Engineering Design Project 2021
Mars Rover Report

Alden Sentil Kumaran : 01776884
Ritvik Shyam : 01701560
Jayjun Lee : 01727735
Wenlin Yi : 01541991
Yuze Feng : 01708226

Contents

Section 1

Introduction

The Mars Rover project aims to build and remotely operate an autonomous rover system consisting of multi-disciplinary subsystems implemented to navigate in a remote terrain of obstacles without direct human supervision once the obstacles are localised. The rover comprises of the vision, control, command, drive and energy subsystems, which are assembled by the integration subsystem, as shown in Figure 1.1.

The vision module processes the visual data through a camera and an FPGA to perceive the terrain for the localisation and mapping of the balls. The control module connects to other modules via several communication protocols and processes received data before relaying it to other subsystems. The command module consists of a web application which remotely controls the rover by sending commands to the control subsystem which in turn relays to the drive subsystem, whilst simultaneously receiving updates regarding the rover's status and dynamics alongside displaying a map of obstacles and the rover path – all of which are visualised graphically on the dashboard. The drive module oversees moving the rover based on the commands sent from the dashboard and the commands generated from the path finding algorithm once the mapping is complete. The integration module combines all the modules together and oversees testing inter-module connections under remote conditions.

In this report, the detailed implementation of each submodule and the approaches taken to problem solving is discussed followed by the integration and testing of the assembled rover as well as analysing its performance metrics and identifying possible improvements and extensions.

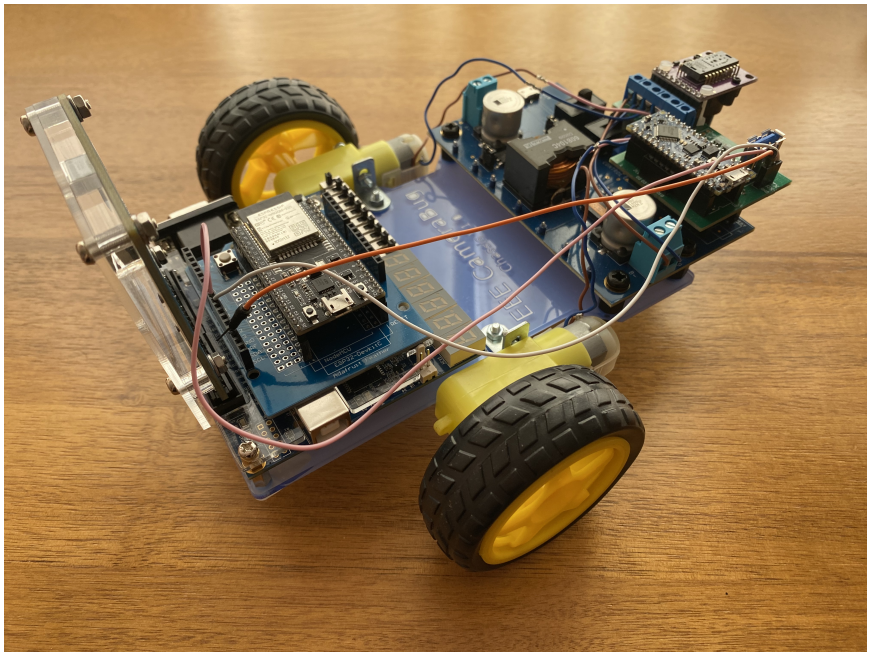


Figure 1.1: Image of the assembled Mars Rover.

Section 2

Project Management

In terms of project management, the major aspects considered are the meeting schedules, documentation of progress, version control and approaches to problem solving. Right from the commencement of the project, a definitive timetable for group meetings is maintained whereby the whole group meets every Monday, Wednesday and Friday between 10 am to 1 pm BST which is extended to 9am to 2pm BST every day for the last week. The purpose of a meeting is to generally update everyone on the status of their respective subsystems, perform intermodular and full system testing, as well as collaborate on documentation for the report and video. In situations requiring problem solving, debugging errors and general queries, Piazza and weekly one-to-one consultations with the concerned module leaders are utilised. Moreover, from the beginning of the project, a private GitHub repository is used to enable version control of the source code of the command, control, vision and drive subsystem. The documentation for project management consists of meeting minutes which are found on the private channel on Microsoft Teams and a Gantt chart to set goals and track milestones as shown below in Figure 2.1.

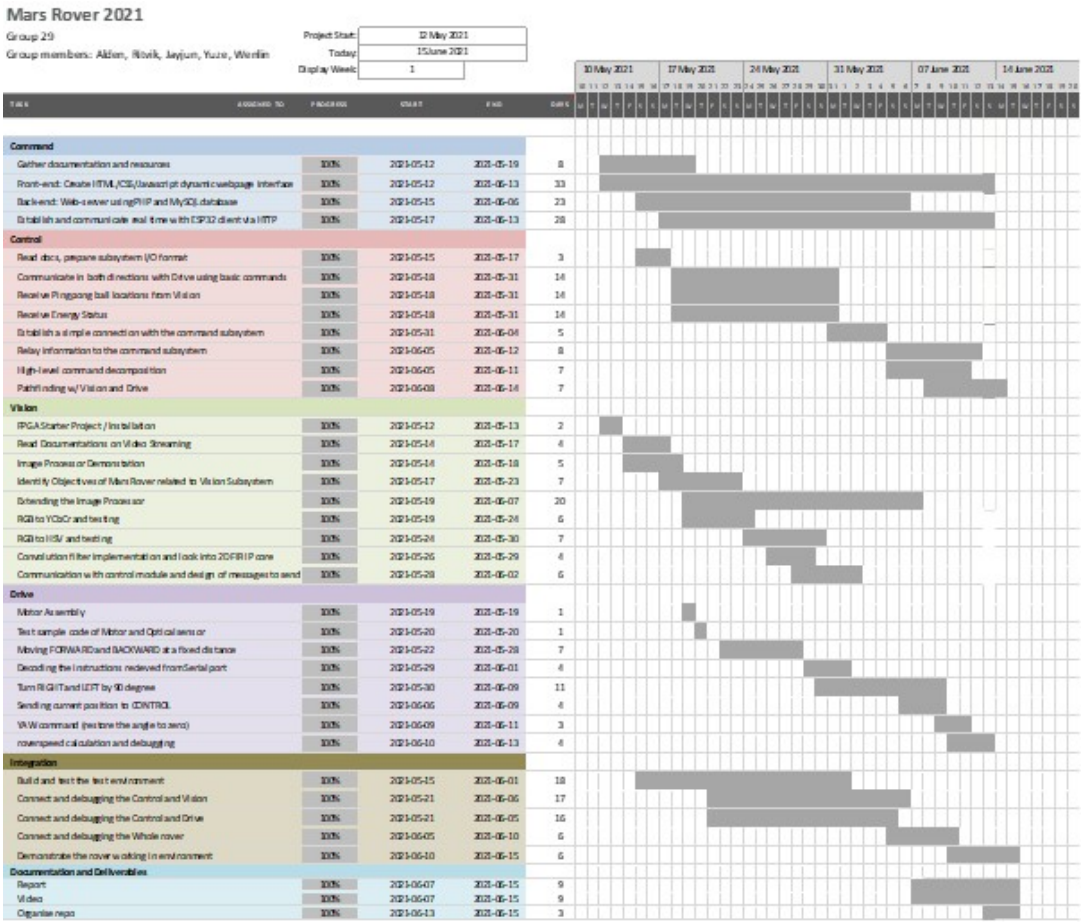


Figure 2.1: Gantt chart.

Section 3

Functional and Non-Functional Requirements

3.1 Functional Requirements

For the Mars Rover project, the following functional requirements are identified for each subsystem and the assembled rover by the integration subsystem. These requirements define the basic Mars Rover system behaviour.

- Drive
  - i Decode the command from control subsystem and extract the distance required to move.
  - ii Implement the forward, backward, turn right and turn left instructions.
  - iii Keep track of the rover's position for odometric localisation.
- Vision
  - i Detect and compute location of the obstacles of different colours around the rover.
  - ii Localise and map a terrain of obstacles.
- Control
  - i Communicate with other on-board subsystems using different communication protocols.
  - ii Communicate and relay information between the rover and the remote web server.
- Command
  - i Create a web dashboard to remotely control the rover by sending motion commands and fetch data from the associated database.
  - ii Receive updates on the rover's energy status in terms of battery life and drive status, including rover dynamics (position, speed and yaw) and obstacle map.
- Full System
  - i Integrate all sub modules to assemble the rover.
  - ii Build a working map of its surroundings as it moves across the environment.
  - iii Follow high-level commands issued by the command subsystem.
  - iv Navigate and avoid crashing into obstacles autonomously given the destination coordinates once the mapping is complete.

3.2 Non-Functional Requirements

Moreover, non-functional requirements are established as the following to describe the quality attributes and how the rover system should specifically behave overall and for each module along with some limitations on their functionality.

- Drive
  - i Compute an error to control the movement of the rover.
  - ii Slow down the motor speed to ensure the accuracy of the movement.
  - iii Record the yaw angle to determine the position change in x, y coordinate.
- Vision
  - i The image processor uses a convolution filter to remove noise that it is robust under noisy environments.
  - ii The image processor uses RGB to HSV colour space conversion to detect the candidate pixels of red, green, blue, yellow and pink balls.
  - iii The image processor draws bounding boxes reliably that fit the balls in the field of view.
  - iv The image processor sends the coordinates of the bounding boxes to control subsystem where it computes the distance and angle of deviation to the ball from the camera relative to the rover.
- Control
  - i The rover communicates with the command system web server using the HTTP protocol which is a robust and intuitive method of data transfer over the internet.
  - ii The rover communicates with all onboard modules using the UART protocol which is fastest method of serial data transfer.
- Command
  - i The front-end of the web application is designed using HTML, CSS and JavaScript, primarily using the jQuery library and Google Charts for the graphical displays.
  - ii The back-end web server is hosted on a free web hosting platform called 000WebHost which provides a NGINX web server that implements PHP scripts to connect to MySQL database managed by the phpMyAdmin tool.
  - iii The server and the database are secure from external malicious attacks using a securely stored private API key and login credentials.

Section 4

High-Level System Overview

The Mars Rover is overall a complex system with various components involved, which are structurally interconnected with respect to their objectives and functionalities. Therefore, the structural and functional models are provided in this section to simply represent the high-level model and to identify interactions and notable data flows within the life cycle of the system.

4.1 Structural Model

Figure 4.1 outlines the structural diagram of the Mars Rover system, specifying the parameters involved in each of the components utilised by each submodule.

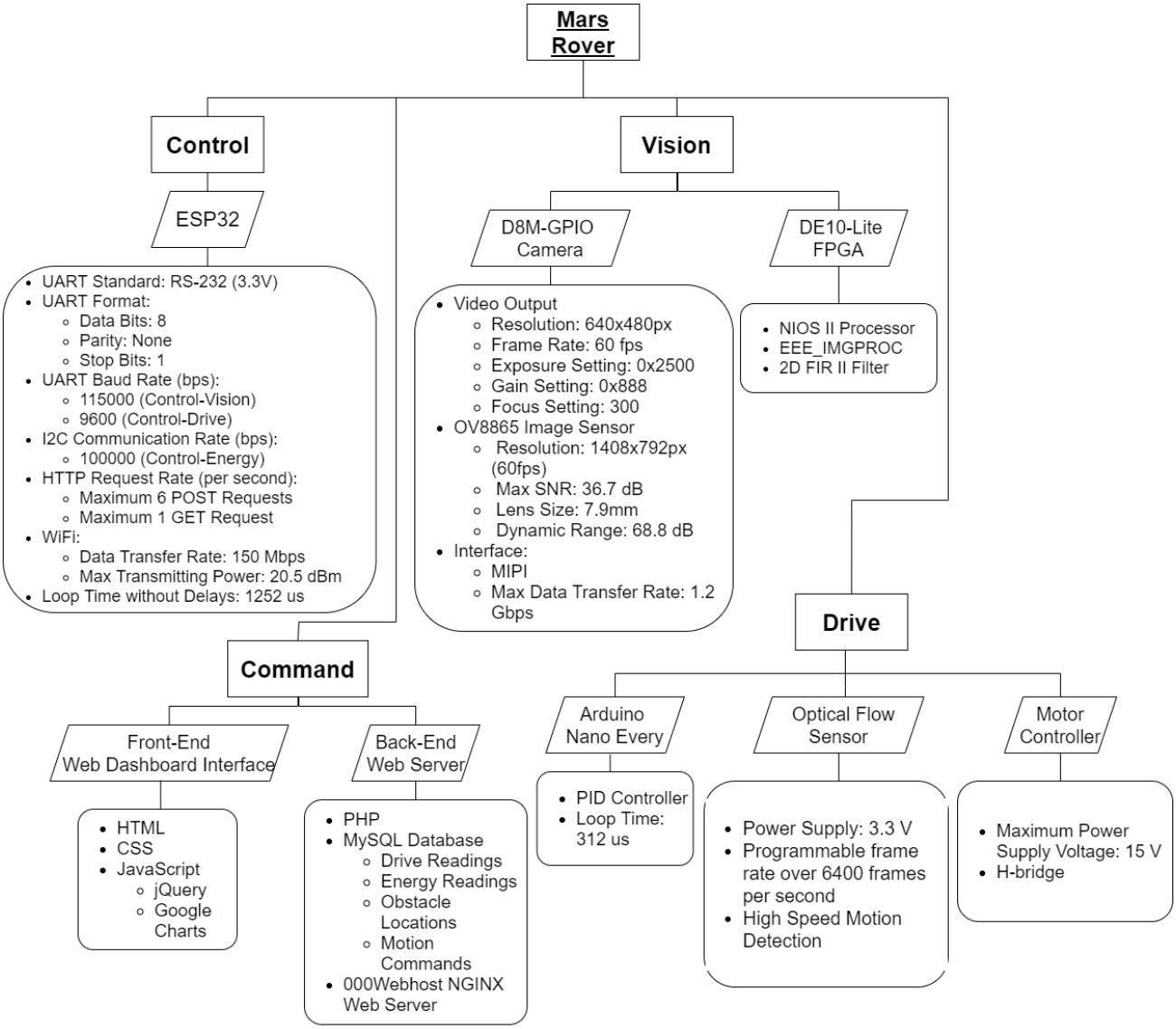


Figure 4.1: The structural diagram of the Mars Rover system.

Some key elements from the datasheets and specifications of the boards (DE10-Lite FPGA, ESP32, Arduino Nano Every), sensors (D8M-GPIO Camera, ADNS-3080 Optical mouse sensor) and actuators are extracted [1–3].

4.2 Functional Model

Figure 4.2 showcases the functional use case structure of the Mars Rover. The diagram outlines the data-flow and interactions between each subsystem and their associated components.

The modules' functionality can be briefly described as the following. The control system interconnects vision, drive, energy and command systems as well as doing relevant processing to the data that flows within the system. The vision module detects obstacles using the camera and sends information on the size of the bounding boxes to the obstacles to control for further processing. The drive module controls the motor drive according to commands from the control module and keeps track of the rover's position using the optical flow sensor to update its status. The energy module sends information regarding the rover's energy status and the expected life. The command module receives all these information through the control module to display the data in the right format and sends high level motion commands to control the rover.

The life cycle of the rover involves these modules operating simultaneously with the data sent across modules to function as a whole and complete its duty.

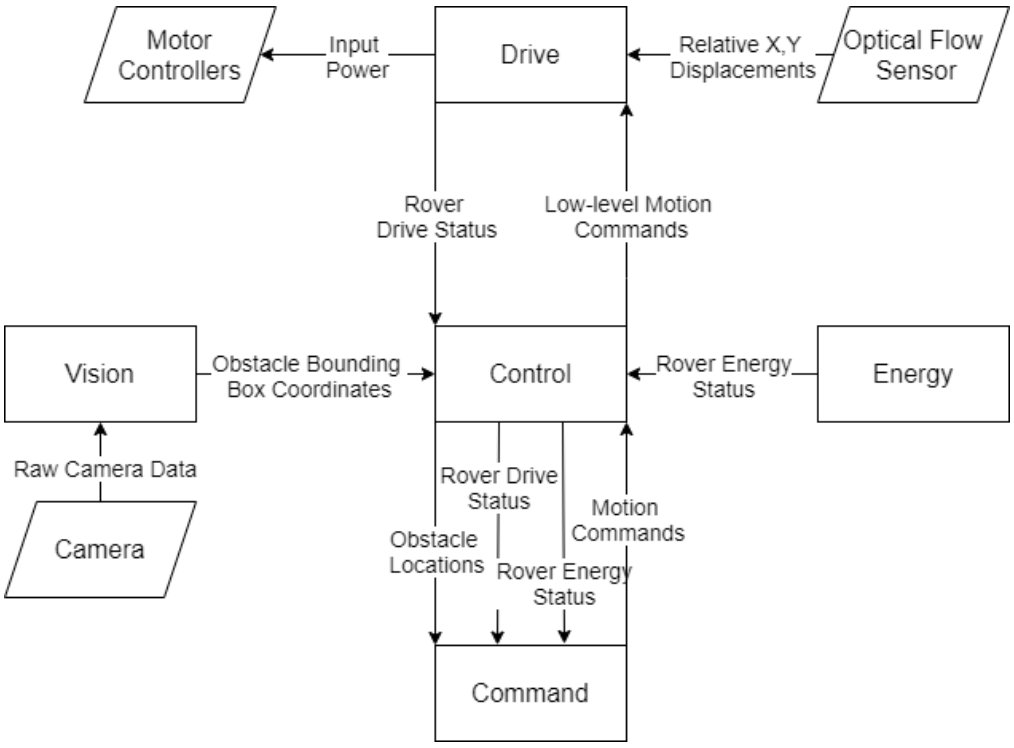


Figure 4.2: The functional diagram of the Mars Rover system.

Section 5

Subsystems Design and Implementation

5.1 Command

5.1.1 Overall Objectives and Functionalities

The primary objective of the command module is to implement a web application accessible via a user's web browser to connect to the control module to remotely command the rover's motion, track and display the rover's status as well as monitoring parameters from the energy and drive modules.

The web application is hosted on a free web hosting service called '000WebHost' which uses the NGINX web server to serve dynamic HTML contents on the network which is requested by the ESP32 client. In terms of functionality, the front-end provides a graphical interface to navigate the rover's motion with the following commands: reset, move forward by 10 cm, move backward by 10 cm, turn left by 90 degrees, turn right by 90 degrees, as well as an input field to autonomously move the rover to another coordinate in the x-y plane.

Moreover, the front-end also allows users to graphically visualise and monitor the rover's current position, a terrain map of obstacles with the rover's dynamic and energy status in real-time. The front-end of the web interface was designed using HTML, CSS and JavaScript (including jQuery library) and the back-end web was built using PHP scripts communicating with a remote MySQL database, managed by phpMyAdmin which is a free and open-source database administration tool.

5.1.2 Front-End Web Development

The front-end of the web application is designed using HTML and styled using CSS, with JavaScript to develop a dynamic webpage. The entire front-end source code is contained in the 'index.php' file which includes the HTML, CSS and JavaScript sections.

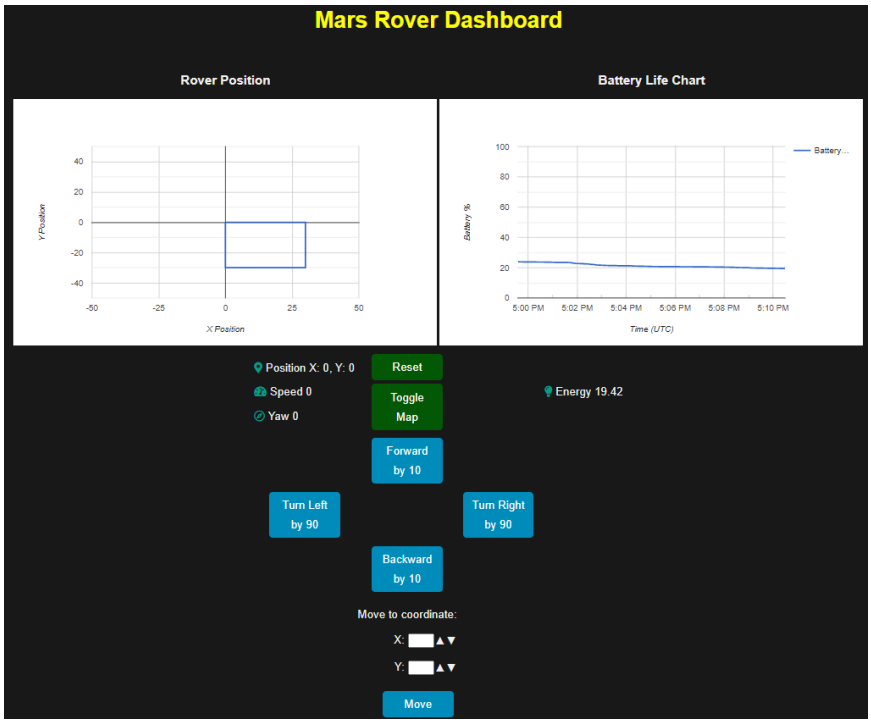
The main elements of the front-end webpage comprise of a header, title and charts to display the rover position and an obstacle map, an energy status chart and a command panel consisting of buttons alongside an input field as shown in Figure 5.1 to issue different types of motion commands to the rover, as described in Table 5.1. Figure 5.1(a) shows the rover's position and path, which can be toggled to the terrain map of obstacles in Figure 5.1(b).

Table 5.1: List of commands sent by the web dashboard.

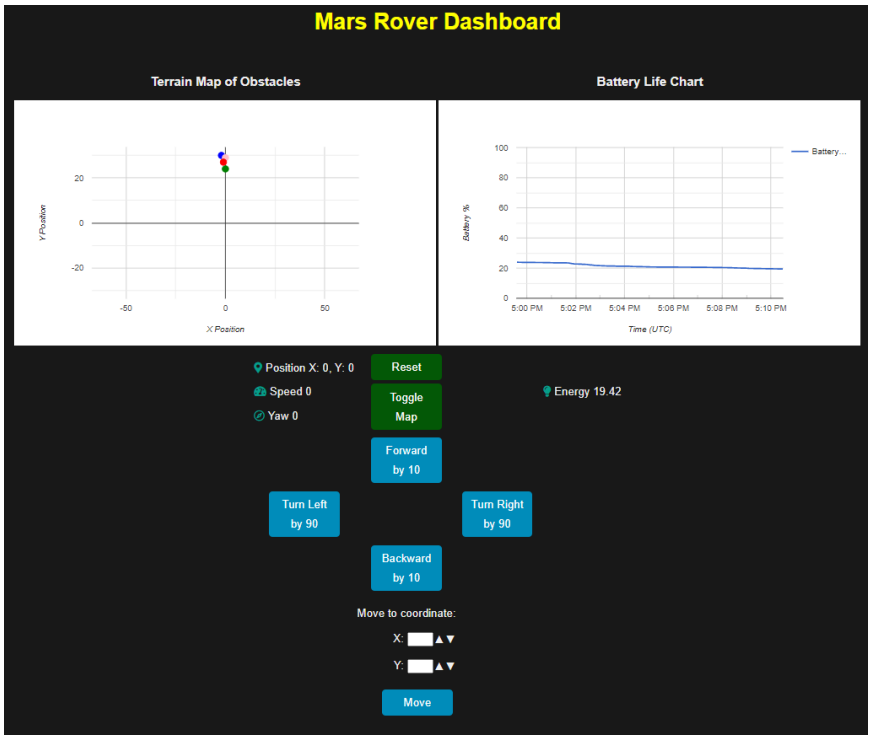
Command Name	Command Format	Description
Move Forward	"fw,10,0"	Move forward by 10 cm
Move Backward	"rv,10,0"	Move backward by 10 cm
Turn Left	"tl,90,0"	Turn left by 90 degrees
Turn Right	"tr,90,0"	Turn right by 90 degrees
Move to Coordinate	"mv,x,y"	Move to (x,y)

To generate the graphical display charts, the Google Charts JavaScript charting library is implemented as it is a free web service which supports all relevant chart types. JavaScript is used to program the behaviour of the dynamic front-end webpage using the jQuery library which is one of the most popular lightweight libraries providing simplified features for AJAX (Asynchronous JavaScript And XML) calls. The AJAX method allows for data exchange with the server and asynchronously updates the webpage

without the need to manually reload it. Furthermore, jQuery has plugins for almost all tasks and can run exactly the same in all major browsers [4].



(a)



(b)

Figure 5.1: Front-end web dashboard. (a) rover path toggled. (b) Terrain map of obstacles toggled.

5.1.3 Back-End Web Development

Initially the two options considered for back-end web development framework were node.js with mongo DB and PHP with MySQL. Having explored both options, PHP with MySQL was chosen because PHP is a thin layer language with fewer variables and certain elementary functions and it does not use any JAR

files and compilers, thus enabling a developer to create a web application with simply an editor and PHP files. Moreover, PHP facilitates its developers to mix codes with content by simply opening PHP tags and code without the need of templates or other files [5].

The back end of the web application is developed on a NGINX web server provided by the free hosting platform using PHP scripts to create, access and write to a MySQL database. Several PHP scripts are used to build the back-end and to interact or query the database to fit its purpose and functionality, which are illustrated in Table 5.2.

Table 5.2: PHP scripts for the back-end and their functionality.

PHP Script	Functionality
index.php	Handles main front-end user interface which uses: • esp32-get-energy-array.php to visualize the battery life chart using the google chart library. • esp32-get-position-array.php to visualize the rover path using google chart library. • esp32-get-ping-pong.php to visualize the terrain map of obstacles using google chart library. • esp-post-command to queue up commands into the remote database. • sql-query-latest.php to get the latest readings of position, speed, yaw, and battery life.
esp-log-PP.php	Invoked by the ESP32 client, on the Mars Rover, whenever an obstacle is sighted to log its position and colour in the 'ESP32PingPong' Database.
esp-log-data.php	Invoked by the ESP32 client, on the Mars Rover, every second to log its current position, speed, yaw, and battery life to the 'ESP32DriveReadings' and 'ESP32EnergyReadings' databases.
esp-post-command.php	Invoked by the front end which appends commands to the remote database to be fetched later by the Mars Rover. If the script is called with the name parameter being 'clear' it deletes all commands in the database.
esp32-get-energy-array.php	Invoked by the front end when visualizing the battery life chart. Queries the 50 latest energy readings.
esp32-get-ping-pong.php	Invoked by the front end when rendering the terrain map of obstacles. Queries the 5 latest readings of each obstacle and returns the average (x,y) coordinates of each obstacle grouped by colour.
esp32-get-position-array.php	Called by the front end when rendering the rover path. Returns all the received coordinates of the rover across its life cycle.
sql-query-latest.php	Used by both the Mars Rover and the front end. The front end uses the script to obtain the latest readings of position, speed, yaw, and battery life. The rover uses the script to extract the earliest command which has not been performed yet, after which the command is deleted from the database to avoid repetition.

The remote MySQL database schemas hosted in the back-end are displayed in Table 5.3. The commands queue schema holds the first-in first-out (FIFO) queue of commands that is stored until the command has been executed, which is then removed from the database. This is to prevent accidentally sending the same command several times due to variations in the response time from the rover as well as missing a command from a set of commands due to an error or loss in communication channels. The drive readings schema stores information relating to the dynamics of the rover, this includes the position, speed and yaw of the rover. The energy readings schema monitors the current battery percentage of the rover. The obstacle locations schema stores the information about the balls detected by the vision module. It takes in the computed location of the balls and the detected colour and the reading time, which can be utilised for processing and updating from data that could be possibly outdated.

Table 5.3: Database schemata hosted on the back-end.

Database	Database schema diagram
Commands Queue	Name: id — Type: int(6) — Collation: — Attributes: UNSIGNED — Null: No — Default: None — Comments: — Extra: AUTO_INCREMENT Name: command — Type: varchar(10) — Collation: utf8_unicode_ci — Attributes: — Null: No — Default: None — Comments: — Extra: Name: param1 — Type: int(4) — Collation: — Attributes: — Null: Yes — Default: NULL — Comments: — Extra: Name: param2 — Type: int(4) — Collation: — Attributes: — Null: Yes — Default: NULL — Comments: — Extra: Name: command_timestamp — Type: timestamp — Collation: — Attributes: — Null: No — Default: current_timestamp() — Comments: — Extra:
Drive Readings	Name: id — Type: int(6) — Collation: — Attributes: UNSIGNED — Null: No — Default: None — Comments: — Extra: AUTO_INCREMENT Name: PositionX — Type: int(6) — Collation: — Attributes: — Null: Yes — Default: NULL — Comments: — Extra: Name: PositionY — Type: int(6) — Collation: — Attributes: — Null: Yes — Default: NULL — Comments: — Extra: Name: Speed — Type: int(6) — Collation: — Attributes: — Null: Yes — Default: NULL — Comments: — Extra: Name: Yaw — Type: int(6) — Collation: — Attributes: — Null: Yes — Default: NULL — Comments: — Extra: Name: reading_time — Type: timestamp — Collation: — Attributes: — Null: No — Default: current_timestamp() — Comments: — Extra: ON UPDATE CURRENT_TIMESTAMP()
Energy Readings	Name: id — Type: int(6) — Collation: — Attributes: UNSIGNED — Null: No — Default: None — Comments: — Extra: AUTO_INCREMENT Name: reading_time — Type: timestamp — Collation: — Attributes: — Null: No — Default: current_timestamp() — Comments: — Extra: ON UPDATE CURRENT_TIMESTAMP() Name: BatteryLevel — Type: float — Collation: — Attributes: — Null: Yes — Default: NULL — Comments: — Extra:
Obstacle Locations	Name: id — Type: int(6) — Collation: — Attributes: UNSIGNED — Null: No — Default: None — Comments: — Extra: AUTO_INCREMENT Name: Color — Type: varchar(30) — Collation: utf8_unicode_ci — Attributes: — Null: No — Default: None — Comments: — Extra: Name: X — Type: int(6) — Collation: — Attributes: — Null: No — Default: None — Comments: — Extra: Name: Y — Type: int(6) — Collation: — Attributes: — Null: No — Default: None — Comments: — Extra: Name: reading_time — Type: timestamp — Collation: — Attributes: — Null: No — Default: current_timestamp() — Comments: — Extra: ON UPDATE CURRENT_TIMESTAMP()

5.1.4 Security Measures

To enforce security and privacy, the remote database is only able to communicate with clients which can authenticate with the server. This method of authentication is implemented using an API key. The rover stores this API key as a variable while the front-end web page stores this API key in the relevant PHP scripts. Alongside the requirement to sign in to the remote database using MySQL credentials, this two-step authentication method ensures that data cannot be sent or received from any unknown device and prevents malicious attacks such as SQL injections that can harm the Mars Rover system by sending false commands to the rover.

5.1.5 High-Level Overview

Figure 5.2 outlines both the intramodular data flow between the components of the command subsystem, alongside the inter modular data flow between command and the rest of the rover via the ESP32 client.

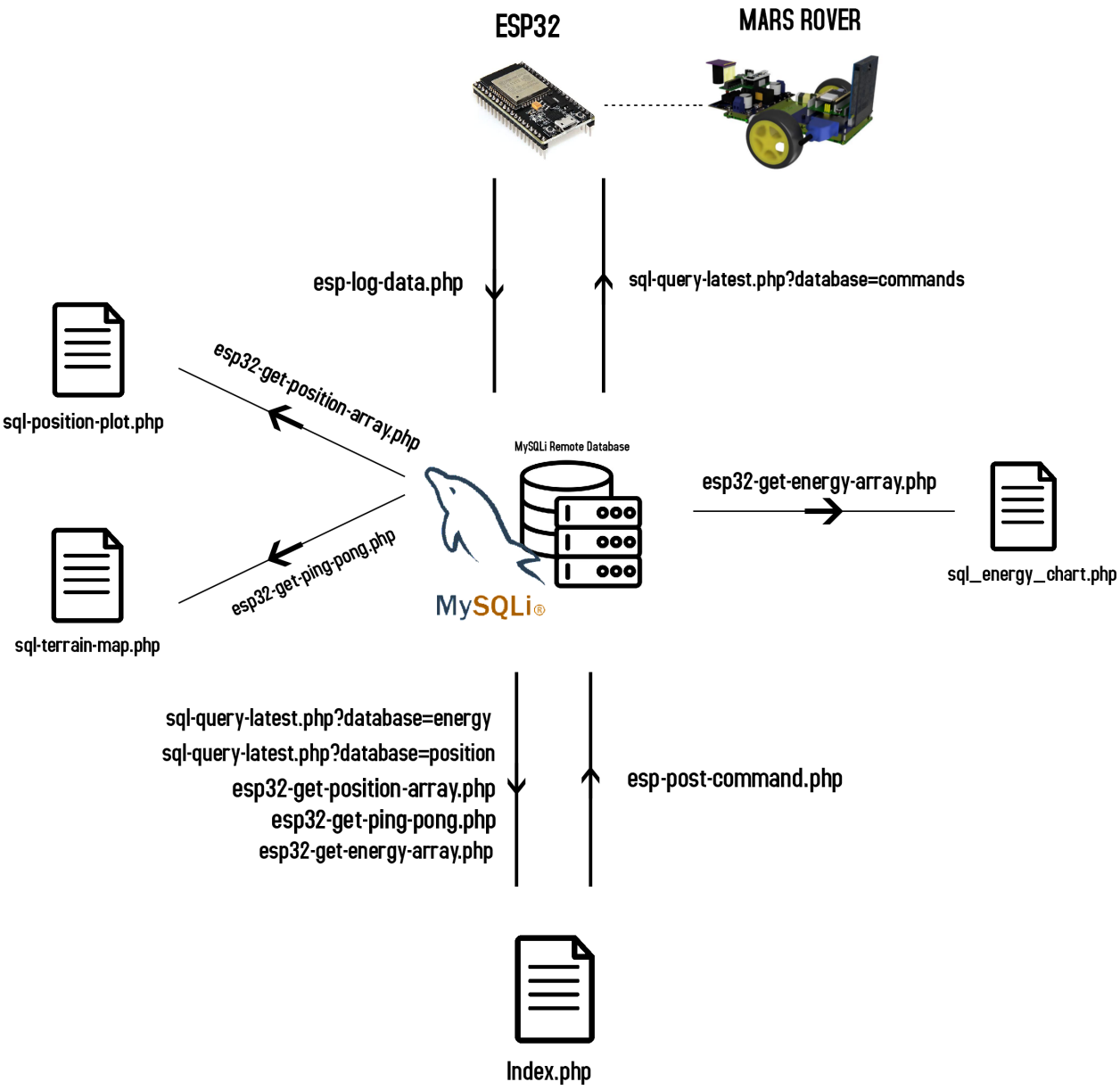


Figure 5.2: High-level functional diagram between the command subsystem and the rover

5.2 Vision

5.2.1 Overall Ball Detection Algorithm

The objective of the vision module is to identify the presence of five ping pong balls of different colours – red, green, blue, yellow and pink – in an unknown terrain and to compute the location of the balls relative to the rover's position and orientation and thus obtain data needed for building a map of the terrain using a camera sensor.

Figure 5.3 illustrates how the images or the pixels are processed into suitable formats for object detection and to minimise noise. The raw camera data captured by D8M-GPIO camera enters the FPGA video pipeline, represented as the green box, where TERASIC_CAMERA block converts it to RGB video packets. Then the video frames are buffered to allow the camera and the output running at different frame rates. These RGB pixels are low-pass filtered by a convolution filter to maximise the signal to noise ratio (SNR) by reducing the level of noise using a 3-by-3 Gaussian kernel. The filtered RGB packets are passed through TERASIC_AUTO_FOCUS block which sends commands to the camera once it measures the blurriness of the image.

Subsequently the RGB packets are processed by an image processor block called EEE_IMGPROC where the colour space conversion from RGB to HSV is done and bounding boxes are drawn for each ball present in the angle of view of the camera after some processing on data packets which are received pixel by pixel and row after row. Any information upon the detection of the balls is sent to the control submodule via RS232 UART from the NIOS II processor as messages that gets queued.

After the processing is complete, modified RGB packets with bounding boxes drawn onto the image are sent to the clocked video output where it exits the video pipeline, entering the VGA port, which can be displayed on a computer through a VGA converter for debugging and testing purposes. This is possible as EEE_IMGPROC contains two streaming pipeline registers where changes can be made to the packets as it passes between the two. In this section, the detailed implementation of the blocks represented as blue in vision submodule will be discussed in the following section.

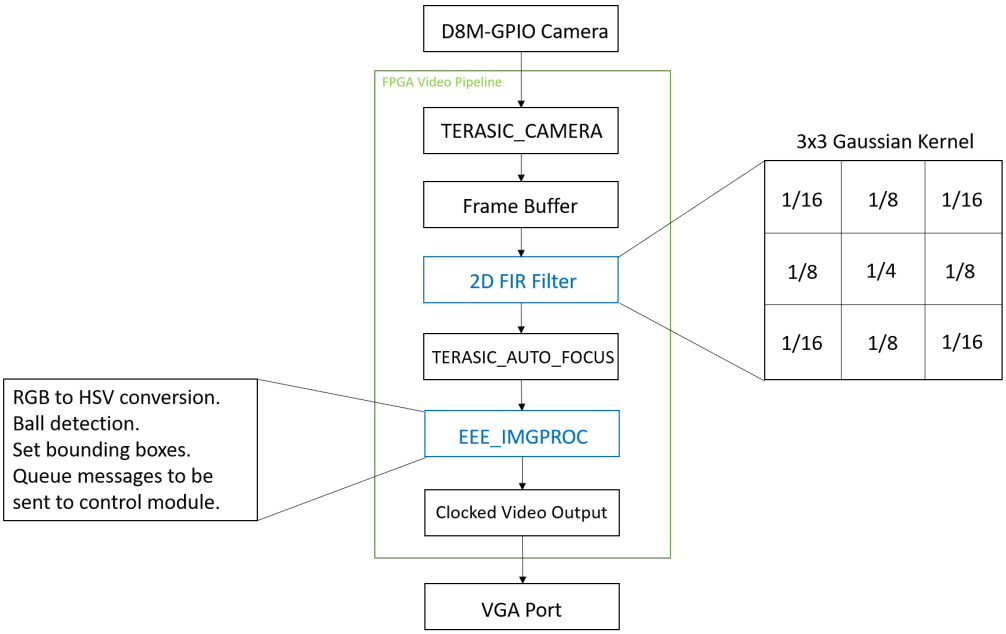


Figure 5.3: Overview of vision module and image processing.

5.2.2 Gaussian Convolution Filter

One of the major problems faced in real-life data or sensor readings is that signal noise due to randomness in nature leads to misinterpretation of data available and this is equally the case for image processing and computer vision. Noise in images can be caused by random variation in brightness or colour information, which is a form of electronic noise, produced by the image sensor or the circuits contained within it. As the raw video data has to be processed in a form of pixel stream, it is critical to remove high frequency noise in images to prevent unwanted or unexpected processing on individual pixels in hardware.

A number of solutions can be sought, for example, the camera focus setting could be deliberately defocused to have a general blurring effect on the image, or the camera exposure setting could be reduced to permit less light to enter the camera lens to block out bright pixels with high RGB content. However, such settings can delimit the original capability of the camera and put constraint on the performance by reducing signal and reducing noise at a higher rate to increase the overall SNR. The optimal approach would be to maximise the signal from the sensor readings and to minimise the noise to exploit the information available as well as maximising the SNR with a slight trade-off with computation cost for processing raw RGB data packets. After experimenting with what merely changing the camera settings can achieve, it was decided to insert a convolution filter in the video pipeline using the Intel IP core 2D FIR II Filter.

There are numerous types of convolution filters that could be applied to images. The 2D convolution filters could vary in size such as 3x3 or 5x5 filters, as well as how the coefficients are chosen. A convolution filter could be an average filter that computes an average of the neighbouring pixels to that of at the centre of a kernel or it could be a Gaussian filter where the coefficients are based on the standard deviation of a joint normal distribution of two random variables for a spatial filter. Figure 5.4 displays an example of a 3x3 average filter and a 3x3 Gaussian filter. One key aspect in determining which type of convolution filter to use is its computational cost. Since the kernel coefficients should add up to one as it would lead to loss of information or bright pixels if otherwise it exceeds one, for the 3x3 average filter the coefficients are 1/9 that is equivalent to the recurring decimal of 0.11111 and so on. Following the IEEE 754 Floating Point standard, representing and performing arithmetic especially multiplication then summation is computationally expensive. To represent a number that is close to 1/9, it would require around 10 bits of floating points. However, the 3x3 Gaussian kernel is of negative exponents of the number 2 and thus only requires about 5 bits to represent them all exactly including 1/16 which is the smallest coefficient of all.

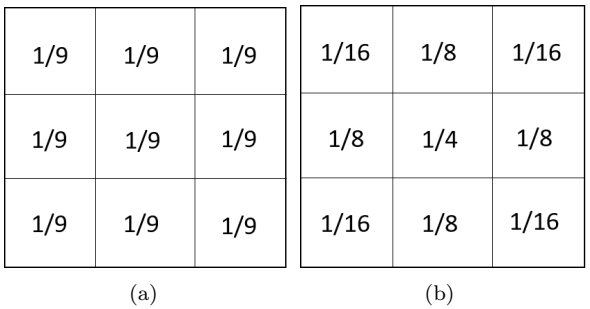
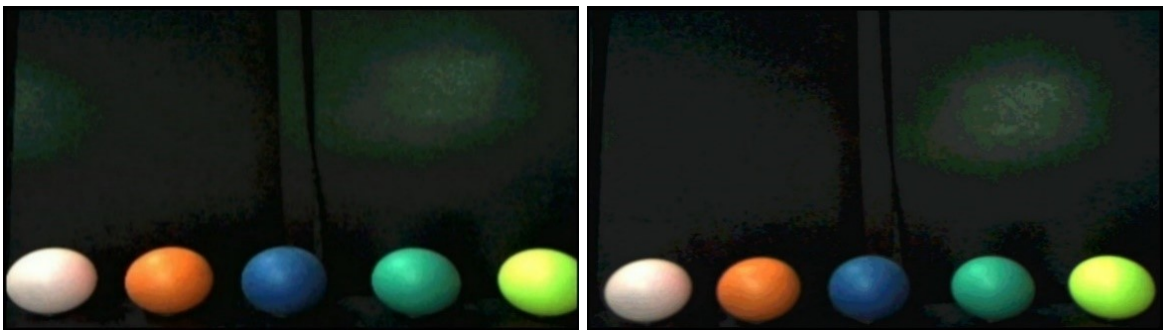


Figure 5.4: Examples of convolution filters. (a) 3x3 average kernel. (b) 3x3 Gaussian kernel.

Although a kernel size of 5x5 or larger could be preferable in terms of reducing the noise in the pixels, yet their computation costs are non-negligible especially where the video feed in the pipeline is running at certain frame rates and at some exposure setting. Therefore, 4 different combinatorial tests were done using 3x3 or 5x5 kernel size and the average or Gaussian filter. The effect of average filtering was equivalent to blurring the image and thus can be achieved by defocusing the camera without having to execute 10 bits floating point arithmetic operations. To add on, as the video image size is 640x480, 5x5 filters were likely to distort the camera video output or result in a black screen, indicating possibly a broken pipeline.

Moreover, when more memory was allocated for the UART file pointer for the communication with control submodule, it was unable to fit the design on the FPGA. Therefore, considering various aspects regarding computational cost and the ability to low-pass filter the noise to maximise the SNR, 3x3 Gaussian filter as shown in Figure 5.4 is used. The output display when the filter is applied is shown by Figure 5.5 on the right comparing to the one without the filter on the left. Although it is not quite visible, the image on the right tends to be slightly segmented into contours of colours as if it was quantised into a number of colour intensity levels, which is the desired behaviour of using a Gaussian convolution filter.



(a) (b)

Figure 5.5: Comparison of the video output when filter is and is not applied. (a) Without filter. (b) With filter.

5.2.3 Colour Space Conversion

Processing the image for detection requires certain feature extraction by examining and manipulating the pixels including the neighbouring ones to find the candidate pixels that could be part of the target object. The data or information available for the ball detection is the 24-bit RGB values for each pixel and no more. This data on the RGB colour space could be viewed on a different aspect at a different domain by the colour space conversion. The choice of which colour space to use to analyse the pixels can directly affect the performance of an object detection algorithm. The limitation with the default values of RGB is that each 8-bit R, G and B field contains the information about the colour as well as the brightness factor. This means that RGB values will differ under different light settings of the environment and therefore the entire algorithm would not be robust as it would depend on the light settings. Figure 5.6 outlines several options of the colour space to perform the algorithm on. As it is essential to separate the colour aspects and brightness aspect of an image unlike the RGB, a combination of grayscale and RGB could be used to set a limit on brightness or HSV could be used as it uses hue (H), saturation (S) and value (V) that distinguishes colour using hue and saturation and brightness using value. Hue refers to the pure colour whereas saturation represents the purity level and intensity of colour and value refers to the relative lightness or darkness of a colour.

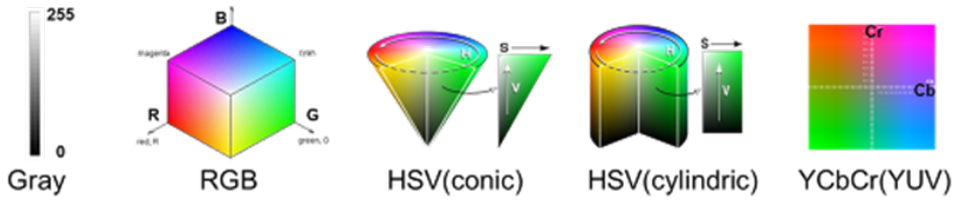


Figure 5.6: Different types of colour spaces [6].

The RGB colour space can be converted to HSV by the following formula outlined in Equation 5.1 to 5.6 [7]. Hue ranges from 0 to 360 whereas saturation and value can range from 0 to 100. However, Equations 5.4 and 5.5 are modified such that saturation and value ranges from 0 to 255 to be able to set ranges to colours more precisely.

$$C_{\max} = \max(R, G, B) \tag{5.1}$$

$$C_{\min} = \min(R, G, B) \tag{5.2}$$

$$\Delta = \mathrm{C}_{\max} - \mathrm{C}_{\min} \tag{5.3}$$

$$H = \begin{cases} 0, \Delta = 0 \\ (G - B) \times 60/\Delta, C_{max} = R \\ 120 + (B - R) \times 60/\Delta, C_{max} = G \\ 240 + (R - G) \times 60/\Delta, C_{max} = B \\ H + 360, H < 0 \end{cases} \tag{5.4}$$

$$S = \begin{cases} 0 & , C_{\max} = 0 \\ \frac{\Delta \times 255}{C_{\max}} & , C_{\max} \neq 0 \end{cases} \tag{5.5}$$

$$V = C_{\max} \tag{5.6}$$

The EEE_IMGPROC has certain HSV ranges for each of the different colours of a ping pong ball (red, green, blue, yellow and pink) that each pixel should fall into in order to be detected as one of the colours of balls as in Appendix. An extra condition on the HSV ranges is set using the RGB values that puts a small constraint to minimise falsely detecting a wrong colour. All pixels that do not fall into the certain colour ranges, then the pixels are classified as grey or being not part of any balls.

5.2.4 Localisation and Mapping of Terrain of Balls

Once all the pixels have been classified as either one of the balls or none (grey) for the current frame, the next step is to set a bounding box for each ball to identify its pixel dimensions and pixel coordinates of the box to compute and localise the ball on a map. The bounding boxes can be drawn after finding the minimum and maximum u, v values of the pixels classified as a ball of a colour as in Figure 5.7. However, this approach tends to inflate the bounding boxes, not fitting closely enough to the ball. Therefore, rather than using the minimum and maximum pixel coordinates, second or third minimum and maximum pixel coordinates are used to consider faulty pixel classification that could occur far away from the actual ball on the camera display or from a different colour of a ball that is right next to each other on the colour spectrum such as yellow, green and blue or red and pink.

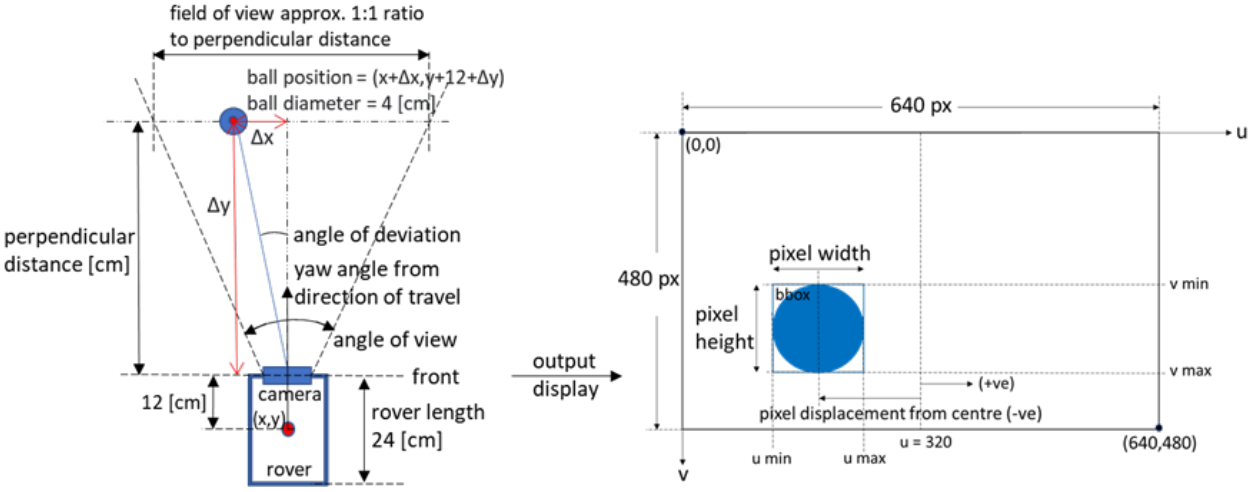


Figure 5.7: Different types of colour spaces.

For the localisation and mapping of the balls detected by computer vision, the relative position in x and y direction with respect to the rover's position must be computed from the information given by the bounding boxes. The key metrics for ball localisation are outlined in Figure 5.7. The bounding boxes

provide the pixel width and height of the boxes and the pixel coordinates of the vertices of the box as shown by the modified video output in Figure 5.7. These coordinates are sent to the control module as seven 32-bit messages by using a 3-bit finite state machine (FSM) first-in first-out (FIFO) queue and the following computation are done on ESP32 as it is much efficient than being done on hardware using Verilog.

Using these values, the relative position of the balls can be found by computing the perpendicular distance from the camera to the ball and horizontal pixel displacement from centre of camera display as shown in Equation 5.7 to 5.9 [8]. To approximate the perpendicular distance, a reference measurement had to be made to calculate the perceived focal length, which required a measurement of the diameter of the ball, the number of pixels taken up on the 640x480 output and the actual perpendicular distance from the camera on the rover. The estimated perceived focal length is found to be 800 pixels, which can be reused to compute the estimated perpendicular distance given the size of the bounding box. The pixel displacement computes how far the bounding box are from the centre of the rover or the camera output as indicated by a vertical line at u = 320 px.

$$\text{perceived focal length} = \frac{\text{pixel width} \times \text{distance from camera to ball}}{\text{ball diameter}} = \frac{71\ \text{[px]} \times 45\ \text{[cm]}}{4\ \text{[cm]}} \approx 800 \tag{5.7}$$

$$\text{perpendicular distance} = \frac{\text{ball diameter} \times \text{perceived focal length}}{\text{pixel width}} = \frac{4\ \text{[cm]} \times 800\ \text{[px]}}{\text{pixel width}\ \text{[px]}} = \Delta y \tag{5.8}$$

$$\text{pixel displacement} = \text{bounding box } x \text{ min} + \frac{\text{pixel width}}{2} - 320\ \text{[px]} \tag{5.9}$$

The conversion from the pixel displacement on the camera output to the actual angle of deviation from the rover can be computed as in Equation 5.10. For the arctangent function atan2() is used to correctly determine the quadrants.

$$\text{angle of deviation} = 90 - \arctan\left(\frac{\Delta y + \text{rover length}}{\Delta x}\right) = \arctan\left(\frac{\Delta x}{\Delta y}\right) = \theta\ \text{[rad]} \tag{5.10}$$

From the perpendicular distance (Δy) and angle of deviation to a ball (θ), the location of the ball relative to the rover (Δx, Δy) can be computed by Equation 5.13. The approximation for Δx is based on how the distance to a ball and the size of the camera view width is of roughly 1:1 ratio after a reference measurement on camera and following that the field of view is assumed to be linear rather than being curved. The ratio of the half of the width of the camera screen (320 px) to the pixel displacement is equal to the ratio of the half of Δy to the Δx, thus leading to Equation 5.13.

$$320\ \text{[px]} : \text{pixel displacement} = \frac{\Delta y}{2} : \Delta x \tag{5.11}$$

These two values can be used to compute the polar coordinates of the ball setting the rover as the origin. The polar coordinate system is needed as the yaw angle from drive module needs to be added to the angle of deviation of a ball from the rover to compute the actual position on the map. For Figure 5.7, as the yaw angle is zero, polar angle is the same as the angle of deviation.

$$\text{polar radius} = \sqrt{(\Delta x)^2 + (\Delta y)^2} = r \tag{5.12}$$

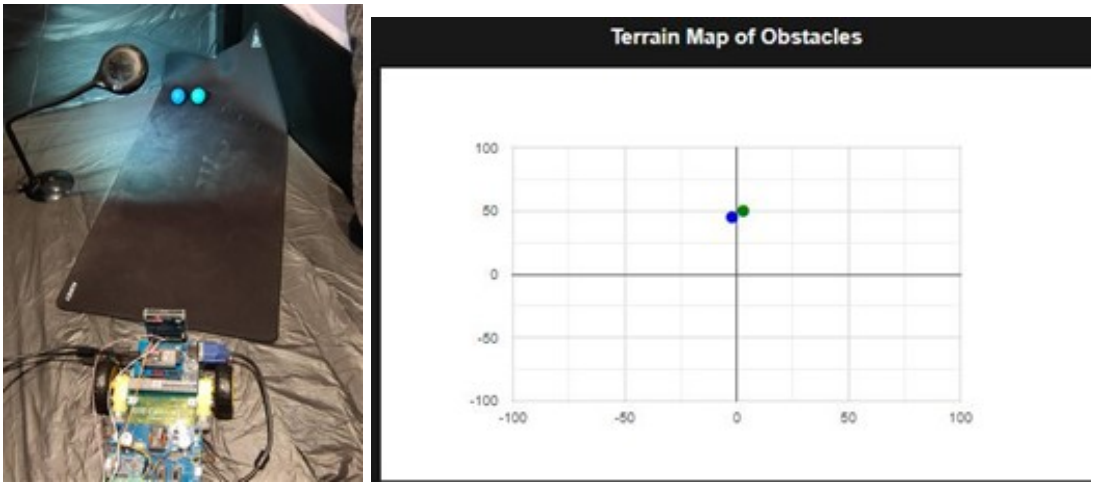
$$\text{polar angle from origin} = \text{rover yaw angle} + \text{angle of deviation} = \psi\ \text{[deg]} \times \frac{\pi}{180} + \theta = \varphi\ \text{[rad]} \tag{5.13}$$

The polar coordinates can then be transformed to a new relative x and y displacement that has taken into account of the rover's yaw angle is then added onto the rover's (x,y) coordinates to obtain the coordinates of the balls on a map as in Equation 5.14 and 5.15.

$$x \text{ coordinate of } a \text{ ball} = x \text{ displacement of rover} + r \times \sin(\varphi) \tag{5.14}$$

$$y \text{ coordinate of } a \text{ ball} = y \text{ displacement of rover} + r \times \cos(\varphi) + 12 \text{ [cm]} \tag{5.15}$$

Figure 5.8 displays the rover's vision module under operation to build a terrain map of obstacles using its camera. The testing environment is set out on a dark background where blue and green, the two hardest balls to differentiate in between using pixel information, are successfully detected and mapped with a high accuracy of distance and angle of deviation.



(a) (b)

Figure 5.8: Vision module at operation at a terrain of obstacles. (a) A terrain of obstacles. (b) Terrain map obtained.

5.2.5 Environment and Camera Setting

As in Figure 5.8, the environment is set out on a dark background to primarily filter out unwanted noise from the background due to lights from reflective surface or directly from light sources which are bright and thus highly likely to cause faulty pixels. This is expected to maximise the functionality and effectiveness of the vision ball detection algorithm. Several settings can be made on the camera to suit the actual use under the terrain environment such as the exposure time, gain and focus settings. The exposure time setting can alter the amount of light that can enter the camera, but it is limited by the frame rate of the camera which is 60 fps. Gain setting can change the level at which the signals are amplified from the camera sensor. The focus setting controls how sharp and detailed the images are and this parameter can be used to blur out the images to remove some of the noise in the background. The exposure setting is chosen as the hexadecimal of 0x2500 and the gain setting is set as 0x888 and for the focus setting, to have some blurring effect, it is set as 300 in decimal.

5.2.6 Testing and Improving Robustness

Despite having a spatial filter and algorithms for setting the bounding box appropriately, noisy pixels are inevitable in most cases and especially in an unknown terrain. Therefore, another layer of filtering or processing of what has been detected by the vision module was required to make the whole system more robust to external factors. Once the bounding boxes are found and the vertices coordinates are transmitted, the extra layer of processing is done by other modules such as the control and command. The first layer of processing is done on the control submodule where it would filter out bounding boxes that are not close to a square such as a rectangular bounding box due to erroneous pixels somewhere in the image also known as glitches. The subsequent layer is to filter by the size or area of the bounding box. For example, if the bounding box size is too small or too large, which should be undetectable by the ball detection algorithm, these boxes would be filtered out and not be sent to the database. Moreover, since the rover is being tested on a flat and even surface, the coordinates of the centre of the bounding boxes

of each obstacle will always be below a certain vertical height. Thus, bounding boxes which exceed this height will be filtered out. The final layer of processing of such faulty detections is done on the command submodule where SQL queries are designed to take the five most recent tuples and take an average of the entries for each ball when visualising the ball location data on a two-dimensional (2D) map of terrain.
Overall, the development process of the vision algorithm has followed the testing flow as shown in Figure 5.9. There were several goals to be achieved that are shown as in rhombus blocks where each one had to be satisfied before moving onto the next step. If one of the goal was not satisfied, the detection algorithm had to be improved by making changes to the code and its logic or finding a better way.

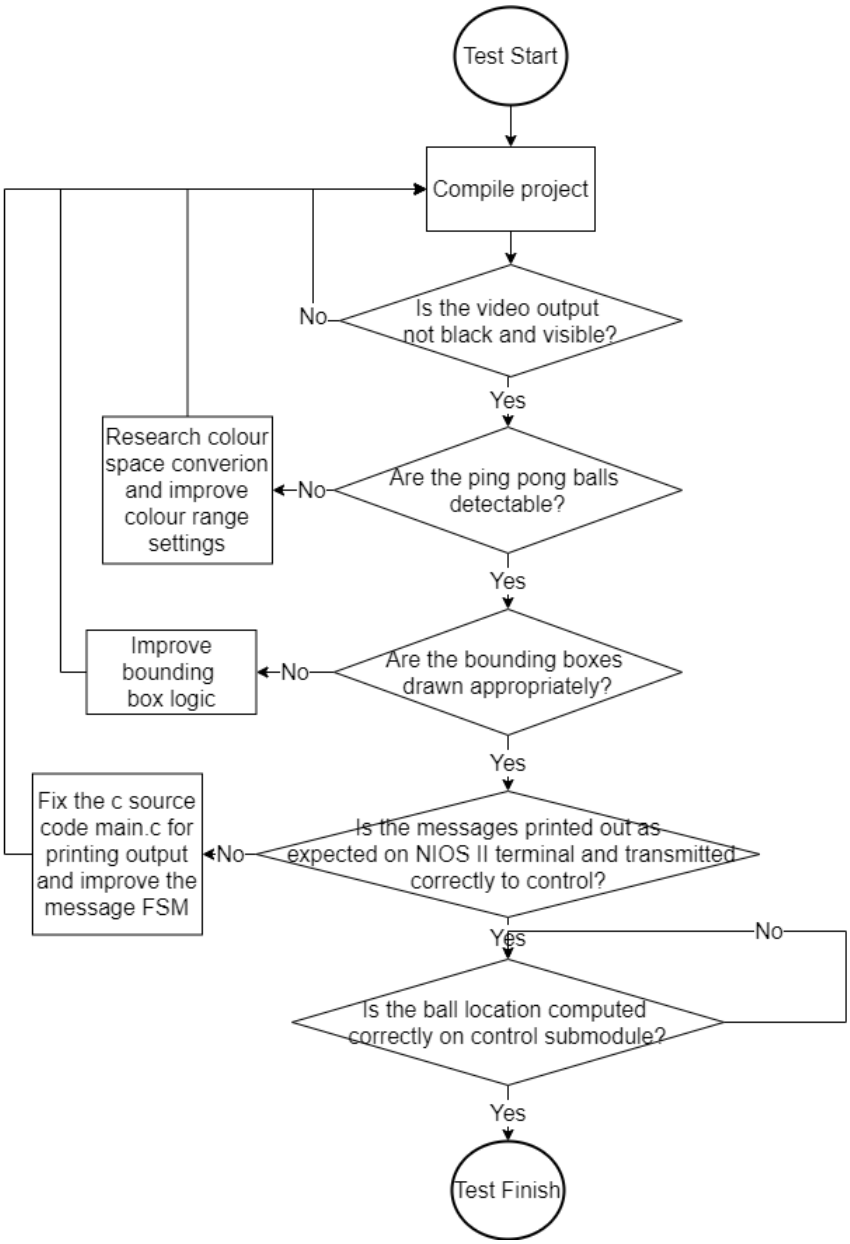
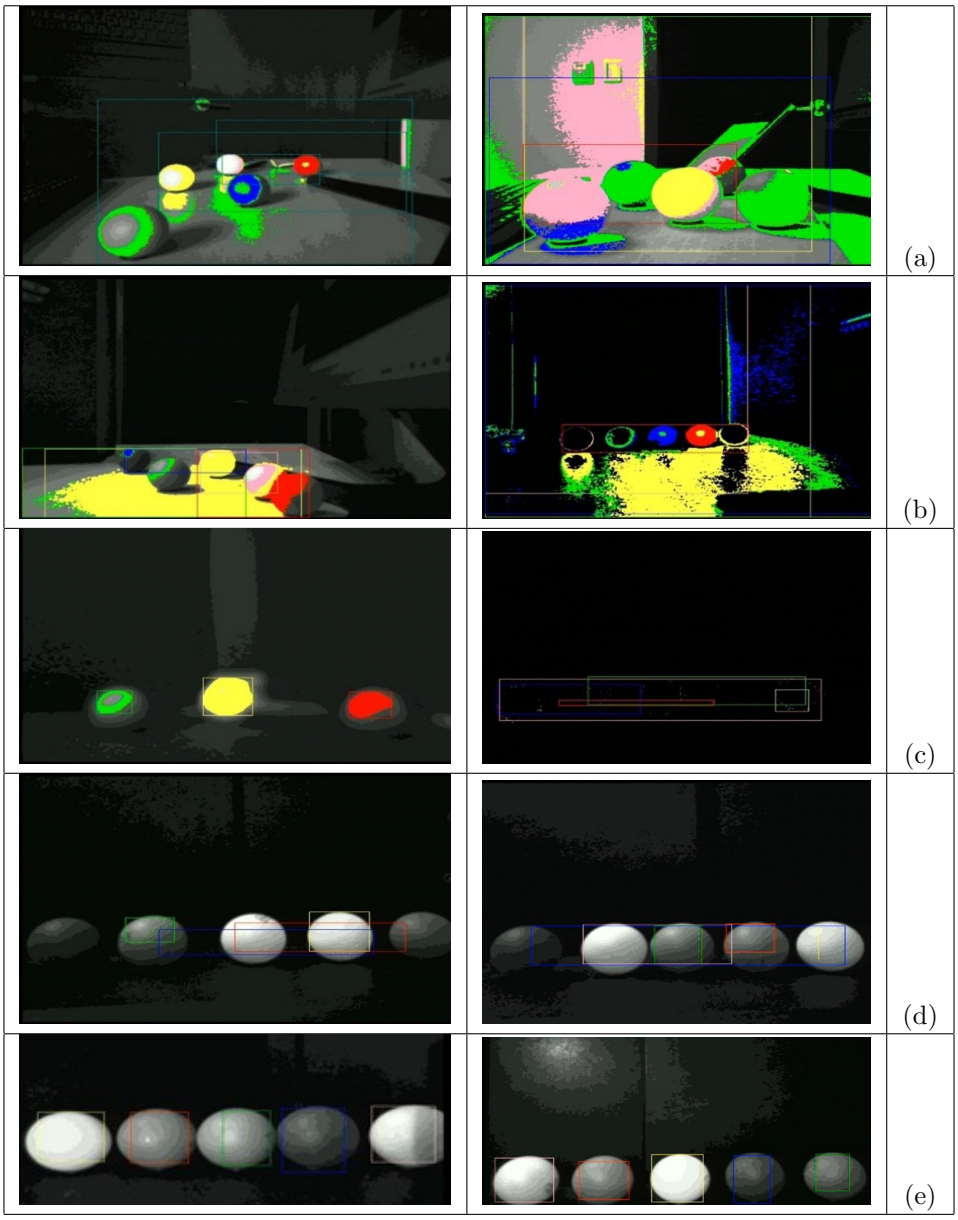


Figure 5.9: Testing flowchart for vision ball detection algorithm.

Following such testing flowchart, there had been numerous trials and errors and varying attempts to better the vision algorithm. Some of these milestones of attempts are laid in Table 5.4. The first attempt (a) was using purely the RGB to detect all five balls at once, where the algorithm encountered a problem that it was extremely dependent on the light setting and prone to noise pixels. The approach (b) started to use the colour space YCbCr as shown in the image on the left and the image on the right illustrates a combined attempt of RGB and YCbCr. Still the bright areas had yellow and green noisy pixels and the dark areas had green and blue noisy pixels. The trial (c) introduced the convolution filter along with the colour space HSV which can be seen by the drastic reduction in noise pixels with some success in being able

to distinguish different colours separately. The attempt (d) involves finding colour ranges that minimise overlapping ranges between different colours. Yet some bounding boxes are rectangular and does not fit the balls to the desired level. The final attempt (e) involves taking not the actual minimum or maximum pixels for bounding boxes but the second or third minimum or maximum pixels to remove overlapping boxes and finding optimal bounding boxes of the detected balls.

Table 5.4: Images of different trials following the test of different milestones and timestamps.



5.3 Drive

5.3.1 Introduction to Drive and Drive-Control Communication

The aim of the drive module is to implement the command received from the control module to control the movement of the rover and compute the position of the rover in the x,y coordinate system. A block diagram of the drive module is shown in Figure 5.10 where its components are outlined.

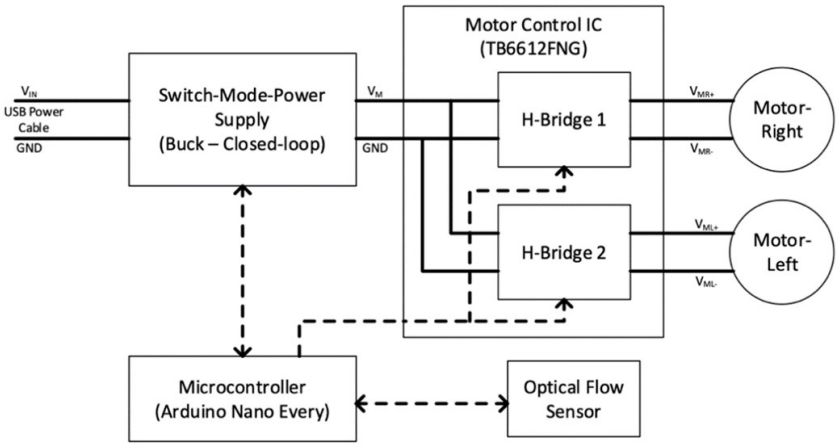


Figure 5.10: A block diagram of the drive module.

A buck SMPS in closed-loop mode is used. The output voltage of the SMPS can be controlled to control the speed of the motors. There are two H-bridge circuits in the motor control IC which are generally used in reverse the polarity of the motors. The optical flow sensor measures the change in position by optically acquiring sequential surface images and mathematically determining the direction and the magnitude of movement [3]. An Arduino Nano Every is used to control the SMPS, the motor control IC and the optical flow sensor. It communicates with the optical sensor via the SPI port to measure the distance travelled in x and y directions. The drive PCB provides serial communication port which allows the Arduino Nano to communicate with the control system. For instance, the control system sends the command to the drive system and drive system sends the information including position, speed and yaw to the control system via the serial port. The format of the output String is shown in Listing 5.1. Every time the command has been executed, the information and a String called "done" is sent. Once these are received by control, the next command would be delivered to drive.

```
String towrite = String(int(posx)) + "," + String(int(posy)) + ","
                 + String(roverspeed) + "," + String(int(overallang)) + "\n";
```

Listing 5.1: Format of the output String.

5.3.2 Command Parser

The command sent to drive is of String type in the format like "fw,10,0" and "tr,90,0" through the Serial port. Aiming to obtain useful information, parsing the command is an inevitable step. When the command is sent, the result of "Serial.available()" becomes one. The first two characters of the command indicate the type of the movement. The substring "fw" is move forward and "rv" is move backward. If the substring is "tr" or "tl", turning right or left instruction is acquired. Then the corresponding boolean value is set to be one and the others are kept at zero. The application of function "indexOf()" locates the position of the two commas in command. In this case, the second information (expected moving distance and expected turning angle) is obtained. The relevant code is shown on Listing 5.2.

```
while(Serial1.available()!=0) {
    command = Serial1.readString();
    fcomma = command.indexOf(',');
```

```
    scomma = command.indexOf(',', fcomma+1);

    if(command.substring(0,2)=="fw"){
        forward = 1;
        backward = 0;
        right = 0;
        left = 0;
        yaw = 0;
        dista = command.substring(fcomma+1,scomma);
        expy = dista.toFloat();
    }
    ...
}
```

Listing 5.2: Parsing received commands.

5.3.3 Implementation of Low-Level Movement Commands

An important background knowledge about the displacement x and y, measured by the optical flow sensor, that needs to be noted is that while moving forward or backward, only the displacement y changes no matter what angle the rover is currently facing. Distance x changes only when a turning command is under progress.

Command: Forward and Backward

In order to achieve the movement of a fixed distance, the displacement in y (recordy) is recorded at the time at which the move forward or backward instructions are received (while Serial.available() is 1). This is added to the expected moving displacement to create a target displacement in y (tary). An error is calculated by subtracting the target displacement from the displacement in y (ey). For the move forward command, if the error is greater than zero and the boolean value of forward is equal to one, the rover is kept to moving forward. Once the error is smaller than or equal to zero, the movement of the rover is stopped and the boolean value of forward is set to be zero to prevent further influence on other upcoming commands. In contrast, the target distance is set to the recorded displacement in y minus the expected moving distance for move backward commands. The rover keeps moving when error is smaller than zero. It stops until the error is greater than or equal to zero. Listing 5.3 tracks the error discussed.

```
float recordy = total_y;
tary = recordy + expy;
ey = tary - total_y;
```

Listing 5.3: Error monitoring of rover displacement in y direction.

Command: Right and Left

Implementation of turning right and left use similar approach to previous forward and backward instructions. On the account of the significant bias in small angle rotation, only turning by 90 degree is implemented. The displacement in x is recorded when the turning command is called. While the rover rotates 90 degree, the change in distance x is fixed at approximately 17 cm. The error is computed differently for each turn right and left command. For the right turn, the error is set equal to the recorded distance x minus 17 then minus displacement x. The turning can be considered as rotating in a circle of radius 17 cm. The rover keeps turning until the error is greater than or equal to zero for right turn by 90 degree. Likewise, for turning left by 90 degree, the rover keeps turning until the error is smaller than or equal to zero. Aiming to achieve more accurate turning, a slower motor speed should be applied to reduce being affected by inertial force. Moreover, a variable called "overallang" is established to log the angle turned and distinguish what direction the rover faced now as in Listing 5.4.

```
dista = command.substring(fcomma+1,scomma);
expang = dista.toFloat();
overallang = overallang + expang;
recordx = total_x;
```

Listing 5.4: Parsing and executing the turn command.

5.3.4 Implementation of High-Level Movement Command

Command: Yaw Reset

Yaw reset is the high-level command designed for the angle restoration purpose of the rover. The execution of this command involves the following two steps. At the time the command is received, the variable "overallang" recorded previously in turning is divided by 90 to produce a variable called "reqturn" which is utilised in determining whether the rover needs to rotate and how many times the rover should turn to fulfil the purpose of restoring the angle to zero. If the "reqturn" is smaller than zero, this means that a right turn is required and vice versa. This is outlined in Listing 5.5. The subsequent step is to execute the right turn or left turn command depending on "reqturn" variable value.

```
if(command.substring(0,2)=="yw") {
    yaw = 1;
    reqturn = overallang/90;
    overallang = 0;
    if(reqturn < 0) {
        right = 1;
        forward = 0;
        backward = 0;
        left = 0;
        recordx = total_x;
    } else if(reqturn > 0) {
        left = 1;
        forward = 0;
        backward = 0;
        right = 0;
        recordx = total_x;
    } else if(reqturn == 0) {
        left = 0;
        forward = 0;
        backward = 0;
        right = 0;
        Serial1.write("done\n");
    }
}
```

Listing 5.5: Yaw reset command.

5.3.5 Rover Dynamic Status and Testing

Position

Owing to the displacement x and y measured by the optical sensor, which are not the coordinates on the x-y plane, the real position of the rover needs to be computed. The calculation of the position depends on the current angle and the distance moved by the rover. If the angle is 0 or 180 degree, the rover is moving in the y-axis. If the angle is 90 or -90 degree, the rover is moving in the x-axis.

```
if(overallang == 0) {                    if(overallang == 0) {
```

```
    posy = posy + expy;
} else if (overallang == 90) {
    posx = posx + expy;
} else if(overallang == -90) {
    posx = posx - expy;
} else if (overallang == 180 ||
           overallang == -180) {
    posy = posy - expy;
}
```

```
    posy = posy - expy;
} else if(overallang == 90) {
    posx = posx - expy;
} else if(overallang == -90) {
    posx = posx + expy;
} else if(overallang == 180 ||'
          overallang == -180) {
    posy = posy + expy;
}
```

Listing 5.6: Move forward command (left). Move backward command (right)

Speed

The speed of the rover can be controlled by varying the output voltage of the SMPS by using "analogueWrite(6, Step)" as in Listing 5.7. The range of the step is from 0 to 255. The larger the step, the lower the output voltage and the slower the speed. The method used to control the speed is simply by receiving a String.

```
void speedmodulate(String speedcontrolvar){
    if(speedcontrolvar == "slow") {
        analogWrite(6,190);
    } else if(speedcontrolvar == "slowturn") {
        analogWrite(6,235);
    } else if(speedcontrolvar == "medium") {
        analogWrite(6,120);
    }
}
```

Listing 5.7: Speed control

However, the actual moving speed of the rover is unknown. The way to calculate the speed is using the function "millis()" to record the time that the rover starts to move and the time that the rover ends the movement. Then simply dividing the moving distance by the time difference results the speed as in Listing 5.8.

```
float speedcalculation(unsigned long t1, unsigned long t2) {
    roverspeed = abs(expy)/float(((t2-t1)/1000));
    return roverspeed;
}
```

Listing 5.8: Speed calculation

Testing

Figure 5.11 outlines the testing flow of the drive module. To test the performance of the drive, the code is uploaded to the rover first. With the serial monitor in the Arduino IDE, it is checked the whether the rover moved to the desired position and checks errors from the actual drive movement compared to the target given from its command.

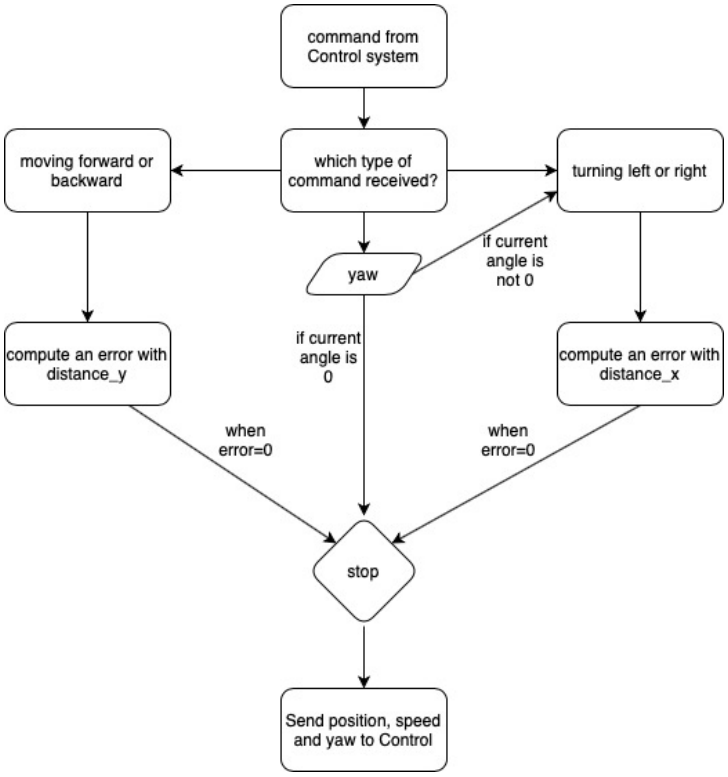


Figure 5.11: Testing flowchart of the drive module.

5.4 Control

5.4.1 Introduction to Control and Associated Protocols

The control module is responsible for connecting all the other subsystems together and relaying coherent data between them. Its main tasks include:

- Receiving commands from the command subsystem.
- Compiling the received commands into low-level commands for the drive module.
- Monitoring the rover's position, speed, yaw and battery life by receiving data from the respective sub modules.
- Receiving obstacle sightings from vision and performing the necessary processing to map them spatially for the command subsystem
- Controlling the autonomous behaviour of the rover by incorporating path-finding into its movement.

The life cycle of the control module is summarised by the flowchart in Figure 5.12 that performs the tasks mentioned above and interacts with the databases to store and query relevant data.

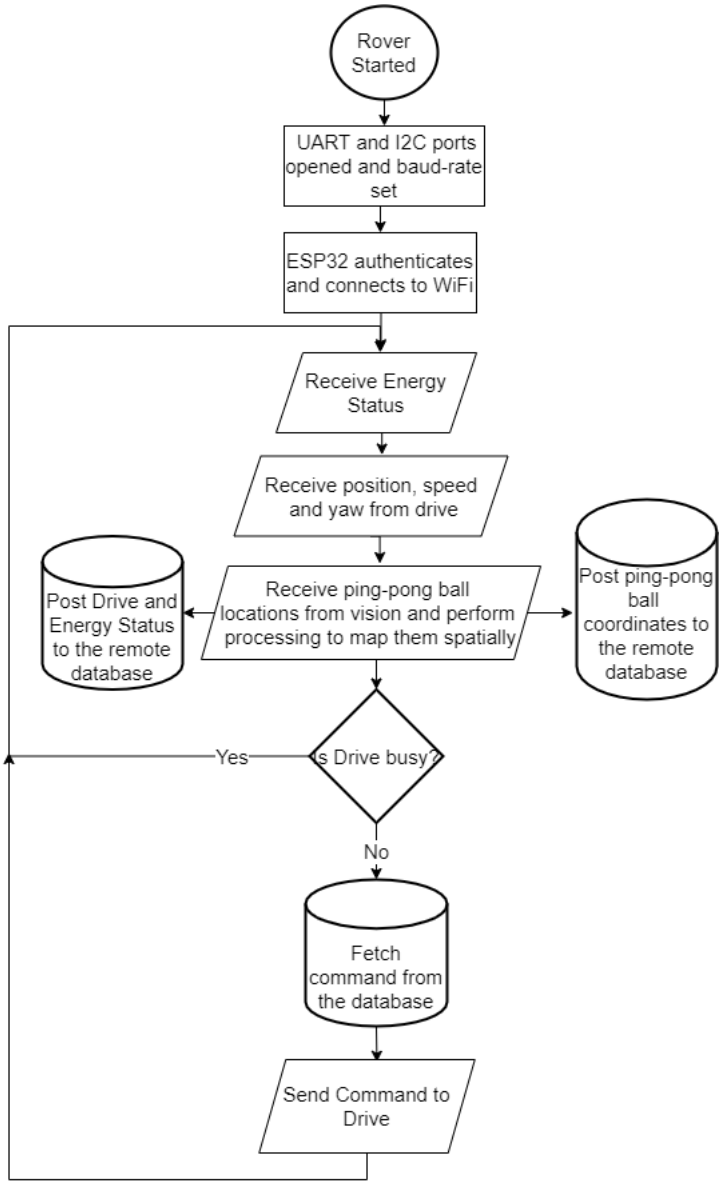


Figure 5.12: Life cycle of the control module.

5.4.2 Control-Drive Communication

The control module communicates with the drive module using the UART (Universal Asynchronous Receiver/Transmitter) protocol. The UART protocol was chosen due to the HardwareSerial library being built into the Arduino IDE which allowed for a much more intuitive and simpler design. The subsystems relay information to each other using CSV-formatted messages. The drive subsystem periodically relays its position, speed and yaw to the control module while the control module sends it new commands fetched from the command database. While a command is being executed by the drive subsystem, the control subsystem will not fetch any new commands until the drive subsystem indicates that it has completed its task. The baud-rate chosen for communication was limited by the maximum baud-rate of the Arduino Nano Every, where in this case it was locked to 9600 bps. Due to this notable decrease in the rate of transfer of information, the Drive subsystem was to be controlled within closed loop with low-level commands sent from the ESP32.

5.4.3 Control-Vision Communication

The control module communicates with the vision module using the UART protocol. Unlike the drive subsystem, the control module only communicates in one-way with the vision module. The FPGA sends a 224-bit message to the ESP32, encoded using 56 hexadecimal characters. This message contains the bounding coordinates for the boxes for each coloured ping-pong ball. The ESP32 then performs processing to calculate the spatial location of the ball using the information coordinates of each ping-pong ball visible on the camera as illustrated in section 5.2.4. The baud-rate chosen for this operation is 115200 to maximize throughput of information transferred between the subsystems. Processing was chosen to be performed on the ESP32 as it would be much more efficient than performing the calculations on Verilog. The UART protocol was chosen due to its ability to send and extract long strings of data compared to the SPI (Serial Peripheral Interface) protocol which required extracting each character individually.

5.4.4 Control-Energy Communication

The control module would communicate with the Energy module using the I2C (Inter-Integrated Circuit) interface. Due to the limitation of having only two UART ports, one of the modules would require a non-UART interface for communication. Energy was chosen due to the robustness of the Wire library, built into the Arduino IDE, for I2C communication and for the lower bandwidth and throughput required as compared to the Control-Drive and Control-Vision communications. The energy module has one-way communications with the control module like the vision module. The control module receives the battery status from the energy module which is sent as 4 bytes concatenated into a 32-bit integer.

5.4.5 Control-Command Communication

The control module uses the HTTP protocol to communicate with the remote web server over WiFi. The ESP32 acts as an HTTP client [9], sending POST requests to send drive, vision and energy data to their respective MySQL databases and using GET requests to fetch commands queued up in the command database. These HTTP requests are sent to their respective PHP scripts stored on the remote server which performs the required action. Each request uses the URL-encoded format to pack the required parameters into the request for the PHP script to extract. The HTTP protocol was used rather than the MQTT protocol due to its intuitive and simple method of interfacing with a remote MySQL database, as well as bring able to extract data much more efficiently due to its URL-encoded format. The ESP32 authenticates with the remote database using an API key such that no other device will be able to fetch and post data to the database without the correct API key [10].

5.4.6 High-Level Command Decomposition

Some of the higher level motion commands that the control subsystem receives from the command subsystem needs to be simplified before it can be sent to drive. The way this is implemented is by breaking down the high-level commands into several low-level commands which can be performed by the drive

subsystem one by one. The double-ended queue data structure is used to store these commands such that they can be pushed and popped in a first-in fisrt-out manner. Figure 5.13 outlines the process in which the high-level command is broken down and instructions are pushed to the queue using a flowchart.

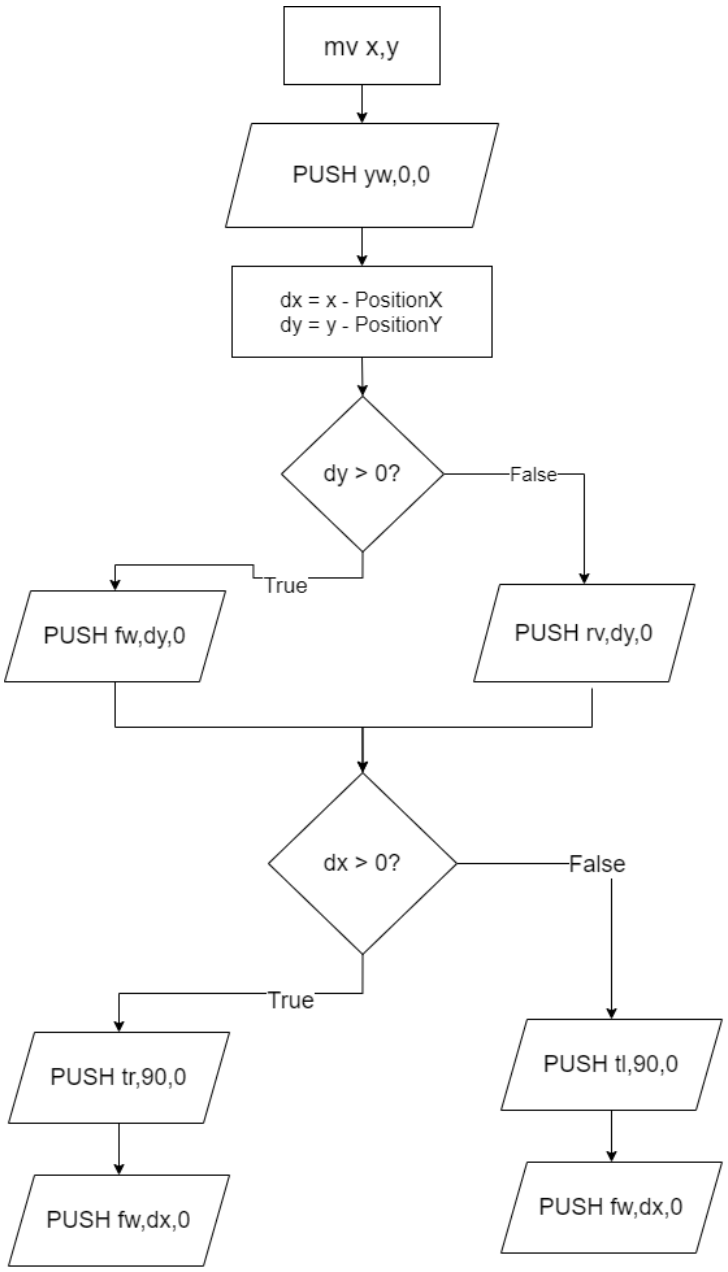


Figure 5.13: Path finding algorithm flowchart.

5.4.7 UART Communications Testing Procedure

Due to the limitations of performing a remote project, integrating the subsystems was made to be a much more challenging task. The initial method of testing UART communications was to use the Serial Monitor to test I/O as they used the same protocol for sending and receiving data as a normal UART communication. Once the message format was decided between Control-Vision and Control-Drive, the next task was to extract parts of each message into the required variables. This was done by sending test inputs through the Serial monitor to check if the decode algorithm yielded the proper results. Once this section of testing was complete, the next step would be to test it on the real rover through the Integration subsystem. Proper documentation was collected to ensure a smooth installation and configuration process from the integration side. Some of the problems which were faced during testing included:

- Receiving unintelligible strings, caused due to a discrepancy in baud-rates between the subsystems.
- Receiving repeated characters in each string, resolved by adding a small delay to the main code loop.
- Receiving nothing, resolved by checking connections.
- Strings received coherently but using the wrong parameter format due to an invalid data type. This was remedied by casting the parameters to the integer data type before transmitting.

Section 6

Integration and Full Rover Testing

6.1 Integrating and Testing the Inter-Module Communications

6.1.1 Command-Control

The earliest test which could be performed was the simple link between the command-and-control subsystems. This was done by transmitting arbitrary values using the HTTP protocol between the two subsystems and checking for any discrepancies. Initially, the Control subsystem was meant to act as the back-end remote server to the front-end web interface. However, the requirement for storing the data from both sides were required and due to the memory volatility and size limitations of the ESP32, the server was chosen to be hosted remotely. The testing procedure was then repeated using the ESP32 as a Client rather than a server and using PHP scripts to push and fetch data between the ESP32 and the remote database.

6.1.2 Command-Control-Vision

The first step in testing control to vision was by sending simple strings between each microprocessor. Since both devices used different libraries for UART communication, extra care was taken to ensure no discrepancy between the parameters involved in UART communication. Some of the key parameters to be decided were: digital pin numbers for communication, baud-rate, UART Format (start bits, parity, stop bits).

After strings were able to be received coherently, the actual data payload was transmitted, and the processing was done on the control subsystem. This was then relayed to the command subsystem, which was then checked for coherency, which in this case was checking if the obstacles were mapped to their correct location.

6.1.3 Command-Control-Drive

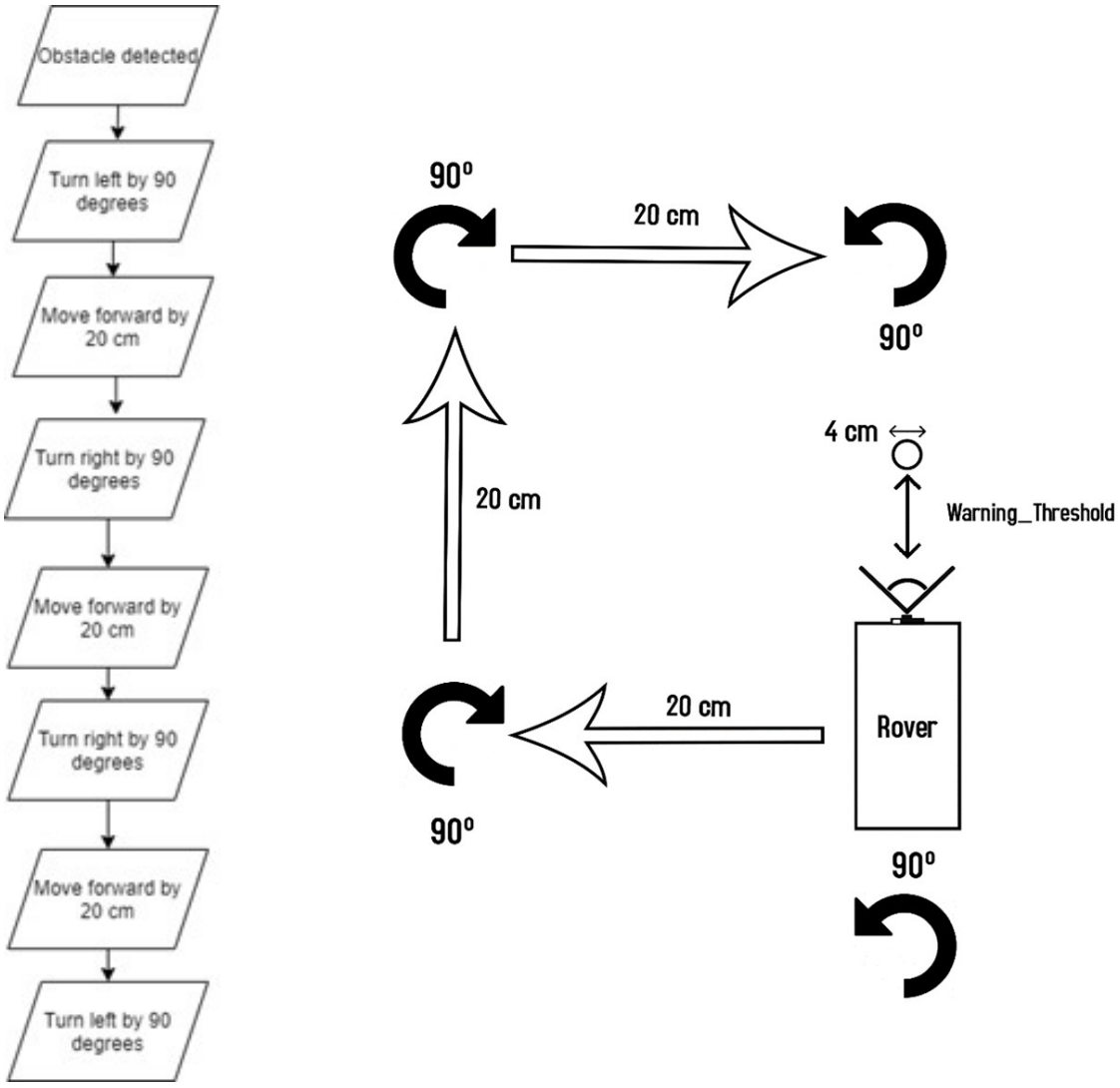
Since the Arduino and the ESP32 shared the same libraries, UART communication was intuitive and simple to implement. The only limitation faced by this setup was the baud-rate cap due to the Arduino Nano Every only being able to send at a maximum of 9600 bits per second from its UART port. The same testing procedure for vision was followed to test Control-Drive communications. Simple strings were sent at first to test coherency in transmitted and received data after which, more advanced testing was performed. Commands were sent through the web interface which were relayed to the ESP32 and then to the Arduino. The following steps were taken to check data coherency for this test:

- Check if the command was sent properly in the right format.
- Check if the command was received by the ESP32 in the proper format.
- Check if the command was relayed (and decomposed if required) to the Arduino.
- Check if the Arduino can parse the command and perform the required task.

6.2 Path Finding Algorithm

The path finding algorithm is implemented to equip the rover with an element of autonomy. The algorithm comes into play when the rover detects an obstacle on its path. The algorithm exploits the

fact that it can only be called during the move forward instruction, as the other low-level instructions either do not change the rover's position or do not move the camera's view closer to any obstacle. For the algorithm to be triggered, the obstacle must be at a certain distance away from the rover, calculated using the threshold variable. After the rover concludes that there is an obstacle in its path, the rover issues a stop command to the drive subsystem to prevent collision. Then it will attempt to steer itself out of the obstacle's path by performing a few extra commands. The algorithm for obstacle avoidance is shown in Figure 6.1(a) as a flowchart and in (b) that visualises the operation from a top-down view.



(a) (b)

Figure 6.1: Path finding algorithm (a) Flowchart. (b) Operation from a top-down view.

However, in some cases, the rover might encounter another obstacle while being in this obstacle avoidance state. The algorithm is designed to steer past any number of obstacles until it avoids all obstacles and moves to its required location. The path of the rover in the case of multiple obstacles is outlined in Figure 6.2.

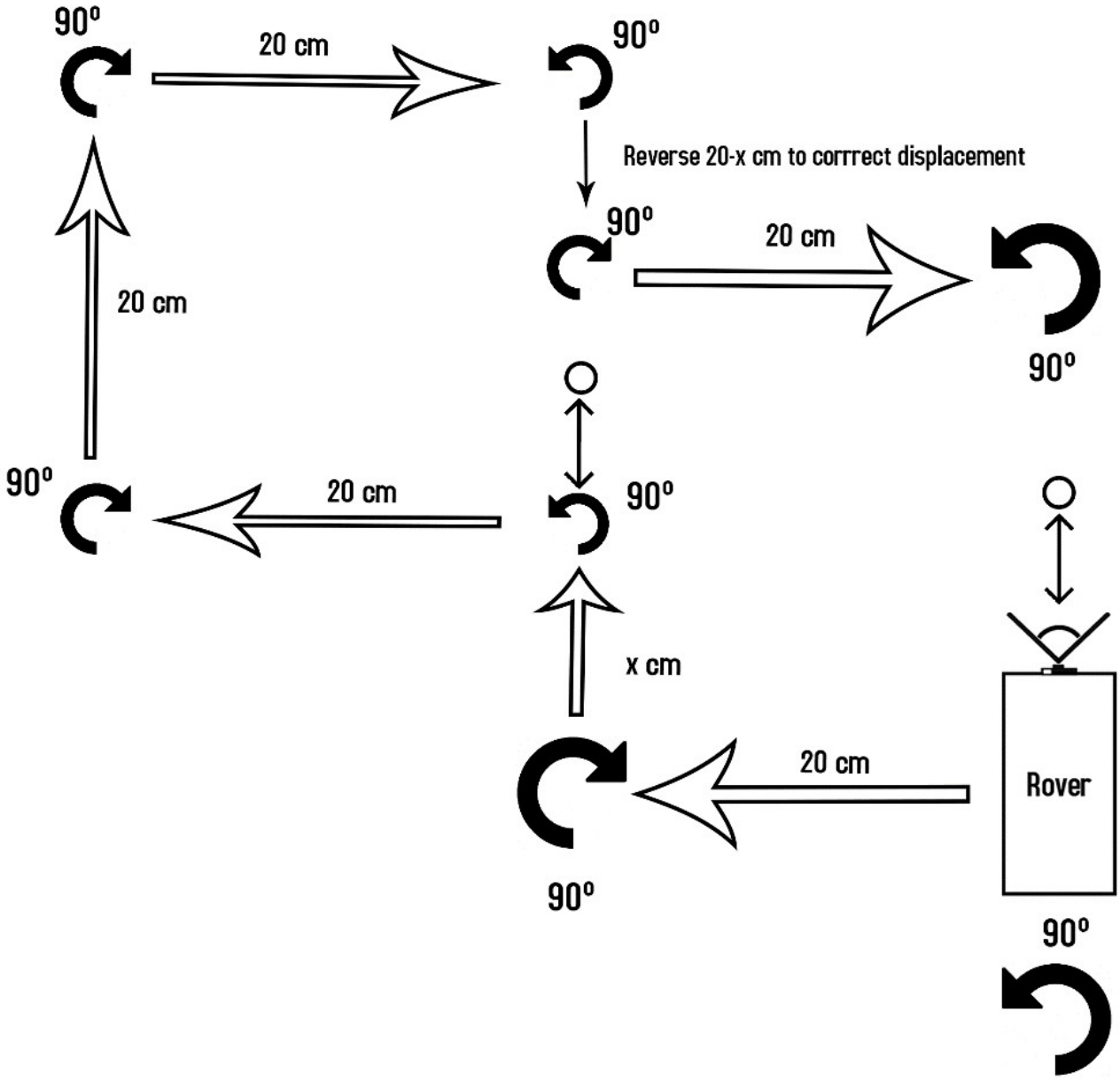


Figure 6.2: Path finding in a terrain of multiple obstacles.

6.3 Rover Testing Environment Setup

Due to the blind spot of the rover's camera, when the obstacles are too close, it enters the rover's lower blind spot, therefore the test environment needs to be wide enough to allow ping pong balls to enter the rover's valid vision area and be detected. In order to avoid any background colour that could interfere with the ping pong ball colours, we decided to build a black-to-dark-brown box as the test environment. To allow the obstacles within the sight of the rover as well as the rover moving and approaching tens of centimetres, we decided to build a 120 cm x 80 cm wide box. In consideration of leaking background colours into the box, if the box height is too low, the camera will overlook into surroundings outside of box, which may have colours that interferes with the ball detection as in Figure 6.3.

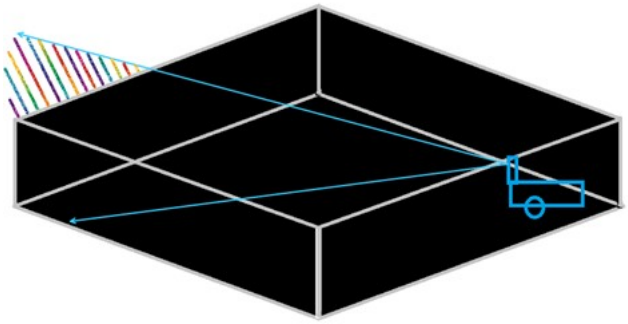


Figure 6.3: Vision range and background noise colour leakage.

Hence measuring from the diagonal distance (144 cm), the maximum height of the rover's vision is around 45 cm, therefore a box where the side of its walls are taller than 45 cm. In practice, the camera may be zoomed in to avoid outside colour leak into vision, but since the distance is short and ping pong balls are normally at the bottom of the frame, a normal zoom cannot perceive the ball as a whole and tends to see the top half of the balls as shown in Figure 6.4. Therefore we also managed to isolate the bottom half

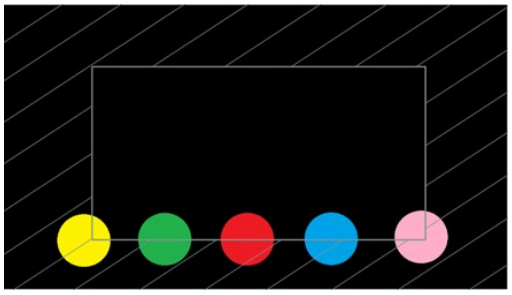


Figure 6.4: Normal zoomed in view from the camera.

only by ignoring the pixels on the top half of the frame as shown in Figure 6.5.

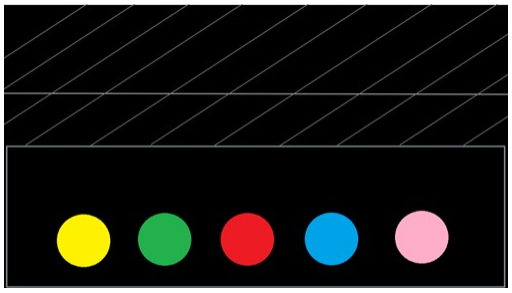


Figure 6.5: Actual view of the camera with the top half isolated.

The dark material has to be matt, so it does not reflect surrounding light and colour too much, and the floor of the box cannot be too light absorbing since the optical flow sensor operates based on the reflected red led light, hence the box floor is chosen to be dark brown wooden, which in practice worked out. The lighting direction also affects the colour detection algorithm, for example, for a blue ball, since the dark and non-reflective materials are surrounding it, light that is coming from the top of the box may mainly light up the top of the ball, leaving the bottom half too dark to detect as in Figure 6.6, which gives a false size of box. Thus, flash lights are added to be shining from below to lighten up and to evenly distribute the colours and light on the ping pong balls when testing.

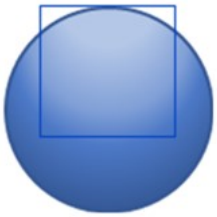


Figure 6.6: Faulty box on a ball of uneven light distribution.

The testing environment was setup physically as shown in Figure 6.7.

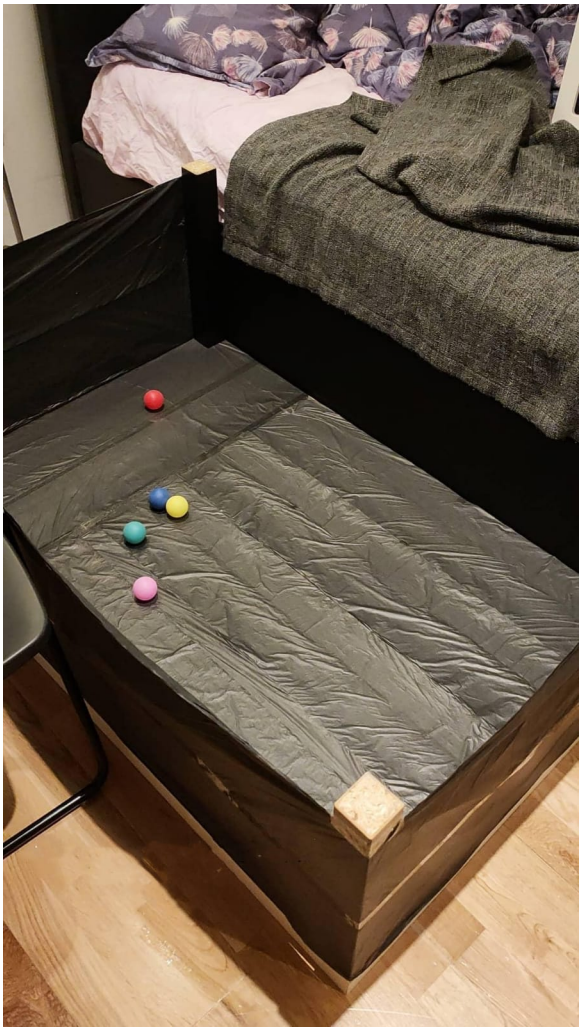


Figure 6.7: The testing environment.

Section 7

Performance Analysis

7.1 Performance of the Submodules

The resource utilisation of the DE10-Lite FPGA from Quartus is summarised as in Figure 7.1. A majority of the LUTs and logic registers is shown to have resulted from the NIOS II processor, the frame buffer block, the 2D FIR filter and the image processor compared to the rest of the other blocks that forms the FPGA system. A number of memory-mapped peripherals have been utilised where most of it were mapped to the onchip memory and other major blocks such as the terasic camera used. To add on, some DSP elements are present in the system due to the FIR filter used.

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements
|DE10_LITE_D8M_VIP	13338 (1)	8833 (12)	1372392	0	6
|FpsMonitor:uFps|	72 (72)	44 (44)	0	0	0
|Qsys:u0|	12967 (0)	8613 (0)	1372392	0	6
|EEE_IMGPROC:eee_imgproc_0|	3902 (1634)	777 (691)	8192	0	0
|Qsys_alt_vip_cl_2dfir_0:alt_vip_cl_2dfir_0|	1283 (0)	1700 (0)	30768	0	0
|Qsys_alt_vip_vfb_0:alt_vip_vfb_0|	1552 (203)	1516 (109)	72192	0	0
|Qsys_altpll_0:altpll_0|	7 (6)	6 (2)	0	0	0
|Qsys_jtag_uart:jtag_uart|	140 (37)	113 (13)	1024	0	0
|Qsys_key:key|	2 (2)	2 (2)	0	0	0
|Qsys_led:led|	12 (12)	10 (10)	0	0	0
|Qsys_mipi_pwdn_n:mipi_pwdn_n|	5 (5)	1 (1)	0	0	0
|Qsys_mipi_pwdn_n:mipi_reset_n|	3 (3)	1 (1)	0	0	0
|Qsys_mm_interconnect_0:mm_interconnect_0|	744 (0)	611 (0)	0	0	0
|Qsys_mm_interconnect_1:mm_interconnect_1|	721 (0)	532 (0)	0	0	0
|Qsys_nios2_gen2:nios2_gen2|	2107 (33)	1606 (39)	62720	0	6
|Qsys_onchip_memory2_0:onchip_memory2_0|	69 (1)	2 (0)	1048576	0	0
|Qsys_sdram:sdram|	267 (217)	242 (154)	0	0	0
|Qsys_sw:sw|	10 (10)	10 (10)	0	0	0
|Qsys_timer:timer|	131 (131)	120 (120)	0	0	0
|Qsys_uart_0:uart_0|	122 (0)	92 (0)	0	0	0
|TERASIC_AUTO_FOCUS:terasic_auto_focus_0|	672 (220)	326 (113)	0	0	0
|TERASIC_CAMERA:terasic_camera_0|	367 (48)	401 (29)	129464	0	0
|alt_vipitc131_IS2Vid:alt_vip_itc_0|	315 (101)	262 (54)	19456	0	0
|altera_reset_controller:rst_controller_001|	0 (0)	3 (0)	0	0	0
|altera_reset_controller:rst_controller_002|	0 (0)	3 (0)	0	0	0
|altera_reset_controller:rst_controller_003|	6 (5)	16 (10)	0	0	0
|altera_reset_controller:rst_controller|	0 (0)	3 (0)	0	0	0
|i2c_opencores:i2c_opencores_camera|	257 (0)	129 (0)	0	0	0
|i2c_opencores:i2c_opencores_mipi|	273 (0)	129 (0)	0	0	0
|pzdyqx:nabboc|	134 (0)	73 (0)	0	0	0
|sld_hub:auto_hub|	164 (1)	91 (0)	0	0	0

Figure 7.1: Resource utilisation analysis of FPGA for vision submodule.

Figure 7.2 outlines the maximum clock frequency of the FPGA's components. The critical path delay of the vision subsystem is derived from the Qsys system which limits the maximum clock frequency of the system to 9.75 MHz.

Fmax	Restricted Fmax	Clock Name	Note
9.75 MHz	9.75 MHz	u0|altpll_0|sd1|pll7|clk[2]	
67.12 MHz	67.12 MHz	MAX10_CLK1_50	
87.94 MHz	87.94 MHz	MIPI_PIXEL_CLK	
96.0 MHz	96.0 MHz	u0|altpll_0|sd1|pll7|clk[3]	
107.62 MHz	107.62 MHz	altera_reserved_tck	
184.67 MHz	184.67 MHz	MAX10_CLK2_50	

Figure 7.2: Clock rate analysis for vision submodule.

The main code for the drive subsystem's Arduino takes 312 μs (accurate to 4 μs) for a single loop. This was quite high and enabled the system to track the rover's dynamics frequently which improved responsiveness and ability to control the rover's speed.

The time taken for the ESP32's main control loop was measured to be 1252 μs (accurate to 4 μs) neglecting the I/O communication delay from the UART and HTTP protocols. When taking these factors into account, the total control loop delay faced by the entire rover was approximately 3.5s (including the forced 1 second delay required for coherent UART communications). This calculation was done by measuring the frequency at which data was sent from the ESP32 to the server as seen in Figure 7.3, which displays the drive readings schema as an example.

←T→	id	PositionX	PositionY	Speed	Yaw	reading_time
Edit Copy Delete	46743	0	0	0	0	2021-06-15 17:00:28
Edit Copy Delete	46744	0	0	0	0	2021-06-15 17:00:30
Edit Copy Delete	46745	0	0	0	0	2021-06-15 17:00:33
Edit Copy Delete	46746	0	0	0	0	2021-06-15 17:00:36
Edit Copy Delete	46747	0	0	0	0	2021-06-15 17:00:40
Edit Copy Delete	46748	0	0	0	0	2021-06-15 17:00:45
Edit Copy Delete	46749	0	0	0	0	2021-06-15 17:00:49
Edit Copy Delete	46750	0	0	0	0	2021-06-15 17:00:54
Edit Copy Delete	46751	0	0	0	0	2021-06-15 17:00:59
Edit Copy Delete	46752	0	0	0	0	2021-06-15 17:01:02
Edit Copy Delete	46753	0	0	0	0	2021-06-15 17:01:04
Edit Copy Delete	46754	0	0	0	0	2021-06-15 17:01:09
Edit Copy Delete	46755	0	0	0	0	2021-06-15 17:01:12

Figure 7.3: Data sent from the rover the associated timestamp

The bandwidth of data which is being sent and received by the MySQL database, on an hourly basis when the rover is in use, is documented in Table 7.1

Table 7.1: Data flow statistics of the MySQL database.

Traffic type	MiB per Hour
Sent	12.9
Received	153.8
Total	166.7

Since most of the data flowing into the database is from the ESP32, we can see that the ESP32 sends a

large amount of data to the server per second. However this is variant on the positioning on the obstacles. If more obstacles are detected, it would POST more records to the database which would greatly affect amount of data transferred.

7.2 Optimisations

7.2.1 Obstacle Mapping Computation

The Mars Rover has been optimised to minimise computational costs and to make efficient use of the limited resources. The first notable optimisation step was applied for the vision module, where the approximate computation of the location of the balls detected were computed on the control module rather than on the FPGA. This is largely due to division operation which is not synthesisable on Verilog as well as using mathematical functions such as atan2() for trigonometric calculations. Moreover, by performing the calculation on the control module, which links all other modules together, it was advantageous in that the rover system was able to implement a warning feature that is triggered when the rover was in the vicinity of a ball on its vision, which will then interrupt the queued commands and execute a sequence of drive commands to avoid the ball. This can be done all in one module, on control, without having to receive warning signals from vision but just using the computed results straight from the control module itself.

7.2.2 Command Decomposition

As mentioned in section 5.5.6, the ESP32 breaks down high-level commands to low-level commands to be parsed and performed by the drive subsystem. This design choice was implemented to simplify the control flow within drive as well as make the subsystem more efficient and increase its operating frequency. This is a key requirement as the operating frequency of the drive subsystem would need to be as high as possible to ensure low-latency tracking of the rover's dynamics especially in real-time.

7.2.3 Asynchronous Communication between Control-Command

The HTTP communication between control to command is done asynchronously. This means the rover can receive and send commands whenever required rather than wait for the server to request data. This significantly increases data throughput between the server and the rover.

Section 8

Intellectual Property

An overview of the major concepts on intellectual property (IP), both registered and unregistered IP, in the engineering context were presented to the student cohort. The main types of IP covered were copyrights to protect original expression, trademarks to protect 'Badge of Origin', designs to protect aesthetics and patents to protect the functionality aspect of inventions. In addition, the legal aspects of procedure and timeline ranging from application to confirmation of acceptance were outlined with prominent focus on the European Patent Applications (EPO) and Patent Cooperation Treaty (PCT).

In the context of this design project, it is relevant to consider the sections and aspects of the rover design and implementation which could be protected as IP as it is a relatively complex system with multidisciplinary subsystems. One part of the rover that can be protected by IP is the EEE_IMGPROC.v image processor block from the vision module for the obstacle detection algorithms just like the IP cores in Intel Quartus software, assuming that the file was written from scratch and neglecting the fact that the original file was cloned from an open source GitHub repository. Another candidate for a potential consideration is the web design and development of the command sub-system, where in particular the front-end webpage design which in terms of graphical interface is unique to each group and no prior specific design templates are provided. Moreover, the back-end web development is implemented using established scripting languages, frameworks and libraries and so uniqueness is largely restrictive in the back-end.

Overall, the entire software source code for the project on the GitHub repository is licensed using the GNU General Public License v3.0.

Section 9

Conclusion

In conclusion, the primary objectives and all the intended functional and non-functional design and implementation requirements are met by all the subsystems and the Mars Rover system behaves according to the requirements. This is evident in the final demonstration where the rover drive system responds accurately to motion commands issued remotely by the web application dashboard of the command subsystem and relayed via the ESP32 control subsystem, with an average delay of 3.5 seconds and executing the commands in the right order. Moreover,the web dashboard accurately displays the graphical charts of the latest rover position and maps the obstacle locations from the vision subsystem via the ESP32 control subsystem onto the terrain map, as well as displaying the energy status as battery life – all with an average latency of five seconds.

To conclude, the performance of the system met the required standards as seen in section 7, and the rover was fully functional and satisfied all the required criteria.

9.1 Extensions

In terms of enhancing the rover's operational functionality, designing a vision algorithm in the vision subsystem that can work irrespective of environment would allow a more accurate and robust detection process with regards to unknown terrain.

To enhance client-side features and user experience, the front-end could also be extended to include more graphics to display other aspects rover's external environment besides obstacles and include the ya arrow on the map. Moreover, elevation can be measured using vision subsystem and a three-dimensional (3D) graph to monitor the rover's position relative to the obstacles position simultaneously all on the same map. Furthermore, authentication a nd the security between rover-server communication could be improved by using the HTTPS protocol or adding SSL to the existing HTTP protocol.

The rover's responsiveness could be improved by exploiting the dual-core structure of the ESP32 to multi-thread operations to reduce the control-loop delay. Additionally, a more efficient path finding algorithm could be implemented to navigate around obstacles without covering too much distance to preserve the rover's battery life. An alternative to preserving the rover's energy would be to adjust speed depending on energy status, such that the rover will be required move slower to conserve energy if the current battery status is below a specified threshold.

Bibliography

[1] Espressif System, "ESP32 Series Datasheet", Aug. 2016 [Revised 2021]

[2] Terasic Technologies Inc., "D8M-GPIO Datasheet", P0447 Datasheet, Sep. 2016

[3] Avago Technologies, "ADNS-3080 Datasheet", 2005

[4] W3schools.com. 2021. jQuery Introduction. [online] Available at: https://www.w3schools.com/jquery/jquery_intro.asp [Accessed 15 May 2021].

[5] www.javatpoint.com. 2021. Node.js vs PHP - javaTpoint. [online] Available at: https://www.javatpoint.com/nodejs-vs-php [Accessed 13 May 2021].

[6] www.darkpgmr.tistory.com 2013. Video Recognition and Colour Spaces (Gray,RGB,HSV,YCbCr). [online] Available at: https://darkpgmr.tistory.com/66 [Accessed 21 May 2021].

[7] n.d. RGB to HSV color conversion [online] Available at: https://www.rapidtables.com/convert/color/rgb-to-hsv.html [Accessed 20 May 2021].

[8] Rosebrock, A., 2015. Find distance from camera to object using Python and OpenCV. [online] PyImageSearch. Available at: https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/ [Accessed 19 May 2021].

[9] "ESP32 HTTP GET and HTTP POST with Arduino IDE — Random Nerd Tutorials", Randomnerdtutorials.com, 2021. [Online]. Available: https://randomnerdtutorials.com/esp32-http-get-post-arduino/. [Accessed 18 May 2021].

[10] "ESP32/ESP8266 Insert Data into MySQL Database — Random Nerd Tutorials", Randomnerdtutorials.com, 2021. [Online]. Available: https://randomnerdtutorials.com/esp32-esp8266-mysql-database-php/. [Accessed: 15 May 2021]