Improvised Exploring Device

Year 2 Group Project

Department of Electrical and Electronic Engineering
Imperial College London
June 2023

Hasnat Chowdhury
Harry Griffiths
Kishok Sivakumaran
Michael Yau
Ratul Shek
Zeyd Chamsi-Pasha

Contents

1 Abstract

A balancing robot is required to navigate a maze autonomously. The arena is coloured black and surrounded by black curtains. White LED strips are laid out on the arena to create a maze. The task given is to design a segway which can balance on 2 wheels and autonomously navigate the maze without colliding with the white LED strips. As it traverses the maze, a map should be built, with the current position and shortest path through the maze being displayed.

This project was broken down into several key components and tackled in subgroups decided upon as a team:

- **Web application** – A backend which receives telemetry data from the rover and produces a map of the maze given data and a frontend which displays all of these for the user to see clearly.
- **Control system** – To balance the rover and drive the motors for a given movement.
- **Power delivery system** – To power the LED beacons with sufficient brightness and constant power for beacon detection.
- **Computer vision** – Detects beacons and uses triangulation to accurately determine position on the mazeDetects beacons and uses triangulation to accurately determine position on the maze.
- **Navigation and routing** – Control the path of the rover ensuring it does not collide with walls and navigates through the maze.
- **Integration** – Joint effort by group to ensure that all the systems developed independently interact with each other smoothly for finished product.

These key components were designed with scalability and adaptability in mind so that while they were developed independently, they could be connected to complete the design. For example, the web API can handle multiple rovers with different IDs.

2 Specification

There were several guidelines and requirements when designing the autonomous balancing robot. The system must:

1. Autonomously move through a maze without crossing an illuminated line.
2. Autonomously survey the layout of the maze and produce a map of the discovered layout, overlaid with the position of the robot and the shortest path through the maze.
3. Balance on two wheels.

The autonomous balancing robot is assessed in the following ways. The system must be:

1. Reliable and able to complete its task without human intervention.
2. Robustly and efficiently constructed.
3. Coded for:
    a. Usability
    b. Testability
    c. Maintainability
    d. Scalability

Restrictions are also imposed on the project in the following ways. The permitted means of locating the robot within the maze are:

1. Dead-reckoning based on accelerometer, gyroscope and wheel revolution counting.
2. Optical detection of the maze markings and up to three illuminated beacons by the robot.
3. Illuminated beacons shall be powered by a specified PV-array emulator and provided energy conversion modules.
4. The robot will use the provided FPGA-based camera system

3 Resources

To start off the project, various components were provided and some were given as time progress. Below is the record of the components given. (Please type descriptions of each of the components)

3.1 FPGA board (FPGA Max DE-10 Lite)

The DE-10 Lite is built around the Altera MAX10 FPGA and features on-board SRAM and flash memory allowing for the FPGA to be blasted in-situ. The SRAM will be used as a frame buffer in this project and the on-board flash will store the software.

3.2 Camera module (D8M)

The D8M is an 8MP camera designed for the DE-10 Lite and uses the MIPI protocol to communicate. However, due to the limitations of our processing we are running at a resolution of 640*480. The camera is connected to the FPGA via a 40-pin ribbon cable.

3.3 WiFi microcontroller (ESP32)

A microcontroller by Espressif Systems with 2 Xtensa cores running at 240Mhz. The built in WiFi and Bluetooth radio allows for the device to access the internet and communicate with nearby devices. It provides many GPIO pins which allow for hardware expansion through support for I2C, UART, SPI and other communication protocols.

3.4 Inertial measurement unit (MPU6050)

It is a module with an accelerometer and a gyroscope. It helps to measure motion-related parameters. It provides accurate and very sensitive telemetry values which are perfect for a reliable control system.

3.5 Stepper motors and drivers (NEMA-17 Stepper motors and A4988 Stepper Motor Drivers)

The motors and drivers are the components that drive the segway to move. These were an ideal choice as they were built for precision and provide excellent torque characteristics at low speeds. However, they constantly draw maximum current causing motor and driver circuits to build more heat. Therefore, if power or efficiency were a concern, these components wouldn't be as suitable.

4 Project Management

4.1 Milestones

Several milestones were met in the development of the project, the progression through these milestones were used to keep track of the progress of the project and to identify areas that needed more attention. The progress was projected with a Gantt chart.

The key milestones that were achieved are as follows:

- The completion and testing of the back-end API.
- The completion and integration of the front-end interface.
- The integration of the MPU6050 with the ESP32 and to the back-end API.
- Expanding the vision system to locate other colours of beacon.
- A communication channel between the FPGA and the ESP32.
- Motor driver and control system integration.
- The completion of the power delivery system.

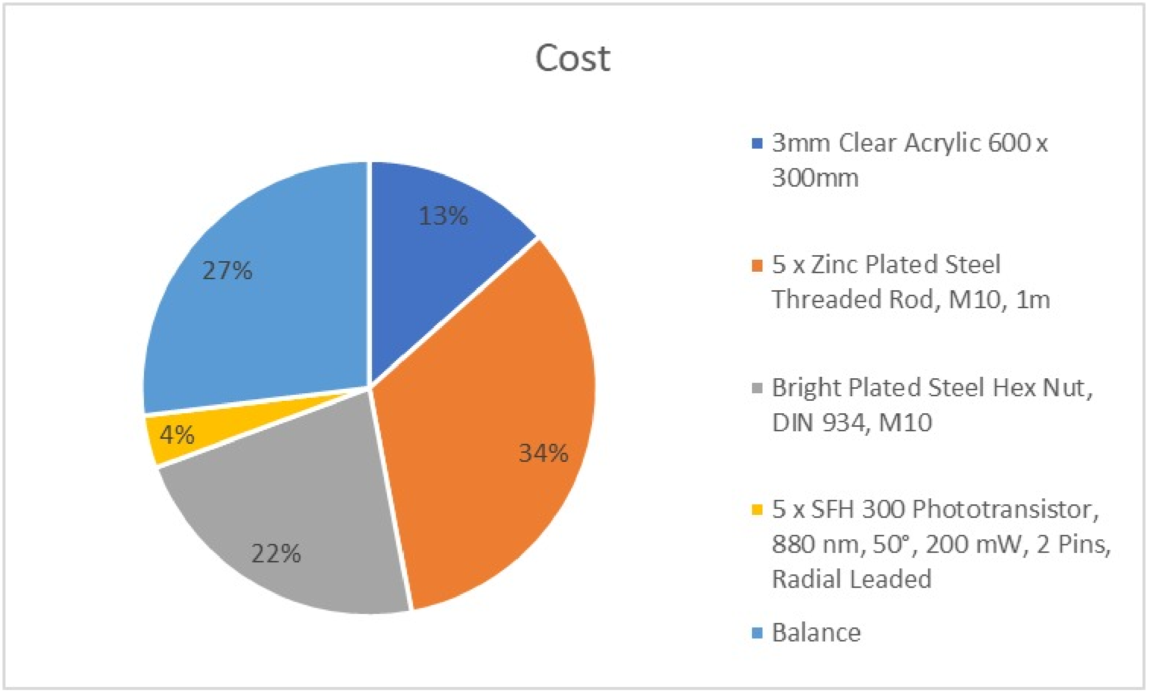
4.2 Meetings and Organisation

Team meetings were held on Microsoft Teams to keep each other up to date. In these meetings the progress on each subsystem were discussed and then an overall discussion on the integration of the subsystems.

4.3 Financial Costs

The majority of costs came from building our chassis. This included the funding for the clear acrylic itself costing £8.07, threaded M10 rods costing £20.14, M10 nuts costing £13.38 and five phototransistors for £2.25.

Overall, in total we had spent £43.84. All items were carefully evaluated before purchase ensuring that all items were necessary and used. The costs could have been more than halved if we were able to make orders of specific quantities instead of having to order a minimum amount. To prevent waste and encourage growth among our peers, we gifted any spare items we had.



5 Web Application

5.1 Frontend

The web interface is built around the Angular Framework and is programmed in TypeScript and HTML. Angular is designed around components and services. The components consist of HTML and CSS to define how the component is rendered and typescript class which defines the behaviour of the component. A component can be instantiated in other components to build the user facing webpage. Services provide global functionality and a service can be injected into any given component providing access to the properties and methods of the service. Using Angular directives components can be dynamically generated depending on the data as defined by their class. For example, the *ngFor directive will duplicate a template of HTML for each iteration of the for loop.

Overview

The web interface is built around the Angular Framework and is programmed in TypeScript and HTML. The back end server hosts the web interface and the rover is controlled through the API that the backend exposes.

Features

The webpage consists of 3 tabs, a dashboard, a console log and a settings panel. The dashboard is broken into 4 sub panels.

- Map
- 3D Viewport
- Telemetry
- Control Panel

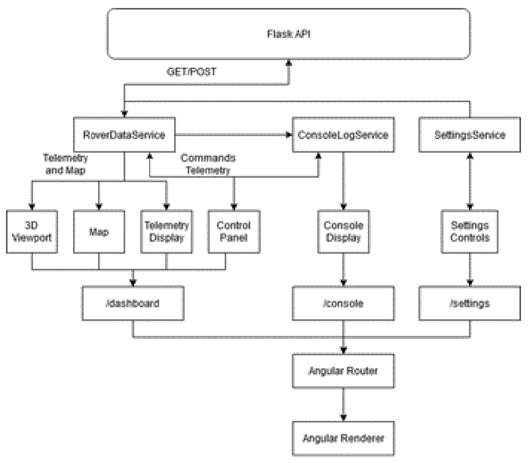


Figure 1: Frontend design overview

There are 3 services running in the background which maintain the information that is rendered. The rover data service maintains the connection to the server, the console logging service provides interfaces for adding to the log and events for when a new entry is added and the settings service maintains a global settings object which can be modified by a component and also provides an event for when that has happened. The Angular router is a system to dynamically update the DOM based on the URL in the client browser without hosting separate web pages on the server. This enables the production of single page applications (SPAs) and can dynamically load components from the server based on the URL in the browser without changing the URL from the server's perspective. [1]

Map Display

The map display is a 2D representation of the segway's internal map. The map is updated in real time as the segway explores the environment and identifies features. The map is rendered using Two.js as it is a simple and lightweight library that allows for easy rendering of shapes. It is designed for modern browsers and can render to multiple systems, WebGL, Canvas or by drawing native SVG shapes. [2] The map is rendered as a series of lines representing the walls and connections between nodes on of the maze

3D Viewport

The 3D viewport displays a 3D model of the segway that is updated with the tilt and orientation of the segway from the telemetry service. It is rendered with Three.js, again as it is an easy to use library that is relatively lightweight. [3]

Telemetry Display

The telemetry panel displays the current state of the rover. This includes the current position, orientation, state, and a feed from the sensors

Control Panel

The control panel is a set of buttons that allow the user to send commands to the segway manually. As the segway is autonomous this is a START/STOP button that allows the user to start and stop the segway's autonomous exploration. There is also a toggle to enable auto-updating of the telemetry and map data.

Console Display

The console displays a log of events, warnings and errors that are generated by both the front-end and the segway itself. These are stored in the ConsoleLogService. This is useful for debugging and monitoring the segway's state.

Settings Display

The settings display provides a simple interface to modify an internal state of settings.

Rover Data Service

The rover data service uses the Angular http client to make GET requests to the server. The responses from the server are instantiated as an interface. The design of the interfaces matches the data returned from the API. The service has an interval timer which calls an update function every 200ms by default which constantly overwrites the internal data store. When this happens an EventEmitter is triggered which a component can subscribe to so that it is informed when the latest telemetry is available. A similar process is followed for the map data. An example of one of the interfaces is shown in figure 2.

```
export interface Telemetry{
    id:number;
    position:Position;
    accelerometer:xyzValue;
    gyroscope:xyzValue;
    steps:number,
    state:"start"|"stop"|"mapping"|"solving"|"error"
}
```

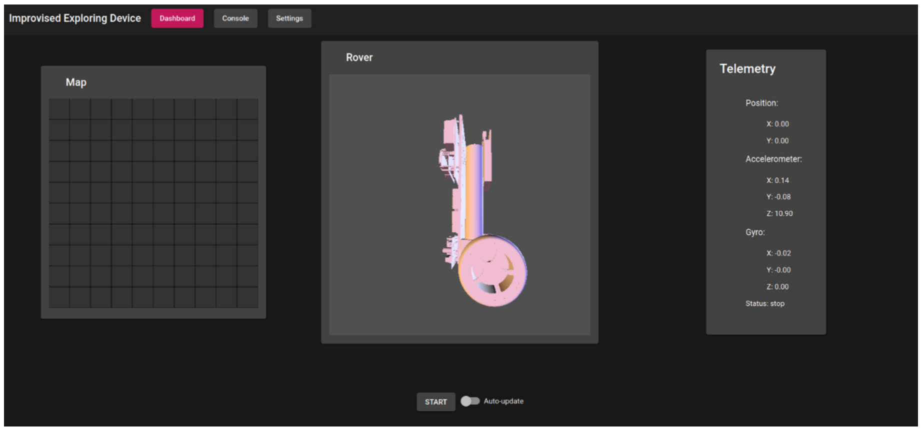
Figure 2: The telemetry interface

Implementation

Each panel is implemented as a separate component in Angular. The components are then combined into a single page application using the Angular router. The telemetry is backed by a service that polls the server and maintains an internal telemetry state, matching with the latest telemetry sent from the rover. The console is backed by another service that stores the log events and provides EventEmitters for new log events.

Design

The design of the interface is based on the Material Design specification. This is a design language developed by Google. The components are provided by the Angular Material package which provides a set of components such as buttons, input panels and menu elements. All of which follow the Material Design specification. [4]



Telemetry

Position:

X: 0.00

Y: 0.00

Accelerometer:

X: 0.14

Y: -0.08

Z: 10.90

Gyro:

X: -0.02

Y: -0.00

Z: 0.00

Status: stop

Figure 3: The dashboard interface

Testing

Angular provides a unit testing system using spec files to define the intended behaviour of a given component. When running then ng test command each component is instantiated and properties of the component is compared against the specification defined in the .spec file.

Further Goals

The implementation of an authentication system would have been ideal as currently any client can access the information of any segway and there is no verification that the information sent form the segway is truly from the device you are using. This could be done using authentication tokens handed out by the server. When a segway is onboarded to the server it responds with a randomly generated token which will be sent over HTTPS. A similar process for the client, this way each API call must have the associated token to be accepted. An account-based log-in system could authenticate the client and give them control over a limited number of segways for example.

5.2 Backend

Flask

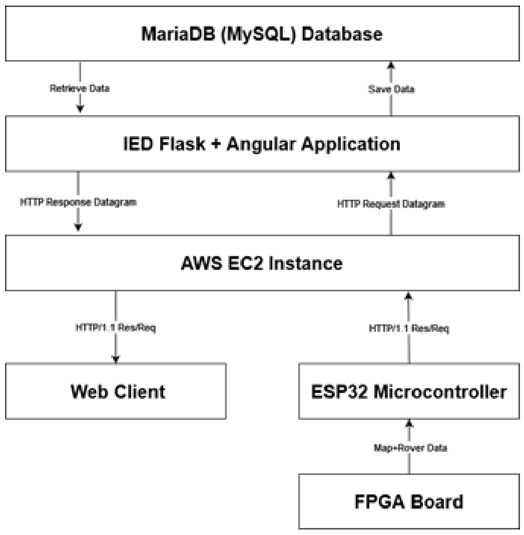


Figure 4: Backend design overview

The backend of the web application is based on Flask Web Framework. Flask is a lightweight framework used to build web applications using python [5]. Flask acts as the back-bone to the whole Tech stack providing the interfacing and connection between the frontend Angular application, The AWS EC2 instance and the MariaDB database.

The choice of Flask was made primarily due to personal preference and due to prior experience with the Python programming language, Flask's main advantage is its minimalistic design which allows for modularity and also, as it's written in Python, it enjoys the flexibility and extended support that the programming language allows for. The main reason why Flask was chosen instead of other technologies such as, JavaScript and Node.js is because of a steeper learning curve required in being able to build an entire web application [6] and it being more suited for a larger scale environment that is beyond the scope of this project.

The way the web application is designed out is thus: an AWS EC2 instance which holds the full stack application, the frontend in Angular and the backend in Flask which contains the ORM model of the database using a library called SQL-Alchemy, which allows for interfacing with MariaDB, a database based on the MySQL engine. This entire application is hosted on the AWS server, therefore the outside world will not be able to reach it normally. The application uses Gunicorn and NGINX to deal with that. Gunicorn is a Web Server Gateway Interface (WSGI) [7] which allows for communication between the main Flask application and HTTP requests, it allows for Python to be able to understand and execute HTTP requests/responses, this is then paired with NGINX which is a web server that acts as a reverse proxy [8] that allows requests sent to the AWS' IP-address to be rerouted to your flask application. Using these technologies enable the linking of the web server and the web application including features that allow for reliability, maintainability, and scalability due to the services built-in both gunicorn and nginx. These services are what allow seamless integration of the RESTful API used and interfacing with the MariaDB instance.

Database

One of the functional requirements of the whole system is that the segway should store data about the map and about itself so that it can then be accessed, managed, and later retrieved. To do that a database is used for the project. MariaDB, a relational database running on the MySQL engine, was chosen as the database for the design. The choice was made by taking into consideration factors such as development time constraints, size of database and prior experience of the development team; MariaDB is chosen over vanilla MySQL due to its greater performance in speed when executing queries.

The Relational Model consists of two primary entities, the Map, and the Segway. The Map is based on a typical graph data structure which uses vertices, “Nodes” and non-directional weighted edges. This implementation is used due to the requirement of being able to then calculate the shortest path and using a graph data structure allows us to then perform some sort of shortest path algorithm such as Dijkstra’s. The Segway entity’s main role is in providing the telemetry data to the front-end application, this includes data on the position of the segway, the accelerometer and gyroscope data and also data on the steps and date of the segway. This data is crucial in being able to locate the segway relative to the map and also provide the position of the nodes that make up the map, it also serves to assign the weights to each respective edge.

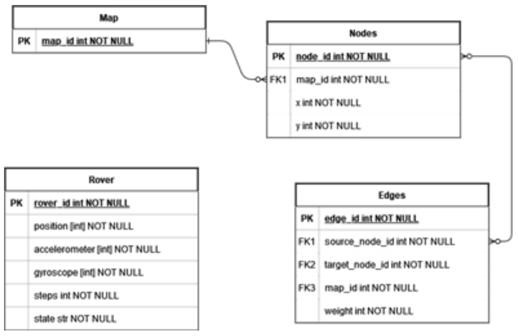


Figure 5: Map relational model

Taking a closer look at the Map model, it will have an id which will then be used as a foreign key on the Node table, this establishes one to many relationship between the Map entity and the node entity, each map can have multiple nodes but not the other way round. In turn the Node table has its own identifier, values for the x and y coordinates and also the foreign key from the Map table. The Edge entity has multiple foreign keys, namely the ids of the source and target nodes, its own respective id and also a weighting parameter to keep track of the weight assigned to that particular edge, the Edge entity has a many to many relationship with the Node table as nodes can have multiple edges and edges attach to multiple nodes, the source and target node. It is important to assert these cardinality relationships as its what establishes the connections of the map and overall content that can then be retrieved by the server when either transmitting navigation commands or calculating the shortest path between the start and end nodes.

REST API

An API can be simply described as a set of protocols to be used as a universal method of communication between a client and a server [9]. The API is an essential part of the Web server as it acts as a middleman between the web application and the segway it provides a way to organise and share resources and information whilst also providing security control and authentication. The main purpose of the API is to be able to display the front-end application when one enters the IP address of the AWS EC2 instance but also it is used as a way to perform CRUD operation. One example of this could be in the form of a POST request to send information to be stored to the database and conversely a GET request would return information particular to that request. One example of this would be sending the Telemetry data of the Segway, on the client side (the actual segway) data would be transmitted about he segway’s position, mpu6050 data and other necessary data, this would be done in the ESP32 microcontroller through a HTTP request, this request would contain in its body a JSON format file, the choice of using json was made due to its versatility and almost universal usage across various platforms. For this specific use case there are currently seven routing endpoints in the API used. Three each for both the map and segway entities, and one for loading the index.html page. Taking a closer look at Telemetry for example when sending a POST request to update the segway's data, the microcontroller would send a HTTP request to the server containing the necessary data, the server would receive it and route it to the API which would then perform the necessary functions and CRUD operation on the database.

6 Control System

A well-designed control system is integral for the segway to balance, compensate for any disturbances, and achieve precise control by continuously monitoring quantitative specifications such as its tilt and adjusting the motor output to maintain stability. This will all need to be integrated with the rest of the subsystems as it has the potential to act as the foundation for the segway. Control systems are typically incorporated with a feedback loop, such as a PID (Proportional-Integral-Derivative) system, to dynamically respond to deviations and ensure stable operation.

6.1 System Requirements and Constraints

When it comes to thinking about system requirements, it's important to be mindful of things like operating voltage / power consumption, accuracy of the controller determined by properties such as weight, weight distribution, torque of the motors and inertia of the entire design.

One of the challenges faced was using the correct pins from the ESP32 as they had different specificities (referring to GPIO pins). Furthermore, being limited via the power of the battery meant that the speed and output power of the motors are limited to allow for enough voltage and current to all the other components. One of the ways the effect of this was minimised, was by adjusting the potentiometer values on the A4998 motor drivers to allow for a high voltage without compromising too much on current.

6.2 Design Approach

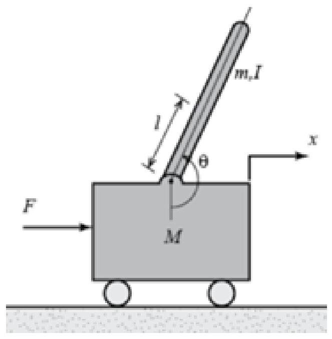


Figure 6: Inverted Pendulum

The concept of a self-balancing Segway is derived heavily from the theory of an inverted pendulum. The primary aim of the control system is to balance this inverted pendulum to keep it in an upright pose by applying a force to the base of the segway, via torque produced by the wheels. This is clearly visualised below.

The inverted pendulum system was stabilised using a proportional-integral-derivative (PID) control approach. A response proportionate to the discrepancy between the desired and actual positions of the pendulum is provided by the proportional term. While the derivative term projects future errors based on the error's rate of change, the integral term integrates the error through time to eliminate steady-state errors. The response of the control system can be enhanced by varying the weights given to each component otherwise known as tuning the PID values.

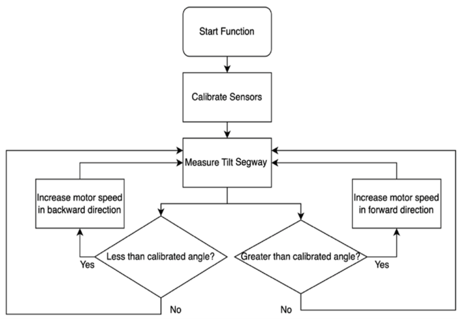

Start Function
Calibrate Sensors
Measure Tilt Segway
Increase motor speed in backward direction
Increase motor speed in forward direction
Less than calibrated angle?
Greater than calibrated angle?
Yes
Yes
No
No

Figure 7: Control flow diagram

Before diving into building a PID controller, the idea of just a P controller working would allow for easy implementation and design. This flowchart represents a controller that provides a simple reaction by increasing and decreasing the speed of the motor wheels to account for the changes in tilt. This was the original designed controller but proved to have many issues including a slow reaction speed to impulse responses, inaccuracies, and imprecise responses when subject to subtle perturbations.

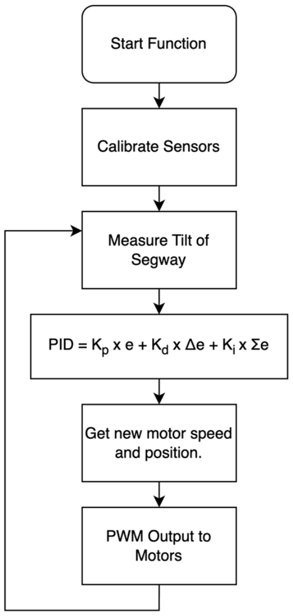

Start Function
Calibrate Sensors
Measure Tilt of Segway
PID = K_p x e + K_d x Δe + K_i x Σe
Get new motor speed and position.
PWM Output to Motors

Figure 8: PID controller flow diagram

This led to building a PID controller code to consider error. This will allow for improved stability, by eliminating steady state errors that compound over time, reducing overshoot, reducing latency, and increasing robustness of the system.

The code calculates the error by subtracting the target angle from the current angle. It then performs the PID control calculations, which involves multiplying the error by the proportional gain (Kp), the integral gain (Ki), and the derivative gain (Kd), respectively. The integral term and previous error are also updated accordingly.

Adjusting motor speeds based on the PID output, adjusts the speeds of the two stepper motors. If the output is above a certain threshold, the motors rotate in opposite directions to balance the pendulum. If the output is below the threshold, the motors stop.

The calibrateMPU6050() function collects calibration data from the accelerometer and calculates calibration offsets for each axis. These offsets are then subtracted from the raw accelerometer readings to obtain accurate tilt angles.

This in turn would theoretically make the control system self-correcting and thus successfully balance the segway.

6.3 Simulation

With this design in mind, it would help to build and test a control system using Simulink through MATLAB for further testing and understanding.

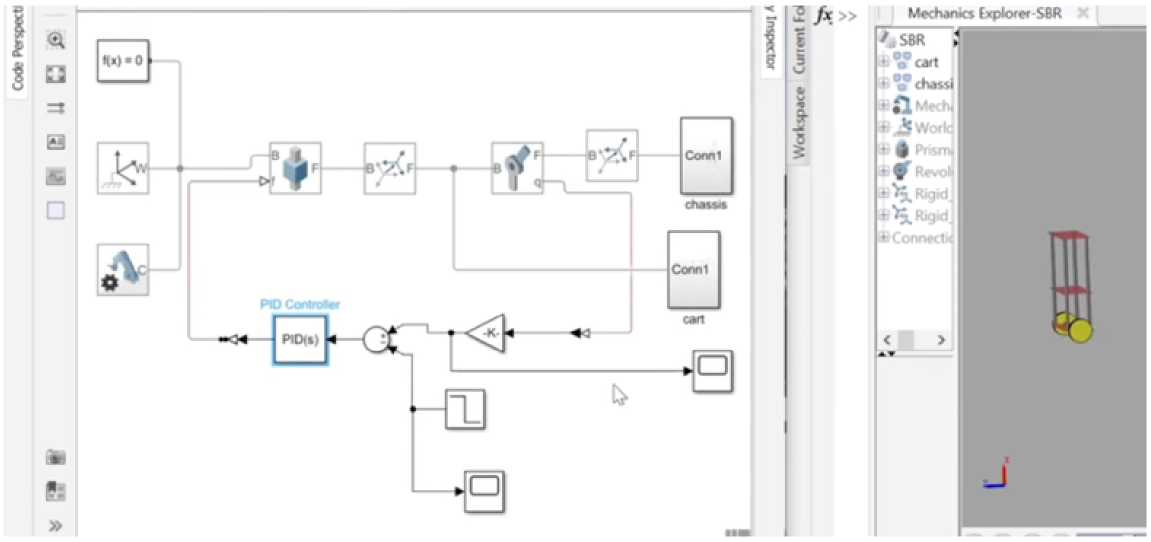


Figure 9: Simulink model of PID controller

Using this it is easy to simulate what effect, different values of PID will have on the system. The desired goal is to reach a system that has very small oscillations and is ultimately asymptotically stable as demonstrated in figure 10.

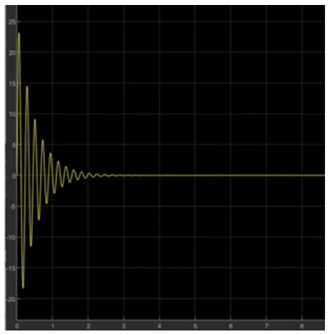


Figure 10: Simulink simulation result

By experimenting with the simulation, it became apparent that increasing the P, I, or D terms has the following effects on the overall system:

Increasing the proportional gain (P) amplifies the response of the controller to the error signal. In essence, it makes the controller more sensitive, leading to a stronger corrective action to reduce the error, faster initial response, and potentially higher overshoot.

Increasing the integral gain (I) increases the contribution of accumulated error over time to the control signal, eliminating steady-state errors by continuously adjusting the control output. This would allow the controller to better handle constant or slowly changing errors, improving the system's ability to track and maintain the desired value. However, too high of a value can cause overshoot, oscillations, and instability if not moderated.

Increasing the derivative gain (D) amplifies the contribution of the rate of change of the error to the control signal. It helps the controller anticipate and react to changes in the error signal providing faster response to sudden changes in the error, again, reducing overshoot and damping oscillations.

To summarise, finding the right balance and tuning these gains is crucial for achieving stable and efficient control. To find the true PID values, it would be wise to keep varying the gain values on the finished rover.

Writing the code for it to balance proved to be trivial after installing all relevant libraries and building the simulation on Simulink. Initially, the primary goal of the control system is to get it to balance and that is what is found in the appendix.

6.4 Movement

To enable movement while keeping the inverted pendulum upright, one approach is to gradually accelerate the wheel in the desired direction. However, maintaining a constant speed without toppling the pendulum is challenging due to the need for precise torque when starting and stopping.

When the pendulum is tilted, a force acts on the wheels, requiring an opposing force for balance. To address this, a perturbation method can be employed. By temporarily adjusting the balancing angles obtained during calibration, forward or backward movement can be initiated, followed by a return to the balancing position. Continuous repetition or adjusting the delay allows for consistent movement.

For left and right motion, a function can be implemented to rotate the pendulum quickly. By carefully controlling the movement of both motors independent of the balancing axis, tilting can be minimized. The pendulum can then be returned to a balanced state.

Implementing these strategies enables movement while maintaining the upright position of the inverted pendulum. This is the next step to successfully implement this.

Sensor integration of the MPU6050 was limited to the nearest 0.5 to act as a filter ensuring the that the smallest perturbations, in which case, may just be noise has no effect on the motors. This allowed the segway to stay relatively still when balancing. After adjusting the potentiometer values on the A4998, it was tuned to allow them to draw the same the voltage and current. This will enhance the segway's stability property as it ensures that both motors will provide the same torque for forward and backward manoeuvrability.

Challenges were encountered during calibration, including finding the right balance between responsiveness and stability. Strategies such as trial and error, iterative testing, and observation of the system's behaviour were employed to overcome these challenges. The calibration process ensured that the control system parameters were optimized by minimizing oscillations, preventing overshoot, and maintaining a steady state.

The calibration results demonstrated the impact of the chosen gains and thresholds on the robot's balancing performance. Through careful adjustment, the control system achieved improved stability and accuracy. Evidence of calibration results, such as graphs showing the system's response to disturbances, can be presented to showcase the effectiveness of the calibration process. Regular monitoring and fine-tuning of the control system parameters ensure continued optimal performance and adaptability to varying environments.

6.5 Testing

The efficiency of the control system was tested by rigorous testing processes, demonstrating the robot's capacity to balance on two wheels. While limitations were discovered in response to rapid changes or external forces, these difficulties can be addressed through enhancements to the control algorithm and varying chassis height. Calibration was critical in optimising control system characteristics, with evidence showing that modified gains and thresholds had a direct impact on balancing performance. Iterative modifications to PID values and empirical testing were used to overcome problems during calibration. The table can be seen in figure 11 below.

Iteration	**kP**	**kI**	**kD**
1	1.00	0.00	0.00
2	1.50	0.01	0.00
3	1.50	0.00	0.01
4	1.50	0.20	0.20
5	1.80	0.25	0.20
6	1.90	0.25	0.35
7	2.10	0.47	0.30
8	2.50	0.50	0.40
9	2.80	0.45	0.45
10	3.00	0.57	0.50
11	4.50	0.43	0.35
12	4.80	0.45	0.30
13	4.90	0.80	0.20
14	4.98	0.81	0.26

Figure 11: PID Iterations

The inclusion of the control system will allow for the segway to balance freely and precisely on the terrain. However, to perfect the control system, it is important to lighten the load on the chassis to allow for a more suitable amount of torque, fully build in the movement functions as mentioned above, utilise both cores on the ESP32 board to ensure that there are no clock delays between the sensor and motor signals and use filters to prevent high frequency oscillations within the system.

7 Power System

In order for the segway to know its location in the maze and map it accordingly, it will need some reference points in the demo arena for it to recognize. This will be in the form of three LED beacons that may be placed anywhere around the arena.

7.1 Power Supply

The power supply of the whole DC grid is provided by solar energy. Throughout the project, PV panels are used as the power supply of the DC grid. However, since the actual run of the device will be in an undesirable environment for PV panels, a power supply unit will be used to emulate the performance of the PV panels.

To start off, the performance of the PV panels is tested. IV characteristics are plotted for PV panels in different configurations. The ratings of individual PV panels are given as 5V and 230mA.

The data in the figures are taken in a sunny day in a bright environment. The result is as expected. The IV characteristic exhibits a flat line at lower voltage and a steep drop and higher voltage. This also results in a PV graph that has a shape as expected. The power increases linearly at lower voltage and drops drastically at higher voltage. This leads to the maximum power point of the PV panels. The maximum output power of the PV panels is given at around 4.5V to 5V.

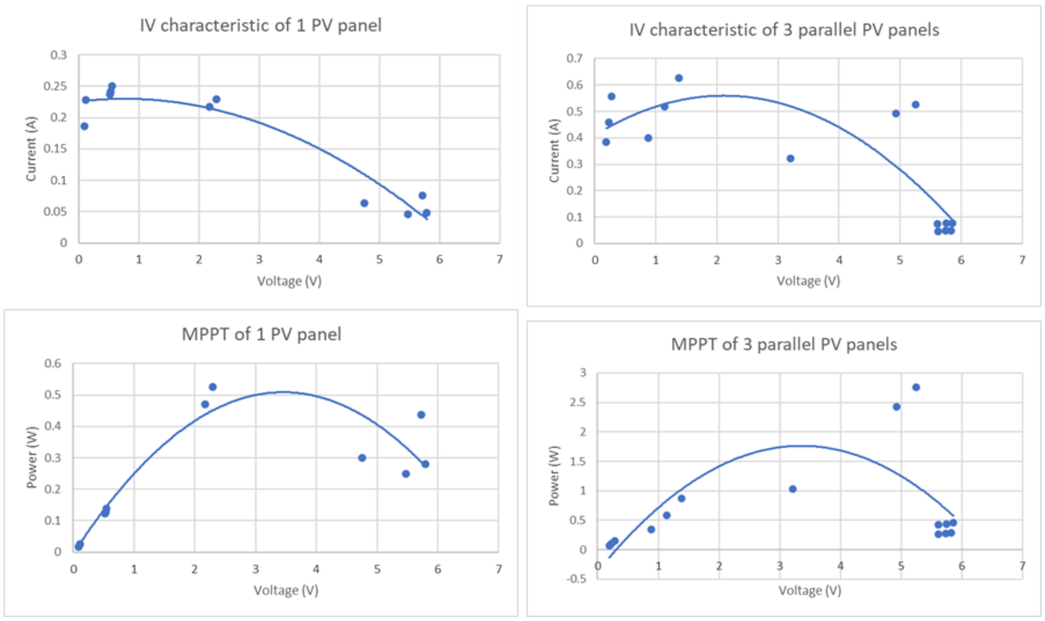


Figure 12: MPPT and IV characteristics of PV panels in different configurations

From figure 12 above, showing the IV characteristics of a single PV panel, has the expected trend for a maximum current of 0.24mA and drops at around 4V. When compared to figure which shows the IV characteristics of three PV panels in parallel, the shape of both graphs is similar. However, the parallel configuration produces a much higher output current, which leads to a higher output power. Higher output current is desired since quite a lot of current is needed to drive three LEDs in parallel. Series configuration has also been tested at the same day. However, it is not ideal to use it although it gives a much higher voltage. Its output current is very unreliable and has a large fluctuation of output voltage. If anyone of the PV panels is half covered, the output voltage and current drops significantly, which is undesirable as a power supply.

The PV panels are also connected to a boost switch mode power supply (SMPS) for further testing to simulate the real demonstration. Three parallel PV panels are connected to a boost SMPS and a 75Ω load, or a 10Ω for lower voltages, to create an IV characteristic for the PV panels with a boost SMPS connected and a load. The result are shown in figure 13.

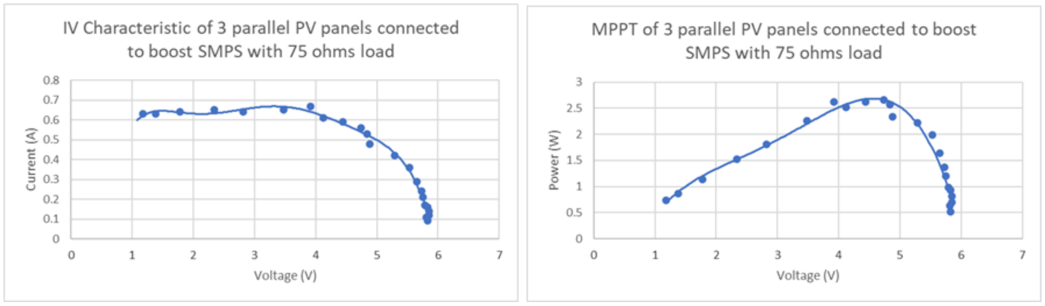


Figure 13

The results are as expected. The IV characteristic shows a plateau at lower voltages with around 0.65 mA and has a significant drop at larger than 5V. This results in an expected graph of power, which shows that the maximum output power would be around 4V to 5V.

7.2 Maximum Power Point Tracking

```
power1 = vb * iL;
power_diff = power1 - power0;
power0 = power1;
vb_diff = vb - vb_old;
vb_old = vb;
if (power_diff > 0){
    if(vb_diff > 0){
        open_loop=open_loop-0.001;
    }
    else{
        open_loop=open_loop+0.001;
    }
}
else{
    if(vb_diff > 0){
        open_loop=open_loop+0.001;
    }
    else{
        open_loop=open_loop-0.001;
    }
}
```

Figure 14

Maximum power point tracking (MPPT) is used mainly to maximize the power output from the PV panels by adjusting the operating point. The algorithm continuously monitors the output voltage and current and adjusts the operating point. This ensures the system to harvest the maximum power from the PV panels for most of the conditions. This will negate small oscillations coming from the current supply due to weather conditions such as a cloud blocking the sun. The algorithm is implemented in a perturb and observe method **ref:mppt**. The code implemented is shown in figure 14. The code gets the current value and past value of voltage and current and compares them. When the power and voltage are both larger, it means that it has not reached it's maximum power, and duty cycle is further increased. In the code, the duty cycle is decreased. This is correct since the PMOS is the boost SMPS reverses the operation. The algorithm ensures that the power in the DC grid will be as high as possible at any given time, given a steady brightness of the LED beacons as much as possible.

7.3 LED

The main component of the energy system are the three LED beacons, which will be essential to help the robot to navigate through the maze. There are three different colours of LED, red, blue and yellow. The exact ratings of the LEDs are not given except for 1W. Preliminary testing is needed to find out the diode drop, maximum voltage and current of each colour of the LEDs. Initially the LED beacons are run from a power supply unit at a voltage of 2V and the current limit to approximately 50mA. After that, the voltage limit is brought up until the voltage will rise to a fixed value. That value is the voltage drop of the diode. After that, the current is brought up such that the diode consumes 0.9W. That will be the expected maximum brightness of the LED. The voltage drop at lower current levels are also plotted for reference. The maximum operating point is purposedly tested at 0.9W. This prevents unexpected situations that will break the LEDs. Data is taken from the display of the power supply unit. The experimental results are in figure 15.

	Yellow	Blue	Red
Diode Voltage (V)	1.95	2.69	1.95
Maximum Voltage (V)	2.32	3.09	2.41
Maximum Current (A)	0.427	0.311	0.376

Figure 15

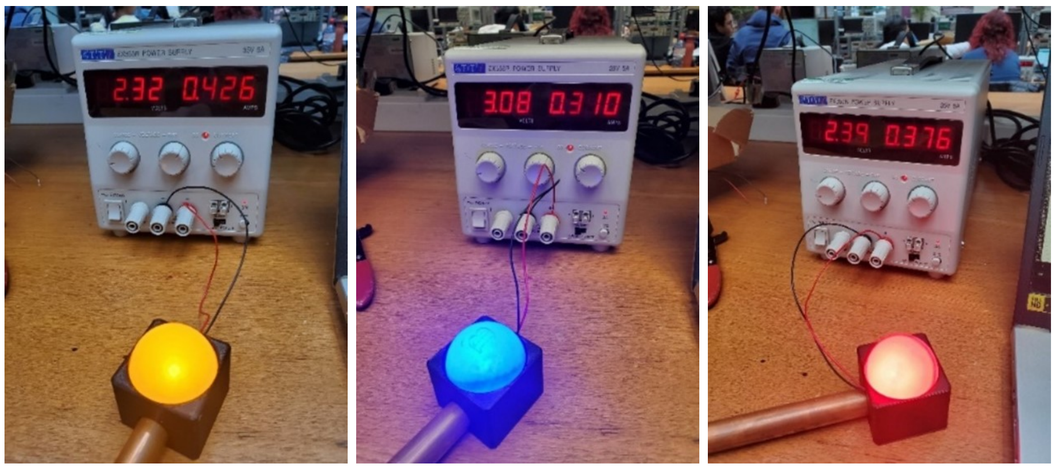


Figure 16: LED Beacons

After the initial testing of the LEDs, each of them is connected to a separate LED driver, which is a buck SMPS. It controls how much current can flow through into the LED and prevents it from breaking. For better and stable control of the output from the LED driver, closed-loop buck SMPS is chosen for the LED driver. The advantage of a closed-loop controller is that the output voltage is regulated at a desired value by our command and it does not vary with output current when compared to open-loop buck SMPS.

Proportional-integral-derivative (PID) controller is used in the design. In the implementation of the closed-loop buck SMPS, a PID loop in created within a PID control loop. The inner loop controls the SMPS output current, a reference current is provided by the current sensing resistor at the output side of the LED driver. The current reference is generated by the outer voltage control loop. It uses the output voltage to calculate a voltage error and then demands a current from the inner loop. This results in a good voltage regulation and a high efficiency across a range of output current. Figure 17 is an excerpt from the controller code of the PID function for controlling the voltage.

```
def pidv(pid_input):
    global e0v, e1v, e2v, u0v, u1v, delta_uv
    e_integration = e0v
    if u1v >= uv_max or u1v <= uv_min:
        e_integration = 0

    delta_uv =  (kpv * (e0v - e1v) + kiv * Ts *
                e_integration + kdv / Ts * (e0v - 2 * e1v + e2v)
                )
    u0v = u1v + delta_uv

    saturation(u0v, uv_max, uv_min)

    u1v = u0v
    e2v = e1v
    e1v = e0v

    return u0v
```

Figure 17: PID control loop for voltage control

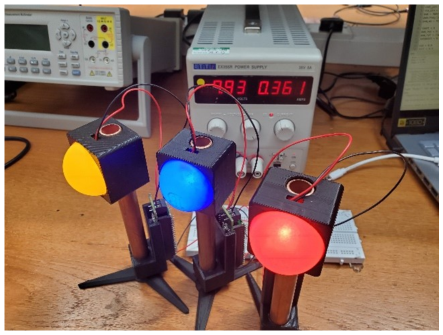
```
def saturation(sat_input, uplim, lowlim):
    if sat_input > uplim:
        sat_input = uplim
    elif sat_input < lowlim:
        sat_input = lowlim

    return sat_input
```

Figure 18: Saturation function

A saturation function is also introduced in the code to prevent overshoot. This is used mainly to limit the duty cycle so that it does not fall below 0 or go above 1. It is also used as a safety net to prevent too much current going through the output into the LED.

The result of the controller is that the input of LED driver could work up to 15V without the LED breaking on the output side. This is sufficiently enough since the LEDs would not be wanted to work on the edge of breaking and the voltage limit on the LED drivers is 18V. Moreover, the output of the boost SMPS does not allow such high voltage. Therefore, the design of the LED drivers is safe enough to not break.

7.4 Initial Design



After testing out each of the LED drivers with LED lights, three of them are put together in parallel connect to the power supply. The test is operated at a voltage of 10V. All 3 of the LED beacons are successfully lit up. The next step is to the with an emulated PV panel with a power supply. Previously, the boost SMPS with PV panels and each of the LED beacons are tested separately and each of them are working as intended. When combined into a circuit, it is expected to light the LED beacons up. To emulate PV panels, the current limit is set to 0.7A and the voltage is changed from 4V to 5V.

The circuit is working as intended. All three LED beacons light up. When the voltage is at 5V, the beacons are lit up as expected as is operating with around 0.7W to 0.8W. This ensures that the LEDs do not get overpowered and breaks. When the voltage is at 4V, the beacons are still lit up but is very dim. Although the camera could still pick up and differentiate the three colours in the state, it is not ideal since the lights are on the verge of turning off and it is very prone to unexpected circumstances.

To solve the problem, a supercapacitor is introduced in parallel to the three LED beacons. The theory of the part of the supercapacitor is that when the voltage in the DC electricity grid changes, it is able to react and counteract to the change in voltage. When the voltage in the electricity grid is higher than a pre-set reference, the supercapacitor will charge up from the excess voltage in the DC grid. When the voltage in the electricity grid is lower than the pre-set reference, the supercapacitor will discharge into the electricity grid to compensate for the low voltage. This ideally keeps the DC electricity grid at a relatively constant voltage, which eliminates the problem of the brightness of the LED beacons varying with the voltage of the DC grid.

In theory, a bi-directional SMPS could be implemented for this supercapacitor. The code for the controller can be designed so that it can switch between buck and boost depending on the voltage at the port connected to the DC electricity grid. When the voltage is higher than the reference point, the SMPS will be in buck mode. The port connecting the DC grid will be the input and step down the voltage to charge up the supercapacitor. When the voltage is lower than the reference point, the SMPS will be in boost mode. The supercapacitor will discharge and the voltage will be boosted into the DC electricity grid. For example, if the power supply is working in the range of 0V to 5V, and the DC electricity grid is around 0V to 15V, the reference could be around 7.5V. This makes half the time charging the capacitor and half the time discharging the capacitor, achieving a more stable DC electricity grid for the LED beacons. This is based on the assumption that there will be half the time more than the average and half the time less than the average.

In reality, the bi-directional SMPS was not provided in time, resulting in the brightness of the LED beacons highly dependent on the supplied voltage. This is not ideal since the LED turns off when the power supply is at 4V. The LED needs to be lit up as long as possible for better navigation for the robot. For testing purposes, the supercapacitor is connected directly to the DC electricity grid to see its effects on the whole circuit. When the switch is turned on, the LED beacons did not light up immediately. Instead, the LED beacons turn on gradually. This is due to the supercapacitor gradually charging up from the supplied voltage. The voltage of the DC electricity grid increases gradually to 9V, in which the LED beacons turn on at around 1 second after switching on and charges up fully at 2 seconds. When the power supply is turned off, the LED dims out gradually.

7.5 Evaluation

As mentioned above, the bi-directional SMPS was not provided in time for any testing with the supercapacitor. This results in the supercapacitor directly connected to the DC grid, which mean that it could not be controlled to charge and discharge relative to the DC grid. However, the EE2 lab SMPS can actually be modified to a bi-directional SMPS. The switch on the SMPS for buck and boost can be bypassed through modifications in the controller code. Unfortunately, due to time constrains, it is not able to do so which results in the lack of testing in the supercapacitor.

8 Computer Vision

8.1 Beacon Detection

A key part of the project was the detection of the beacons to assist the dead reckoning to ensure accuracy in the localisation of the rover. The requirements of this system are as follows:

- The hardware specified on the FPGA must be able to detect each of the individual colours of the beacons (red, orange, blue)
- Bounding boxes must be accurately placed around each of the beacons
- Coordinates of corners of the boxes must be stored and passed onto the FPGA

The sample code provided for the FPGA vision system was looked at as a starting point, mainly focusing on the starter image processor implemented in Verilog (EEE_IMGPROC.v) [10] [11]. In this Verilog module, the 24-bit ”sink_data” input from the streaming sink is broken up into separate 8-bit wires: red, green and blue. These wires are used to determine the values assigned to the 1-bit ”red_detect”, ”blue_detect”, ”orange_detect” and ”white_detect” wires, with the first three wires required for beacon detection and the latter wire for lane detection. These Boolean values are determined by evaluating the individual 8-bit RGB values from the video feed, for each of the beacon colours (red, orange and blue) there are a range of RGB values which are acceptable for each colour [12]. For this reason, each of the red, green and blue wires were checked against threshold values as shown below:

```
assign red_detect = (red[7:0] >= 8'd200) & (green[7:0] <= 8'd127) & (blue[7:0] <= 8'd127);
assign blue_detect = (red[7:0] <= 8'd127) & (green[7:0] <= 8'd127) & (blue[7:0] >= 8'd200);
assign orange_detect = (red[7:0] >= 8'd200) & (green[7:0] == 8'hff) & (blue[7:0] == 8'hff);
```

Figure 19: Threshold values

Representing threshold values in decimal was purely to make tweaking these values easier for calibration of the beacon detection.

The initial calibration without the beacons with objects of similar colours proved relatively successful in detection of colour in each respective object.

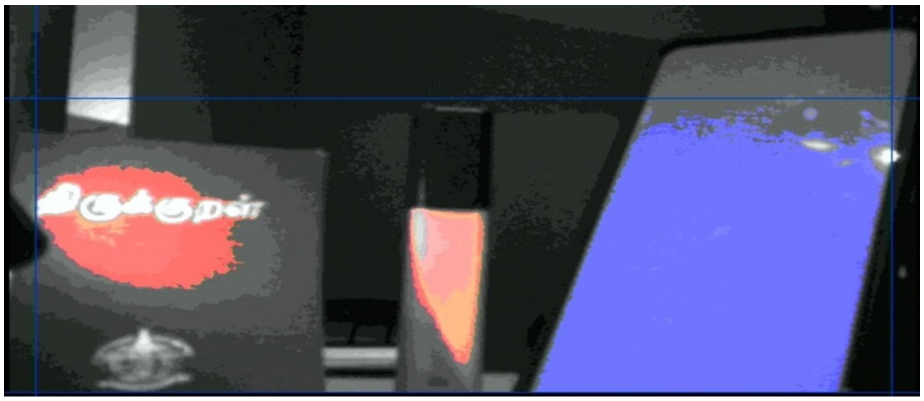


Figure 20: Beacon detection

However, the image processor was unable to detect the object in its entirety due to lighting and overly strict colour detection thresholds. This was problematic as it interfered with the plan of using bounding boxes and their areas to determine the distance from the rover relative to other beacons, which cannot be done effectively if the bounding boxes do not always surround the beacon in its entirety.

In order to address this issue, the colour detection was calibrated and tested with the LED beacons, where it was learned through collaboration with the power system sub-team that the brightness of the LEDs should be limited due to bright light being detected as white light, in addition to the fact that there may be some flashing in the LEDs powered by the PV cell/ supercapacitor. Using this information, the thresholds had to be changed to allow for more robust levels of detection, which required numerous cycles of trial and error (i.e. accounting for the various shades of red, blue or orange as brightness isn't constant throughout the beacon).

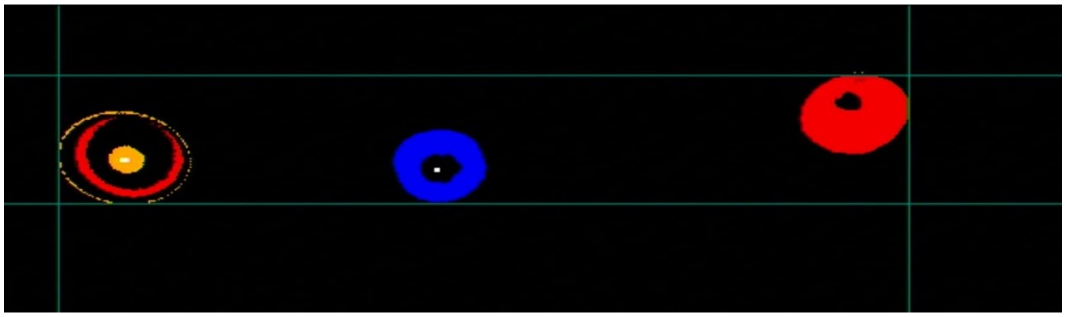


Figure 21: Yellow, blue and red beacons (from left to right)- Misdetection of red in yellow beacon

Due to the overlapping nature of the red and yellow colours, the distinction between red and yellow beacons was blurred when using RGB thresholds. This made RGB-HSV conversion an attractive alternative as it separates the brightness and colour saturation from the colour itself, which allows for more precise detection of colours as well as more simplified thresholds [13]. Attempts were made at converting the colour scheme from RGB to HSV for the detection, the complexity (see appendix for conversion equations) of the implementation, which required multiplication of variable wire values, was time consuming and problematic. This unfortunately meant that it had to be abandoned and more careful calibration of RGB thresholds were to be used.

The coordinates of each of the boxes were passed to the NIOS II processor using the memory mapped interface provided in the starter project [11]. They are sent to the NIOS II processor, with separate state machines for information regarding each colour beacon, implemented in Verilog using . The image processor sends four things for each state machine,

The implementation of the beacon detection was successful in recognising each of the beacons, however there was an issue posed by the orange beacon as it cannot be detected in its entirety. This does not interfere greatly with the function of the segway regarding triangulation as the vision system still detects the centre of the beacon correctly, just not completely. Given more time, it would have been beneficial to the function of the rover if more time was invested in implementing conversion to HSV so that the beacons would be detected more accurately, reducing the risk of error in the coordinates of the centre of each beacon.

8.2 Triangulation

The coordinate information from the vision system is passed onto the ESP32, where it is combined with wheel odometry to determine a position for the segway to complement the dead reckoning to ensure accuracy in the localisation of the rover.

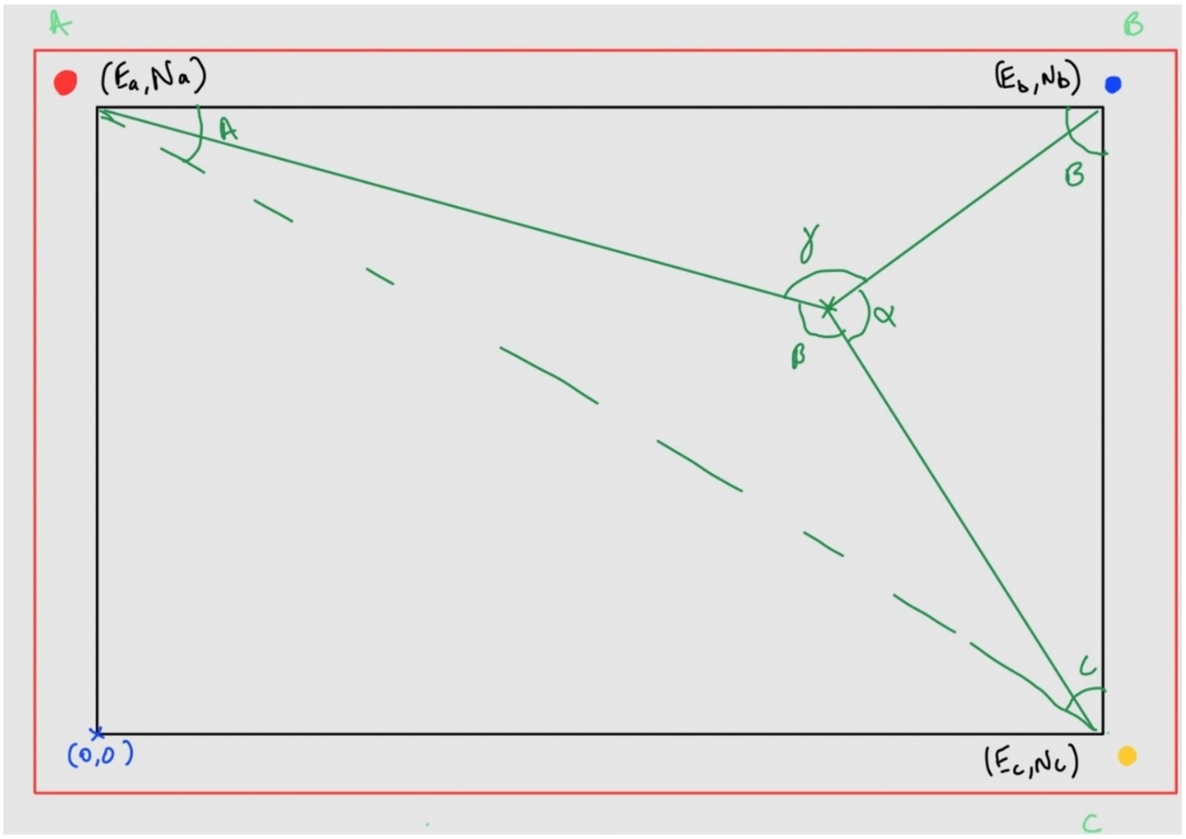
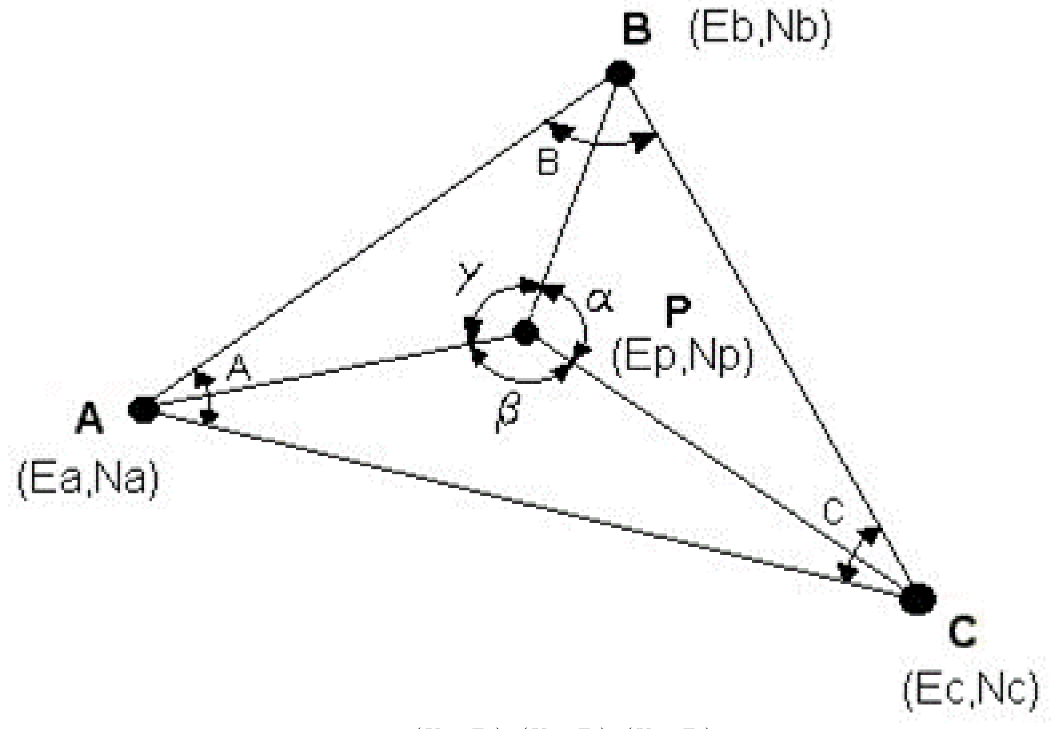


Figure 22: Diagram of maze with beacon

When the rover is placed in one of the corners, the coordinate of that corner is set to (0,0) and the three beacons are placed in the remaining corners outside the bounds of the maze. The vision system sends the top left and bottom right coordinates of the beacons it detects to the NIOS II processor (along with the colour detected), which then passes the data to the ESP32 to be processed by Arduino code. On the ESP32, the coordinates of the middle of each bounding box for each colour is calculated. The code on the ESP32 orders the wheels to rotate until the red beacon is at a certain x coordinate (the centre of the frame), after which the ESP32 instructs the segway to rotate clockwise until the x coordinate of the blue beacon reaches the same point of interest, at which point the total angle rotated is determined by a function which calculates values from the wheel odometry. This is repeated from the blue to yellow beacon and finally the yellow to red.

The method of using angles between beacons transformed the task into a 3-point resection problem, two solutions of which were considered: ToTal [14], and Tienstra's method. The ToTal method may have a faster execution time than Tienstra's method [14], but is more complicated (refer to appendix) in its implementation, and thus the decision was made to use Tienstra's method for implementation of the triangulation algorithm due to time constraints of the project. Tienstra's method for localisation is given by figure 23.



$$E_p = \frac{(K1 \times E_a)+(K2 \times E_b)+(K3 \times E_c)}{K1+K2+K3}$$
$$N_p = \frac{(K1 \times N_a)+(K2 \times N_b)+(K3 \times N_c)}{K1+K2+K3}$$

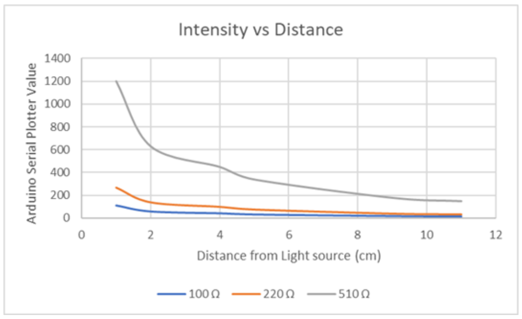
$$\begin{aligned} \text{Where, } K1 &= 1/(\cot(A) - \cot(\alpha)) \\ K2 &= 1/(\cot(B) - \cot(\beta)) \\ K3 &= 1/(\cot(C) - \cot(\gamma)) \end{aligned} \tag{1}$$

Figure 23: Tienstra's method for localisation

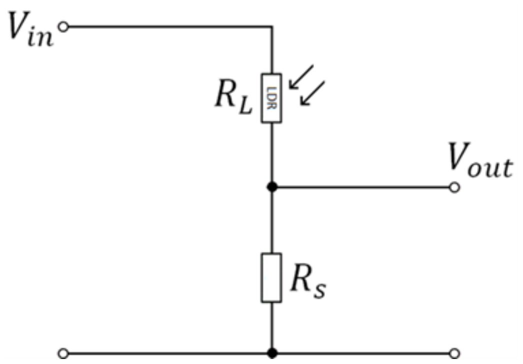
8.3 Navigation

The navigation system developed for the segway is a crucial part of its operation. The system has been designed to navigate autonomously through a maze environment, using a combination of Light-Dependent Resistors (LDRs), a camera, and a sophisticated wall-following algorithm. This paper will outline the techniques and technologies used in the construction of the navigation and mapping system.

The design of the segway includes the integration of LDRs on each side, which are used for wall detection. Every wall in the maze is illuminated, and the LDRs, which are sensitive to light intensity, are utilized to sense the presence of these walls. The LDRs have been carefully calibrated to a specific light intensity threshold, which is used to trigger a decision from the segway's navigation system.



(a) LDR response characteristics



(b) LDR circuit diagram

Figure 24

Using a potential divider circuit, multiple resistor values were tested to measure light intensity variances as they move away from wall. A large spread of Arduinos' measured values between varying distances from the light source (wall) was needed to set the necessary conditions for when to continue forward and when to turn, while also decreasing the margin of error.

The primary navigation algorithm employed is the right-wall following method, where the segway is programmed to follow the wall on its right side. The segway continuously checks the light intensity on its right-side LDR; if the intensity is high, implying the presence of a wall, it moves in a straight line. If the light intensity falls below the calibrated threshold, indicating the absence of a wall, it steers right until the light conditions are satisfied.

Simultaneously, the LDRs also serve a dual purpose of helping to map the maze layout. Whenever the rover encounters a junction (a point where more than two paths intersect), the LDRs detect a sudden change in light intensity and when at least 2 LDRs detect no light. This is recorded as a node in the maze's graphical representation. Each node is sent to a server for processing and storage. All other pathways, representing maze corridors, are marked as weighted edges between nodes and are similarly stored on the server.

To aid the segway in maintaining accurate positioning data, coloured beacons are placed along the way. Each beacon has a pre-known location. The camera mounted on the segway recognizes these beacons and updates the segway's current location in the mapping system. This ensures a continuous update of the segway's current location in the maze, thus increasing the accuracy of the generated map.

In scenarios where the segway moves out of sight of the beacons, a technique called dead reckoning is used. Dead reckoning estimates the segway's position relative to a known starting point (the origin), based on its direction and distance travelled. This provides an additional layer of redundancy to the system, ensuring that the segway's position can be estimated even when other localization systems (like beacon recognition) are not applicable.

In conclusion, the combination of LDR sensors, a camera, and wall-following and dead-reckoning algorithms provides a comprehensive solution for the segway to navigate and map the maze. This multi-faceted approach ensures robustness, reliability, and adaptability to changing conditions within the maze, thereby enabling efficient and autonomous maze navigation and mapping.

8.4 Dead-reckoning position estimation

The dead-reckoning algorithm is used to estimate the segway's position relative to its starting point defined as $(0, 0)$. The algorithm counts the number of steps of each motor and adjusts the estimated angle and position accordingly.

The segway turns by shutting down one motor and rotating the other, therefore the angle is estimated during a turn by using the number of steps of the outside motor as the arc of a circle.

$$\theta = \frac{360\,\text{deg} \times \text{Steps} \times \text{Distance per step}}{2\pi \times \text{Wheelbase}}$$

Figure 25: Angle estimation

The position during a turn is defined by figure 26:

$$\Delta x = \frac{\text{Wheelbase}}{2} \times \sin(\theta)$$
$$\Delta y = \frac{\text{Wheelbase}}{2} \times \cos(\theta)$$

Figure 26: Position estimate during a turn

The position during a straight line, assuming angle 0 is facing towards positive y, is defined in figure 27

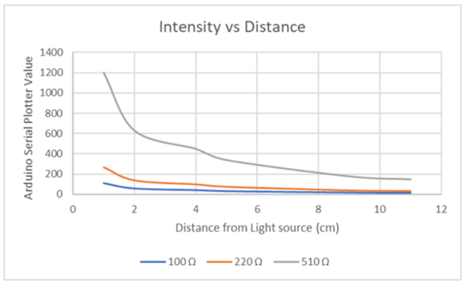
$$\Delta x = \text{Steps} \times \text{Distance per step} \times \sin(\theta)$$
$$\Delta y = \text{Steps} \times \text{Distance per step} \times \cos(\theta)$$

Figure 27: Position estimate in a straight line

8.5 Testing

Three different types of light sensors are tested to see which was most suited for the task of navigating. A high sensitivity to light and a steady variance in light detected depending on distance from light source are needed in the design. A GL5528 photoresistor was tested first:

(a) Photoresistor



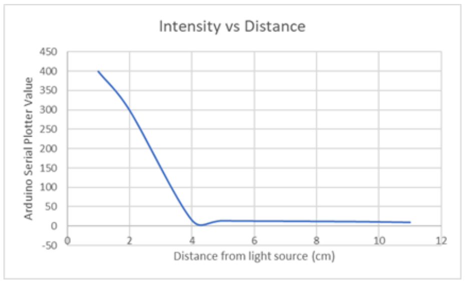
(b) Phototransistor response characteristics

Figure 28

The GL5528 photoresistor proved to have great results when it is paired up with a 510Ω resistor in a potential divider circuit.

An SFH 300 phototransistor was tested next, the idea behind choosing this one was that it would be able to detect light from more directions due to its design:

(a) Phototransistor

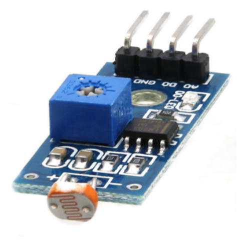


(b) Phototransistor response characteristics

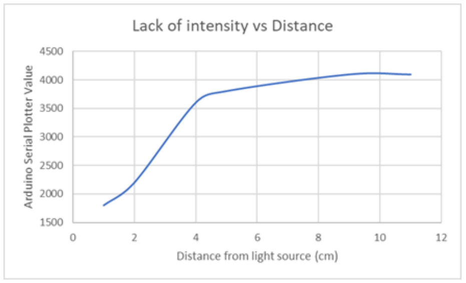
Figure 29

The issue with the phototransistor was that the variance was not steady and it would barely detect anything if it was not immediately next to the light source.

Finally, the last component that was tested was a photoresistor sensor module. The idea behind trying this was to save space on the breadboard as it did not require the use of a potential divider circuit:



(a) Photoresistor module



(b) Photoresistor module response characteristics

Figure 30

Again, the issue with this was that it barely detected changes in conditions until the light source was very close. This performed better than the SFH 300 but worse than the GL5528. In the end, the GL5528 was chosen as it was the best suited for gauging its distance to the light source. This property would be heavily relied on in the navigation.

8.6 Algorithm

When searching for an algorithm that computes the shortest path in a graph, there were a breadth of choices. These include Dijkstra's algorithm, Bellman–Ford algorithm, A* search algorithm, Floyd–Warshall algorithm, Johnson's algorithm. Now these algorithms needed to be assessed with respect to their time-complexities.

Algorithm	Dijkstra	Bellman-Ford	A* search	Floyd-Warshall	Johnson's
Space complexity	$O(E)$	$O(E)$	Heuristic Function	$O(V^2)$	Heuristic Function
Time complexity	$O(V^2)$	$O(VE)$	Heuristic Function	$O(V^3)$	$O(V^2 \log V + VE)$
	V = Vertices			E = Edges	

Figure 31: Time and space complexities of various algorithms

The use of Dijkstra's algorithm was proposed for a non-orthogonal maze traversal problem, where the start and end points are diagonally opposite. The reason is as follows:

Dijkstra's algorithm was a formidable option for our problem set, particularly when the uniform cost and non-negativity are considered of our edge weights. While the A* algorithm's uses of a heuristic function may seem appealing at first glance, it's worth considering that Dijkstra's has an inherent advantage in dealing with graphs that don't provide additional heuristic information. Dijkstra's algorithm operates by treating every direction as equally potential, thus exploring the entire graph uniformly.

Contrary to the A* search algorithm, which heavily relies on a heuristic function, Dijkstra's algorithm does not need any heuristic information to guide the search, and this has the benefit of a more thorough exploration of the graph which is advantageous in situations where the heuristics may not provide an optimal solution.

As for the Bellman-Ford algorithm, it is less efficient than Dijkstra's algorithm when considering time complexities, as there will be less vertices than edges, making this a less practical choice for our situation. It is also usually reserved for scenarios where edge weights can be negative, which is not the case for our maze.

Floyd-Warshall and Johnson's algorithms, meanwhile, are designed to compute the shortest paths between all pairs of nodes. In our specific problem where the only need is to determine the shortest path from a single source to a single target, these algorithms seem to be an overuse of resources.

Dijkstra's algorithm, in its simplicity and uniformity, shines in this particular scenario where there is only a single source and a single target. It's an efficient algorithm, both in terms of time and space complexities, and offers a reliable method of finding the shortest path in a graph. Thus, the use of Dijkstra's algorithm is chosen in our maze traversal.

9 Manufacturing

9.1 Circuit Design

A modular approach was used to construct the required circuit. Each of the components are tested individually before bringing them together. The circuit is shown as follows:

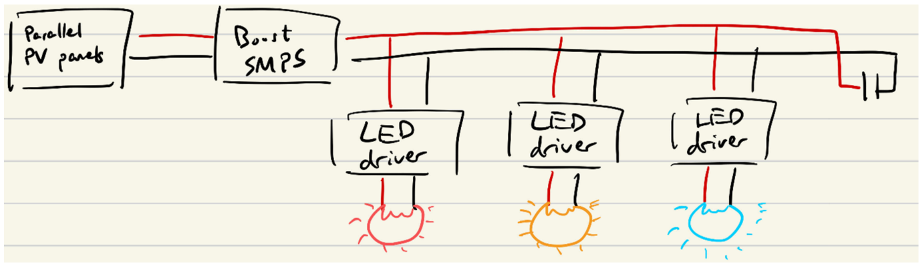


Figure 32

The power supply is chosen to be 3 parallel PV panels since only this configuration gives enough current. The power supply for series PV panels is too unstable to be implemented well in the design.

Two types of controller design are considered for the LED drivers. The first one is a simple controller that sets the duty cycle according to the amount of current through the SMPS. The second one is a PID controller. The advantage of a PID controller is that it provides better control and voltage regulation on the output. Although the result looks the same from both controllers, PID controller will be used since it gives better control over the output by the user.

Unfortunately, the bi-directional SMPS is absent at the time and during the demo, it will not be implemented in the final circuit design. In theory, it should control the supercapacitor for charging or discharging relative to the voltage in the DC grid.

Although this result is far from the ideal circuit, this circuit will be used in the demo since this circuit is easy to implement. Further amendments and improvements are mentioned in the evaluation subsection under power system.

9.2 Chassis Design

Many different design concepts were considered when it came to designing the chassis.

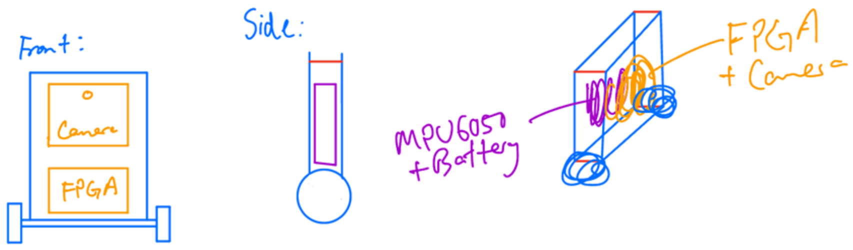


Figure 33: Standing design

This would be the simplest to build since it would only require one swivel wheel and not many changes are required to the existing chassis; just to make the wheels perfectly in the centre. However, it would be challenging to fit everything in such a tight narrow space widthout greatly widening the chassis. This would further prove to me even more difficult when it comes to debugging.

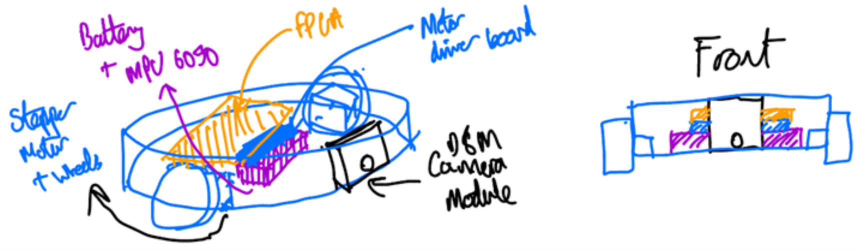


Figure 34: Flat design

This design resembles that of an automatic vacuum and has the most aesthetic look if made correctly. It provides enough space for all components to fit appropriately. With a design with the centre of mass super close to the wheels and ground, the control system cannot be easily implemented all ends could easily touch the ground. This is further supported by the fact that there is virtually no height meaning very little to no inertia at the top. This would mean the gyroscope would have a hard time detecting changes in angle tilt. Furthermore, a circular design would prove to be impractical when it came to driving around on the field.

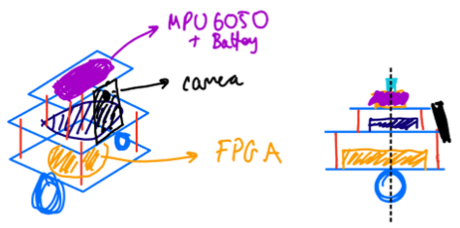


Figure 35: Triple tower design

In the end, the triple tower design set up was chosen. This allowed a variable weight distribution via the use of nuts and threaded rods. This meant that when it came to testing, it was very convenient for us to change the heights of relevant parts to allow for an ideal control system. Making the tower very top heavy, resembled that of the heavy mass in an inverted pendulum. This modular design would ultimately save time and allow for a range of possibilities. Furthermore, this design would allow for quick and easy debugging without dismantling the whole system.

M10 threaded rods were chosen to ensure that the chassis is rigid, robust and durable. This added plenty of mass to the rover. However, the fact that this mass was static meant that it added unnecessary weight. In turn, this meant that the torque produced by the wheels was insufficient, and as a result, bigger wheels would need to be printed to allow for further leverage. In hindsight, thinner threaded rods, and nuts of size maximum M5 would have been more than enough to keep the whole segway secure. This would have further saved time in 3d-printing our own wheels as the torque produced would have been sufficient since a small weight component means a smaller force as according to Newton's second law.

The required torque can be calculated with the formula in figure 36

$$\text{Torque} = \text{height} \times \text{mass} \times \text{acceleration due to gravity} \times \sin(\text{maximum tilt})$$

Figure 36: Torque Formula

Using this formula, it is known that a vague minimum torque required. From this, it is needed to ensure that the maximum torque is below 0.96Nm. This value considers both stepper motors where the nominal value of 0.48Nm is taken from the datasheet.

Therefore, as a next step, by reducing the mass diameter from M10 to M5, 75% of the mass of the threaded rods would be eliminated. This means that the torque produced by the motors would be enough to take full advantage of the control system allowing for smooth operation.

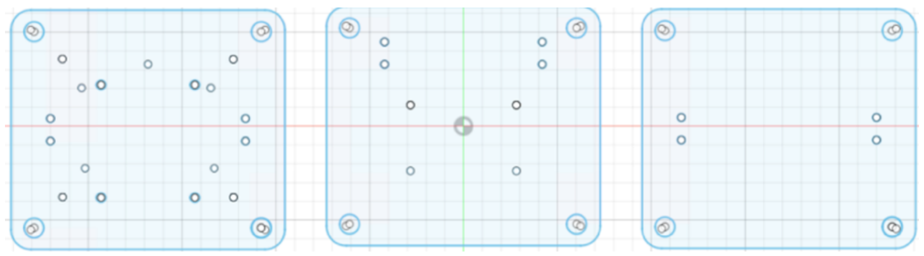


Figure 37

Fusion 360 was used to design the new chassis. A few changes were made based on the original design that was offered to make it better fit our requirements. Notably, the original wheel mounts were placed further apart were placed horizontally further apart with an offset in the Y-axis to allow for the wheels pivot to be in line with the centre of mass.

Having the battery on top was chosen to allow it to be ”top heavy” referring to a configuration in an inverted pendulum design in which most of the mass is concentrated at the top, increasing the pendulum’s potential to tip over. This imbalance has an impact on the system’s stability, requiring precise control to keep the system upright. This design largely relies on inertia, which is the resistance of an object to changes in motion. It is more difficult to quickly adjust the pendulum’s position due to the top-heavy configuration’s high inertia, which derives from the concentrated mass at the top. As a result, strong control measures are required to stabilise the system and reduce its consequences.

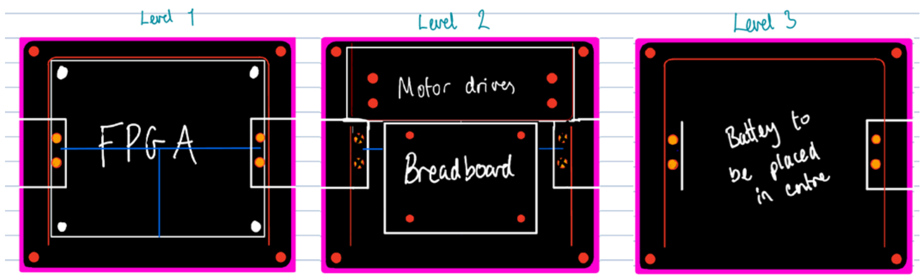


Figure 38

Combining all the levels gave us the following:

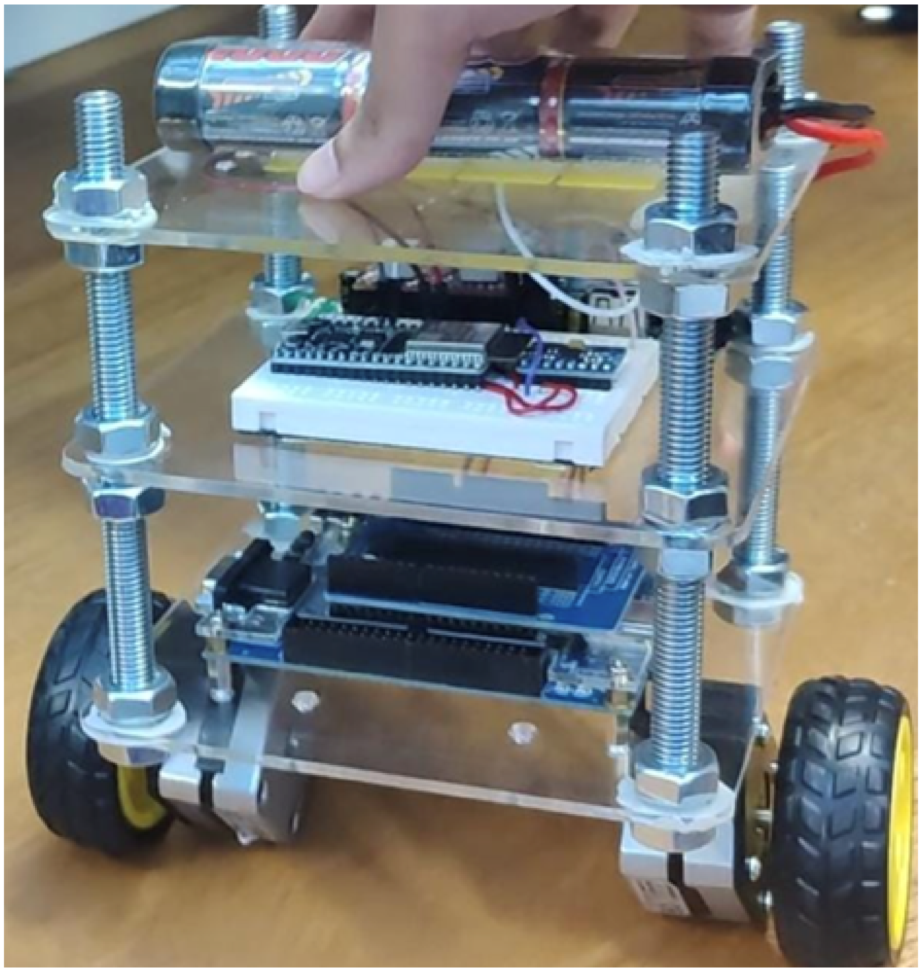


Figure 39

10 Integration

Once all the individual systems were developed, they had to be integrated into the final system. As each subsystem was designed with scalability and expandability in mind, connecting them together was simply a matter of communication between them.

10.1 Inter-device Communication

As the camera sends information to the Nios II processor on the FPGA and the ESP32 communicates with the server over WiFi, a communication channel between the FPGA and the ESP32 is paramount. Initially, I2C was experimented as it is a simple interface and has good support in the ESP32 libraries, however the IP available for the FPGA only provided an I2C master device. The ESP32 does support communication as an I2C slave device, but it was not working correctly in testing. SPI was then chosen to be used as it is a simpler interface and there is good support for the ESP32 as a slave device.

10.2 SPI Bus

The serial peripheral interface (SPI) is a full-duplex master-slave interface with 4 wires. The 4 wires are:

- **MOSI** – Master Out Slave In
- **MISO** – Master In Slave Out
- **SCLK** – Serial Clock
- **SS** – Slave Select

The master device generates the clock signal and a high signal on MISO or MOSI represents a binary 1. This means that the interface communicates at 1 bit per clock cycle. SPI hardware is implemented with shift registers that can be made an arbitrary length, so long as each device agrees on a minimum. The data to be transferred is buffered into the shift register and then when the SS line goes low the shift register begins advancing pushing each bit every clock cycle, another shift register is used to receive data into.

The Intel SPI FGA provides SPI hardware and an interface to the processor as an Avalon Memory Mapped slave which enables sending an arbitrary number of bytes over SPI through the hardware abstraction layer. [15] The IP can be configured with a shift register of 8-32 bits, 8 bits were chosen as there would be wasted time if less than 4 bytes were to be transmitted. The clock speed was chosen to be 12.5Mhz giving a theoretical maximum bitrate of 12Mbps. However, in practice the processor can't buffer the data fast enough and so the effective throughput is limited to 2.4Mbps.

Espressif provide an SPI slave driver which can be configured to read the incoming data into a small memory buffer or, using direct memory access, into main data memory. Since the SPI transactions are smaller than 32 Bytes Espressif recommend using the default configuration without DMA. [16]. The ESP32SPISlave Arduino library provides interfaces for this driver. The ESP32 queues an empty transaction onto the buffer and waits for the driver to trigger an interrupt signalling the start of a transaction, the library does this through the use of a slave.available() method that is captured in a while loop. The data buffers are emptied into a global variable and a flag is set to indicate new information. The main loop then acts accordingly.

10.3 WiFi Communication

The ESP32 has an onboard WiFi module, this enables transmission of information over the internet using the HTTPClient library. [17] The ESP32 stores an internal JSON document using the ArdunoJson library. This is a dynamic document stored on the heap, as recommended by the developers. [18] This enables simple integration with the REST API through GET and POST requests to keep the server up to date with the latest telemetry or mapping data. The ESP32 is pre-loaded with the SSID and password of a local WiFi access point in order to connect to the internet.

11 Evaluation

A rover was designed so as it is able to autonomously navigate through a maze using an outer wall following approach, wall following was done based on light intensity coming from the walls of the maze, this was then used as a basis to perform the mapping of the maze where the light intensity would also tell us about any features of the maze present such as junctions or corners. Using this information it was possible to represent the mapping using a graph data structure which made the process of performing any maze solving algorithm easier to implement, the algorithm of choice was dijkstra's with heaps due to its simplicity and relative low computing cost. This implementation covers two out of the three functional requirements. The balancing was done on a separate rover where an accelerometer and gyroscope module were used in conjunction with a pid control system to be able to balance the robot on two wheels, covering the final requirement of the project.

Many challenges were encountered during the completion of the project. Multiple approaches were tested, such fulfilling all three functional requirements on one rover, increasing the role of computer vision in mapping by implementing edge, contour and corner detection; streaming the video feedback onto the server so that opencv could be used for computer vision. However the group experienced many roadblocks, such as time constraints, complexity of some of the problems and poor computational performance, the amalgamation of these constraints is what led to the final design and implementation of the rover.

12 References

[1] "Common routing tasks." (2023), [Online]. Available: https://angular.io/guide/router (visited on 05/22/2023).

[2] "Two.js documentation." (2023), [Online]. Available: https://two.js.org/docs/two/ (visited on 05/22/2023).

[3] "Three.js documentation." (2023), [Online]. Available: https://threejs.org/docs/ (visited on 05/30/2023).

[4] (2023), [Online]. Available: https://material.angular.io/ (visited on 05/23/2023).

[5] "What is flask python." (2021), [Online]. Available: https://www.pythonbasics.org/what-is-flask-python/.

[6] M. Abiola. "Node.js vs. flask: Key advantages, disadvantages, and differences." (2023), [Online]. Available: https://hostadvice.com/blog/web-hosting/node-js/node-js-vs-flask/.

[7] S. Ilarslan. "What is gunicorn?" (2022), [Online]. Available: https://medium.com/@serdarilarslan/what-is-gunicorn-5e674fff131b.

[8] Knista. "What is nginx? a basic look at what it is and how it works." (2022), [Online]. Available: https://kinsta.com/knowledgebase/what-is-nginx/.

[9] RedHat. "What is a REST API?" (2020), [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[10] H. Merdan. "EEE balance bug repository." (2023), [Online]. Available: https://github.com/hakanmerdan/EEEBalanceBug.

[11] E. Stott. "FPGA starter system documentation." (2023), [Online]. Available: https://github.com/edstott/EEE2Rover/blob/master/doc/FPGA-system.md.

[12] "RGB clor codes chart." (2023), [Online]. Available: https://www.rapidtables.com/web/color/RGB_Color.html.

[13] R. Madsen. "Color models and color spaces." (2023), [Online]. Available: https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html.

[14] V. Pierlot and M. Van Droogenbroeck, "A new three object triangulation algorithm for mobile robot positioning," *IEEE Transactions on Robotics*, vol. 30, no. 3, pp. 566–577, 2014. DOI: 10.1109/TRO.2013.2294061.

[15] Intel. "Embedded peripherals IP user guide." (2023), [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683130/21-4/alt-avalon-spi-command.html.

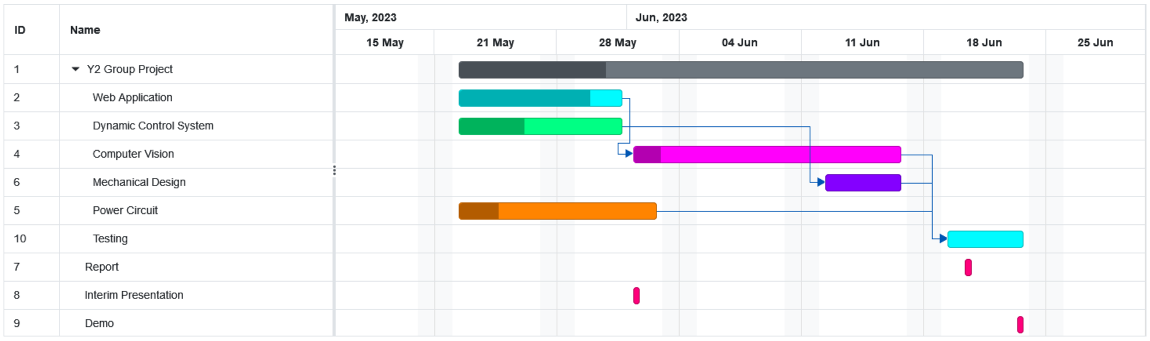
[16] Espressif. "SPI slave driver." (2023), [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/spi_slave.html.

[17] Espressif. "ESP HTTP client." (2023), [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_client.html.

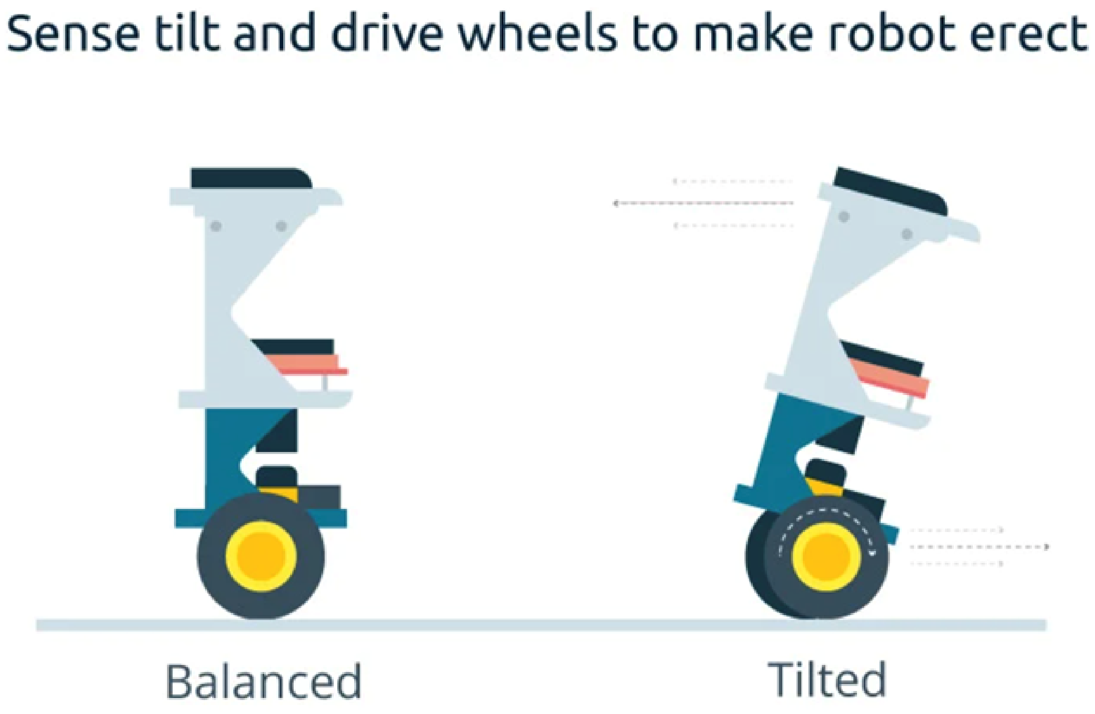
[18] B. Blanchon, *Mastering ArduinJson 6*, 3rd ed. Benoit Blanchon, 2021, ch. 4.2.2, pp. 110–111, ISBN: 9782956430933. [Online]. Available: `https://arduinojson.org`.

13 Appendix

A Gantt Chart



B Tilting example



C ToTal Algorithm

Given the three beacon coordinates $x_1, y_1, x_2, y_2, x_3, y_3$ and the robot coordinates x_0, y_0, and the three angles $\alpha_1, \alpha_2, \alpha_3$:

1. Compute the modified beacon coordinates:
   $x'_1 = x_1 - x_2, y'_1 = y_1 - y_2, y'_3 = y_3 - y_2$

2. compute the three cot:
   $T_{12} = \cot(\alpha_2 - \alpha_1), T_{23} = \cot(\alpha_3 - \alpha_2), T_{31} = \frac{1-T_{12}T_{23}}{T_{12}+T_{23}}$

3. Compute the modified circle center coordinates:
   $x'_12 = x'1 + T_{12}y'_1, y'_12 = y'_1 - T_{12}x'_1,$
   $x'_23 = -x'3 - T_{23}y'_3, y'_23 = -y'_3 + T_{23}x'_3,$
   $x'_31 = (x'_3 + x'_1) + T_{31}(y'_3 - y'_1),$
   $y'_31 = (y'_3 + y'_1) - T_{31}(x'_3 - x'_1)$

4. Compute k'_{31}:

   $k'_{31} = x'_1x'_3 + y'_1y'_3 + T_{31}(x'_1y'_3 - x'_3y'_1)$

5. Compute D (if D = 0, return with an error):
   $D = (x'_{12} - x'_{23})(y'_{23} - y'_{31}) - (y'_{12} - y'_{23})(x'_{23} - x'_{31})$

6. Compute the robot position x_R, y_R:
   $x_R = x_2 + \frac{k'_{31}(y'_{12}-y'_{23})}{D}, y_R = y_2 + \frac{k'_{31}(x'_{23}-x'_{12})}{D}$

D Control system code

```
void keepBalancing() {
  // Read tilt angle from MPU6050
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  // Apply calibration offsets
  float calibratedX = a.acceleration.x - offsetX;
  float calibratedY = a.acceleration.y - offsetY;
  float calibratedZ = a.acceleration.z - offsetZ;

  // Calculate the tilt angle
  float currentAngle = atan2(-calibratedX,
    sqrt(calibratedY * calibratedY + calibratedZ * calibratedZ)) * 180.0 / PI;

  // Calculate the difference between target and current angle
  float angleError = targetAngle - currentAngle;

  // PID control calculations
  double output = Kp * angleError
                + Ki * integralTerm
                + Kd * (angleError - prevError);

  // Update the integral term
  integralTerm += angleError;

  // Update the previous error
  prevError = angleError;

  // Adjust motor speeds based on PID output
  if (output > angleThreshold) {
    motor1.setSpeed(motorSpeed);
    motor2.setSpeed(-motorSpeed);
  } else if (output < -angleThreshold) {
    motor1.setSpeed(-motorSpeed);
    motor2.setSpeed(motorSpeed);
  }
  else {
    motor1.setSpeed(0);
    motor2.setSpeed(0);
  }

  // Move the motors
  motor1.runSpeed();
  motor2.runSpeed();
}

void calibrateMPU6050() {
  float sumX = 0.0;
  float sumY = 0.0;
  float sumZ = 0.0;
  const int numSamples = 500;

  // Collect calibration data
  for (int i = 0; i < numSamples; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    sumX += a.acceleration.x;
    sumY += a.acceleration.y;
    sumZ += a.acceleration.z;
    delay(10);
  }
  Serial.println("Callibration Complete");

  // Calculate calibration offsets
  offsetX = sumX / numSamples;
  offsetY = sumY / numSamples;
  offsetZ = sumZ / numSamples;
  Serial.print("OffsetX: ");
  Serial.println(offsetX);
  Serial.print("OffsetY: ");
  Serial.println(offsetY);
  Serial.print("OffsetZ: ");
  Serial.println(offsetZ);
}
```