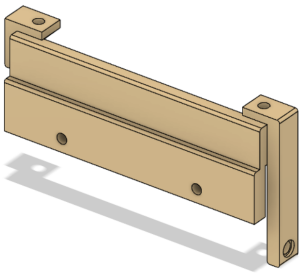


Figure 2.35: Camera and front phototransistor mount

Figure 2.35 shows the initial design for this mount. The holes at the top allow it to attach to the screws underneath the FPGA on the chassis. The holes on the front line up with the screws on the camera PCB. The hole on the side is for securing the phototransistor. There is a larger indent around the hole to prevent the phototransistor from tilting, increasing reliability and robustness. The design in figure 2.36 is a proposed model for another phototransistor holder. This one is intended to hold a sensor facing downwards, right above the leading edge of the wheel. This will be useful for wall avoidance as it will only detect a wall if the wheel is right in front of one, and about to run into it.

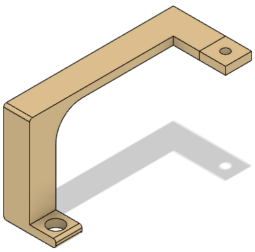


Figure 2.36: Wheel phototransistor mount

Designs 2.35 and 2.36 were 3D printed from PLA, chosen for its low price and ease of use. Since the parts were not expected to be under much mechanical stress, the physical properties of PLA

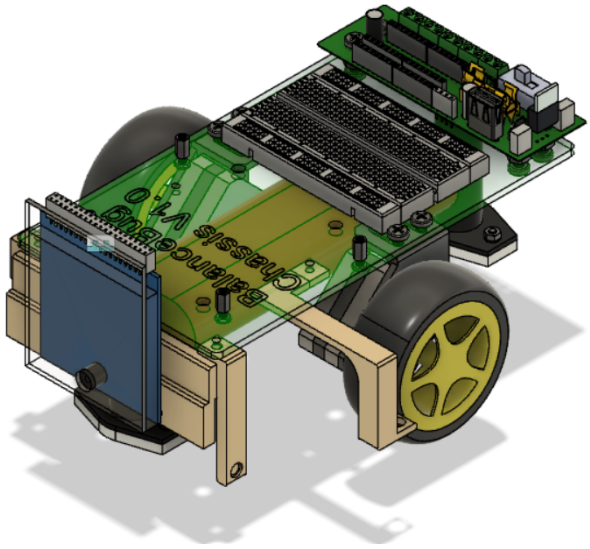


Figure 2.37: Final assembly (FPGA omitted)

were sufficient to ensure its durability.

The CAD design in figure 2.37 shows the intended mounting points and overall assembly of the rover design (FPGA omitted for clarity).

After manufacturing and integrating all components, it was discovered that optimal camera performance requires it to be mounted further down. To solve this problem, a second version of the mount was designed, shown in figure 2.38. For improved flexibility, the holes were replaced with slots that allow the camera height to be adjusted, making it easy to account for any possible future changes.

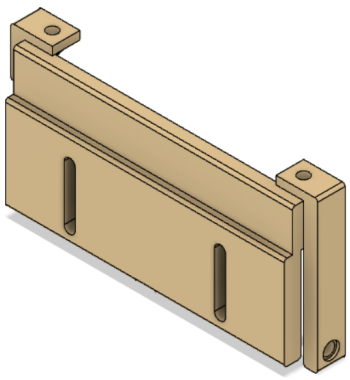


Figure 2.38: New camera mount

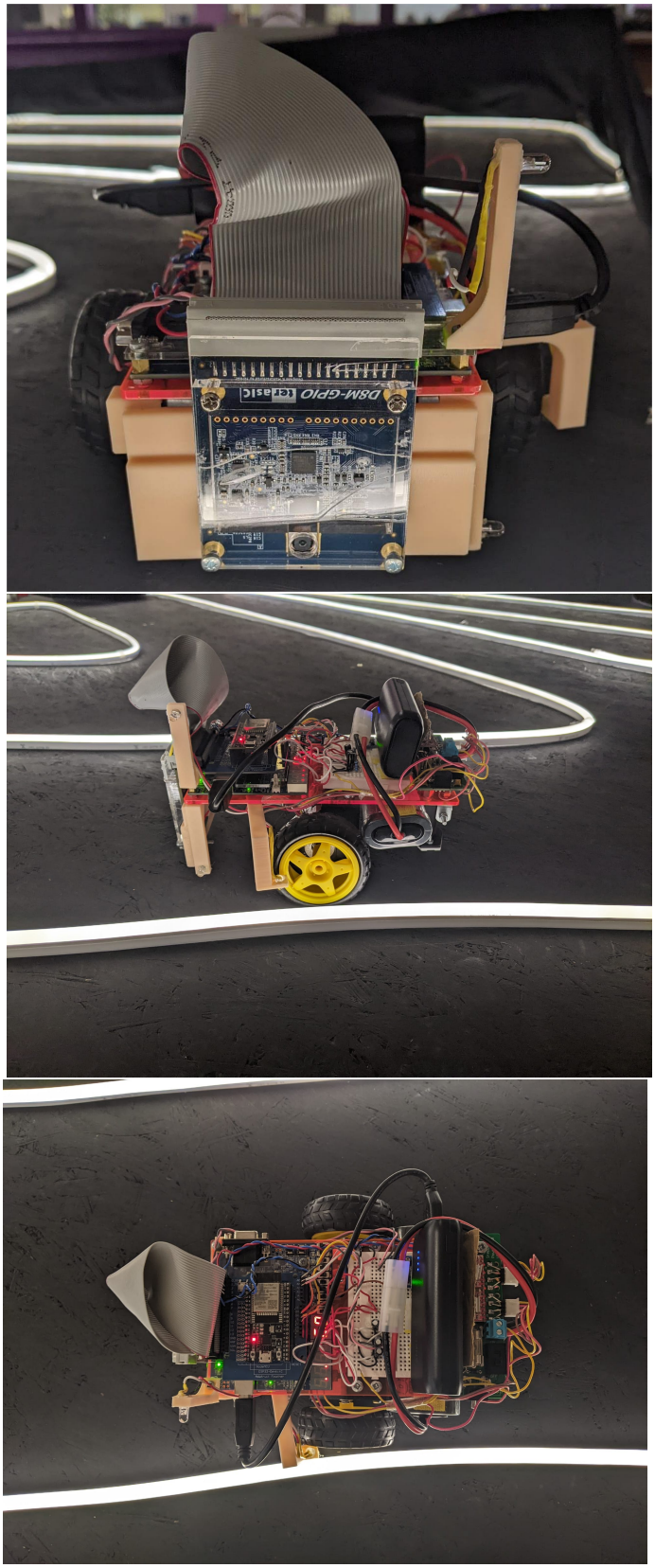


Figure 2.39: Final chassis

3 Conclusion

Contents

3.1 Conclusion 49
3.2 Improvements 50

3.1 Conclusion

In summary, a rover was produced to meet the requirements outlined in the engineering specification. The rover is able to balance on two wheels, traverse and survey the layout, produce a map of the layout, and deduce the shortest path through the maze. A PV energy system was also designed to provide beacons, used to aid navigation, with a stable and safe supply of energy despite external variations in sunlight and solar energy.

This project thoroughly tested the professional engineering skills of the team. While the team applied an overwhelming amount of hard, analytical skills, such as in the design and implementation of many parts of the rover, the project also required soft, non-technical skills such as teamwork, time management, communication, creativity and many more, which were just as important in ensuring that the project ran smoothly.

3.2 Improvements

Although all aspects of the challenge were individually completed, it was not possible to integrate all of them in the final product. The final rover used in the maze was able to carry out the navigation and communication tasks, but was unable to balance on two wheels. This was due to the design choice of using phototransistors to left-wall follow, which required minimal oscillation of the chassis. With more knowledge and research, the camera could have been used to detect all walls, eliminating the need for the phototransistors and the mount. This would then open up the possibility of balancing at the same time as navigation.

A Appendix

- OneDrive: `https://imperiallondon-my.sharepoint.com/:f:/g/personal/an1321_ic_ac_uk/EuBcsdgh49JJslkrH0U0P2QBfnjWzg-c_vV9qcBQHEcSFQ?e=bcVZNn`
  - Media: Image and video demonstrations
    * Balancing Tests: Videos of the control system in action. The rover is able to balance on the spot and is resilient to disturbances and external forces. One of the videos also showcases the cascaded controller's ability to move the rover forwards whilst simultaneously balancing.
    * Beacon Tests: Video of the rover finding its position using the beacons. It performs a full 360-degree rotation to record their positions and calculates its own position within the maze. This coordinate is then displayed on the screen.
    * Traversal Tests: Demonstrations of the rover successfully mapping out regions of the maze. Each region has a video showing the traversal as well as a screenshot of the React.js client of the received path. The path is then overlayed with an image of the actual maze to evaluate the accuracy.
    * Project_Management_App.mov: Video showcasing Notion, the project management tool used throughout. Shows the various organisation and planning utilities that aided teamwork and productivity.
- Github Code: `https://github.com/anish-narain/EEEBalanceBug`
- OneNote (diagrams and notes): `https://imperiallondon-my.sharepoint.com/:o:/g/personal/an1321_ic_ac_uk/EhrBGrcFxH9NvqBRPmurfSQBAyNmrbrH4nIzCSOGT0eCHg?e=pXySUF`

A

Bibliography

[1] L. Szary, *Algorithm to convert rgb to hsv and hsv to rgb in range 0-255 for both*, Stack Overflow, Accessed 2 June 2023, Jan. 1957. [Online]. Available: https://stackoverflow.com/questions/3018313/algorithm-to-convert-rgb-to-hsv-and-hsv-to-rgb-in-range-0-255-for-both.

[2] MATLAB, *How to Segment Images Using Color Thresholding Video*, Video, Accessed 2 June 2023, 2023. [Online]. Available: https://uk.mathworks.com/videos/how-to-segment-images-using-color-thresholding-1572503834930.html.

[3] TruOpto, *Opl50a23101 datasheet*, https://static.rapidonline.com/pdf/502676_v1.pdf, Accessed: 2023-06-19.

[4] T. Green, *Photovoltaic power*, Topics in EE: Solar Energy, [Lecture] Imperial College London, 19th October, 2021.

[5] Cornell Dubilier - CDE, *Dsm254q018w075pb datasheet*, https://www.mouser.co.uk/datasheet/2/88/DSM-3135689.pdf, Accessed: 2023-06-19.

[6] Switch Electronics, *1w / 3w / 10w high power heatsink pcb led*, https://www.ebay.co.uk/itm/281769990142?var=580772787255, Accessed: 2023-06-19.

[7] D. P. George, *Functions related to the hardware*, https://docs.micropython.org/en/latest/library/machine.html, Accessed: 2023-06-06.

[8] Instructables, *Arduino self-balancing robot*, https://www.instructables.com/Arduino-Self-Balancing-Robot-1/, Accessed: 2023-05-25.

[9] R. J. Fetick, *Mpu6050lightarduinolibrarydocumentation*, https://github.com/rfetick/MPU6050_light, Accessed: 2023-06-19.

[10] Allegro MicroSystems, *A4988 motor driver datasheet*, https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf, Accessed: 2023-06-19.

[11] E. Sytems, *Freertos api documentation*, https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html, Accessed: 2023-06-19.