Imperial College London

May – June 2023

EEEBalanceBug

2nd Year Design Project

ELEC50008

Group 1

Mr Hakan Merdan

Word Count: 10,845

David Cai	02019587
Advik Chitre	02019365
Aranya Gupta	02045024
Han Jang	01856804
Ben Smith	02086844
Digo Yu	02024612

Abstract

A two-wheeled balancing robot was created to map a maze autonomously and calculate the shortest path. The maze was bounded by LED strips, and up to 3 differently coloured beacons could be placed around the maze beforehand. The beacons must be powered through a carefully optimised DC grid by solar power.

There was a budget of £60 for the robot. An FPGA-based camera system must be used. Other optical sensors were permitted, but no other cameras.

The methods employed include designing an advanced embedded system, hardware-based image processing, scalable cloud infrastructure and optimal energy distribution systems.

Requirements

"Requirements" are referred to throughout this report in reference to the relevant requirements below [1]:

1. Functional requirements. The system must:
   - 1.1 Autonomously move through a maze without crossing an illuminated line.
   - 1.2 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.
   - 1.3 Balance on two wheels.

2. Non-functional requirements. The system must be:
   - 2.1 Reliable and able to complete its task without human intervention.
   - 2.2 Robustly and efficiently constructed.
   - 2.3 Coded for:
     a. Usability
     b. Testability
     c. Maintainability
     d. Scalability

3. Implementation restrictions
   - 3.1 The permitted means of locating the robot within the maze are:
     a. Dead reckoning based on accelerometer, gyroscope and wheel revolution counting.
     b. Optical detection of the maze markings and up to three illuminated beacons by the robot.
   - 3.2 Illuminated beacons shall be powered by a specified PV-array emulator and provided energy conversion modules.
   - 3.3 The robot will use the provided FPGA-based camera system.

Contents

Summary of Strategy

A modular design chosen so subsections could be tested individually, and people could be reallocated to urgent tasks if necessary. Subsections could progress in parallel, allowing greater efficiency.

The problem of traversing the maze was decomposed and abstracted into 4 commands that can be issued to the robot by a central server that uses a maze-mapping algorithm. The commands (described in the FreeRTOS section) are high-level (for example, "go forwards until a junction is reached", rather than "go forwards 100mm and check if there is a junction") hence the system is decoupled from inaccurate real-world measurements.

The server used a state machine-based approach to command the robot. To meet Requirement 1.2, telemetry data was stored in a database for retrieval by the end-user via a client-side user interface.

Due to the narrow field of view of the camera, detection was only possible of either the beacons or the LED strips. It was decided, considering the computational requirements, only the beacons would be detected by the camera and used for triangulation. Instead, the LED strips would be detected by time-of-flight (ToF) sensors.

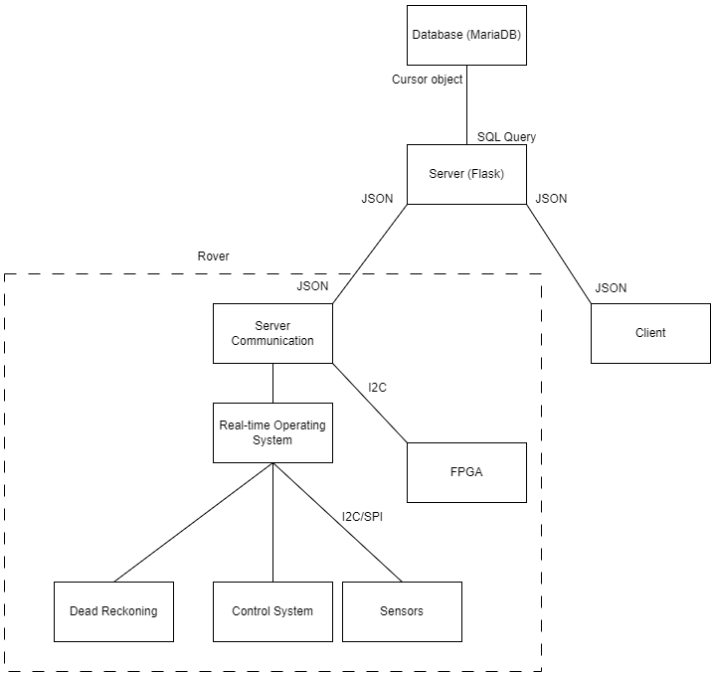


Figure 1 - Project decomposition and interfaces.

Project Management

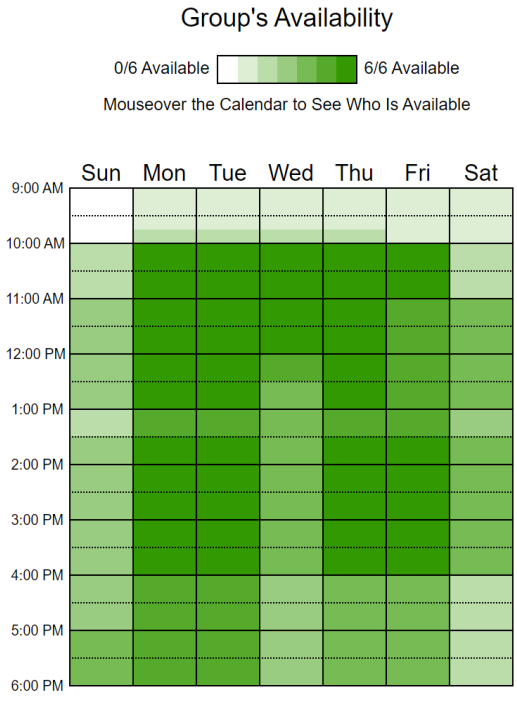


Figure 3 - When2meet availabilities.

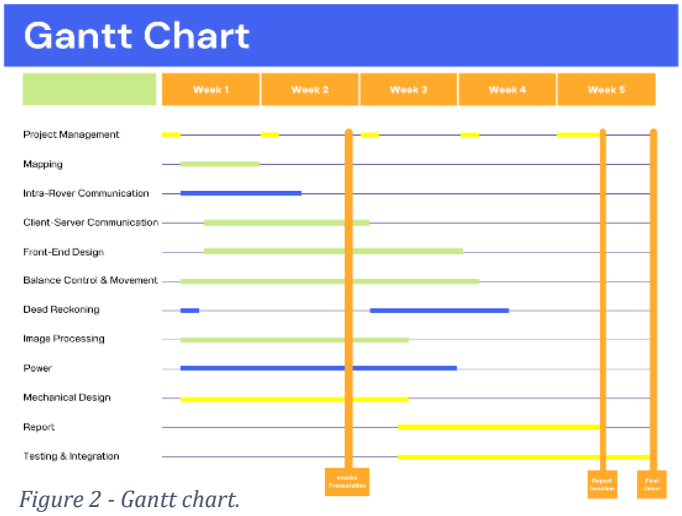


Figure 2 - Gantt chart.

Initial availabilities of group members were established using When2meet (Figure 3). During the first weekly meeting, the strategy presented previously was decided upon. A Gantt chart (Figure 2) was created to give a guide for the project's timeline and tasks were assigned.

The group organised its documents and code through a Git repository. This was accompanied by OneNote for informal ideas, and a Fusion360 team for mechanical design.

Robot

Due to the latency of the onboard Wi-Fi and to ensure Requirement 2.2 is met, many of the systems had to be implemented onboard the robot. This required real-time digital signal processing. The methods used to achieve this, to control the robot and to synchronise the subsystems are described in this section.

Inertial Measurement Unit (IMU)

Overview and Selection

It is important to always know the orientation (yaw) of the robot for maze traversal and the pitch for balancing. Therefore, the IMU and sensor fusion only needed to find pitch and yaw, not linear acceleration. Roll could be ignored, since it is not possible for the robot to move in that axis.

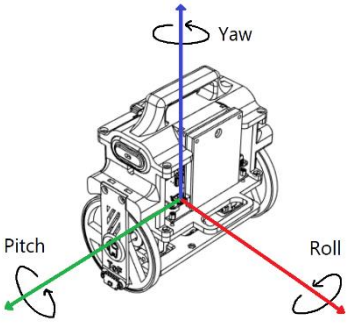


Figure 4 - Axes of rotation.

The ICM-20948 IMU was chosen, over the provided IMU, for the following reasons:

- **Supports SPI:** Faster data transfer (up to 7Mbps) using SPI compared to the I2C interface on the MPU6050 (400Kbps). This allows the sensor to be read at a fast rate (potentially >500Hz), minimising time spent waiting for new data.
- **Direct memory access (DMA):** The SPI peripheral is connected to the ESP32's DMA controller unlike the I2C peripheral. While this functionality was not implemented, it would allow an increase to the system's efficiency.
- **Magnetometer:** The 3-axis magnetometer provides absolute yaw, potentially reducing drift.
- **Digital Motion Processor (DMP):** A small processor built into the sensor that can unburden the host microcontroller of time-consuming sensor fusion calculations and perform background calibration.
- **Reduced drift** compared to MPU6050 [2].

Sensor Fusion

To convert the raw IMU data into usable angles, sensor fusion needs to be applied. Several methods and criteria were considered, and subsequently combined into a decision matrix:

Criteria

- **Ease of use**
- **Accuracy:** How closely the results match the real value.
- **Drift:** The severity of the error accumulated over time, must be negligible in a 10-minute time span.
- **Robustness:** Tolerance to vibrations and fast movements.
- **Computational efficiency:** Amount of processing done by the microcontroller.

Solutions

- **Complementary filter:** Combines sensor data using high and low pass filters, and weighted averaging.
- **Kalman filter:** An optimal recursive estimator that assumes the system is linear with Gaussian noise.
- **Extended Kalman filter:** A Kalman filter optimised for non-linear data (such as IMU data).
- **Madgwick filter:** Lightweight sensor fusion using quaternions to increase computational efficiency.
- **DMP:** Offloads sensor fusion onto a processor built into the IMU. It can be run at a high frequency to maintain accuracy, but with a low output data rate.

Criteria	Weight	Solution				
		Complementary Filter	Kalman Filter	Extended Kalman Filter	Madgwick Filter	DMP
Ease of use	2	0	-1	-1	0	1
Accuracy	3	-1	0	1	0	1
Drift	3	-1	-1	0	0	1
Robustness	1	-1	0	0	-1	0
Computational Efficiency	3	0	-1	-1	0	1
Total		-7	-8	-2	-1	11

Table 1 - Sensor fusion decision matrix.

Implementation and Testing

After the decision matrix, it was decided to proceed with both the Madgwick filter and DMP for further testing. Both were implemented; the Madgwick filter using the C library maintained by X-IO technologies [3] and the DMP using SparkFun's library [4]. Both were tested at the same time to compare their results (Figure 5Figure 5 - Pitch and Yaw test data). The test data was captured using Termite and analysed using Python scripts.

The Madgwick filter's yaw suffers from drift, while its pitch is similar to DMP. Madgwick also occasionally has glitches. With better calibration and implementation, it may have been possible to improve the Madgwick filter, but because of its massive computational advantages, it was decided to use the DMP instead.

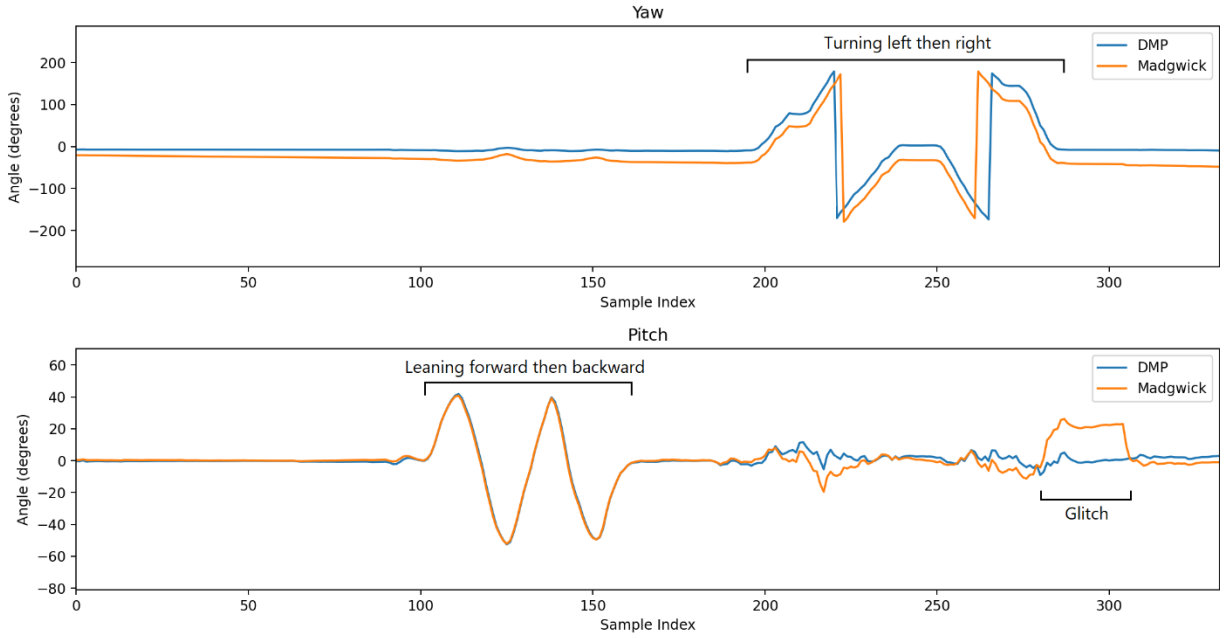


Figure 5 - Pitch and Yaw test data.

The yaw drift over an extended period is also a concern since the robot is required to map the maze in a maximum time of ten minutes. The yaw drift was tested both with and without the magnetometer for this period under different conditions.

Conditions	Yaw Drift (°/s)
No magnetometer, still	0.0011
No magnetometer, shaking	0.0416
With magnetometer, still	0.0007
With magnetometer, shaking	0.0094

Table 2 - Yaw drift in different conditions

Due to the results in Table 2 it was decided to use the magnetometer. This meant that over ten minutes the worst-case yaw drift is 5.64° which is acceptable, provided the other systems take this into account.

Time of Flight Sensors

The other sensors used by the robot (aside from the FPGA camera) are two wheel-mounted and one front mounted VL53L0X ToF sensors. These were selected because of Requirement 3.1.b, which bars the use of ultrasonic distance sensors. Light detection and ranging (LiDAR) sensors were considered but were too expensive. IR collision detection sensors were also tested but the range was found to be 5cm at most, too short for this application. Light dependent resistors were explored, but they suffered from poor directionality.

Figure 6 - Time-of-Flight sensor.

Overview

ToF sensors emit IR radiation and use a single photon avalanche diode array to detect the light that is reflected from a target. By performing calculations using the speed of light and time of flight of the emitted photons, it is possible to work out the distance between the sensor and any object up to 2000mm away. By mounting one on either side of the robot, close to the ground, it is possible to detect the LED strips marking the edges of the maze. This is useful for several purposes:

- When moving down a channel it is possible to stay in the centre of the channel by implementing a path controller that keeps the distances to the left and right walls constant.
- Junctions can be detected either side of the robot.
- When the robot is spun 180 degrees, a full distance map of the surroundings is created. This effectively turns the whole robot into a LiDAR sensor. By using the data revealed, it is possible to find all the passages out of a junction and their angles.

The front mounted ToF sensor is used for collision detection.

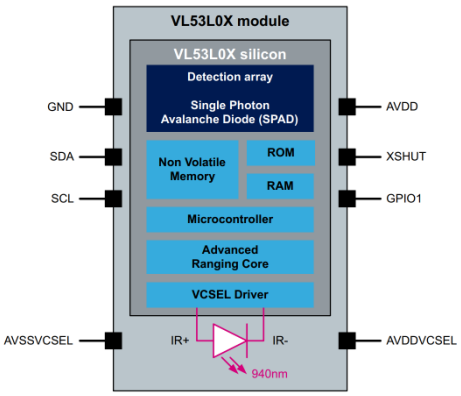


Figure 7 - ToF sensor internal architecture.

Implementation

The datasheet specifies a 25-degree conical field of view [5]. Initially they were mounted perpendicular to the ground which meant that when there was no object present within ~300mm, the sensors returned a reading of 90-100mm due to reflections from the ground. After experimenting, it was found that a 10-degree inclination was ideal for detecting the LED strips at up to 500mm without interference from ground reflections.

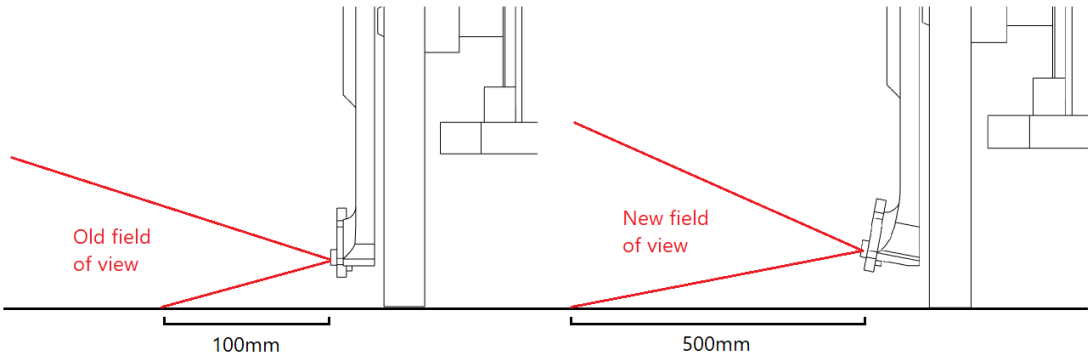


Figure 8 - Effect of mounting angle on field of view.

The software for the sensors was an Adafruit driver [6] that implements the manufacturer's application programming interface (API). Due to limited GPIO pins, the XSHUT pins (Figure 6) could not be connected to the microcontroller. To make the sensors work, part of the driver and manufacturer's API had to be rewritten to allow for a software reset.

To implement the junction detection and LiDAR mode, the signals generated by the sensor needed to be filtered to remove noise. This was done using a low pass finite impulse response (FIR) filter. An FIR filter as opposed to an infinite impulse response (IIR) filter was chosen because they are always stable and because the extra computation required was not an issue. The filter coefficients were chosen using TFilter for a sampling frequency of 20Hz. The passband gain is not important since absolute distance is not the goal, only relative distances. Once the filter had been designed, it was implemented in C++ using a circular buffer.

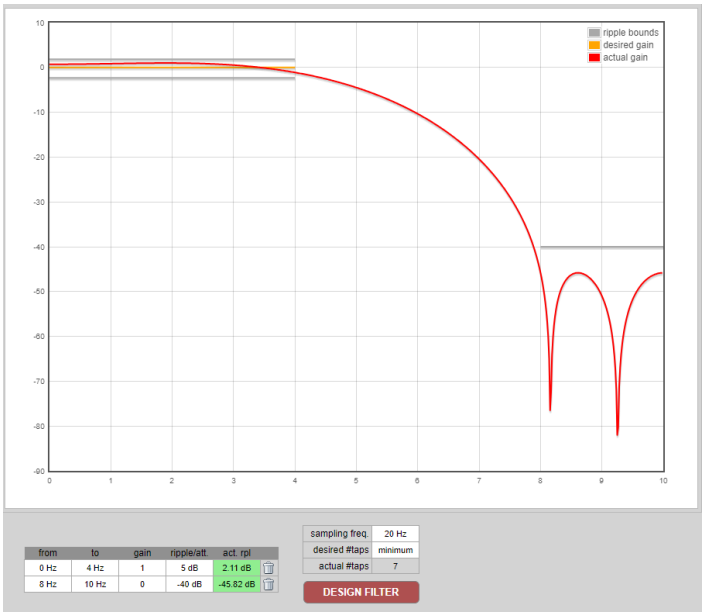


Figure 9 - Lowpass filter generated using TFilter.

Peaks in the filtered distance correlate to the sensor being pointed down a passage. By spinning the robot 180° and detecting these peaks, the number of passages out of a junction and the corresponding angles can be determined. Figure 10 shows the signal generated by spinning the robot at a crossroads junction. Four peaks can be clearly seen corresponding to the four branches.

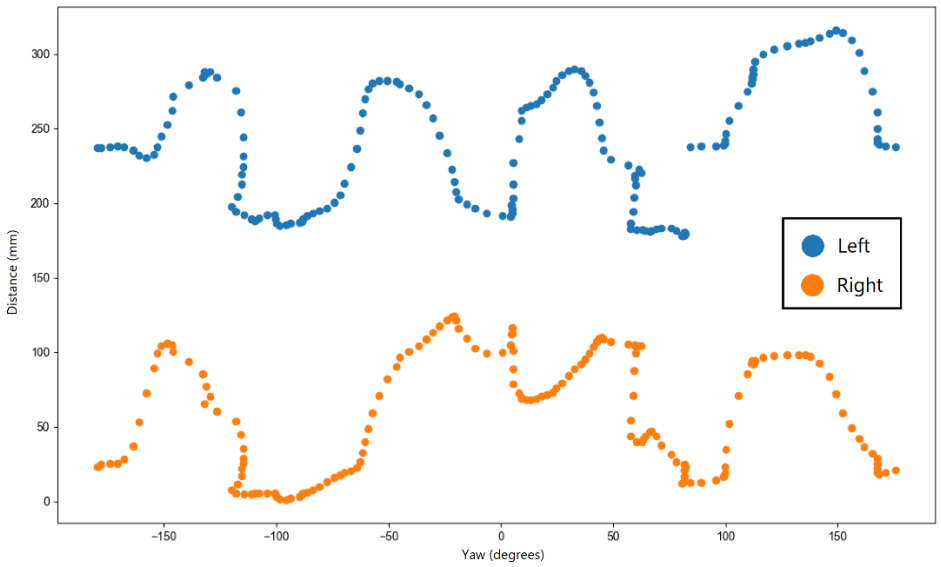


Figure 10 - ToF data gathered at a junction.

FreeRTOS

Overview

The robot is required to execute many tasks in parallel, including tasks with hard real time deadlines (collision detection) and tasks that needs to be run at a fixed frequency (control loops and sampling sensors). This means that a real time operating system (RTOS) is required.

FreeRTOS is the logical choice since it is pre-integrated into the ESP32 Arduino core. FreeRTOS also comes with several other benefits listed below:

- **Makes full use of the hardware:** The ESP32 has a dual core architecture, and the port of FreeRTOS uses symmetric multiprocessing (SMP), where one instance of the kernel can schedule tasks for both cores. This means with little overhead both cores can be used to run tasks simultaneously with few wasted clock cycles.
- **Real time concerns:** Tasks can be run at fixed frequencies and time-critical tasks can be assigned a higher priority to ensure they meet deadlines.
- **Allows the use of advanced structures** such as queues, task notifications, mutexes, and semaphores that aid task synchronisation and robustness.
- **Modularity:** Tasks can be split into individual source files that include the necessary header files, functions, and variables. This allows for encapsulation, maintainability, and testing.

Implementation

The problem was broken down into seven tasks:

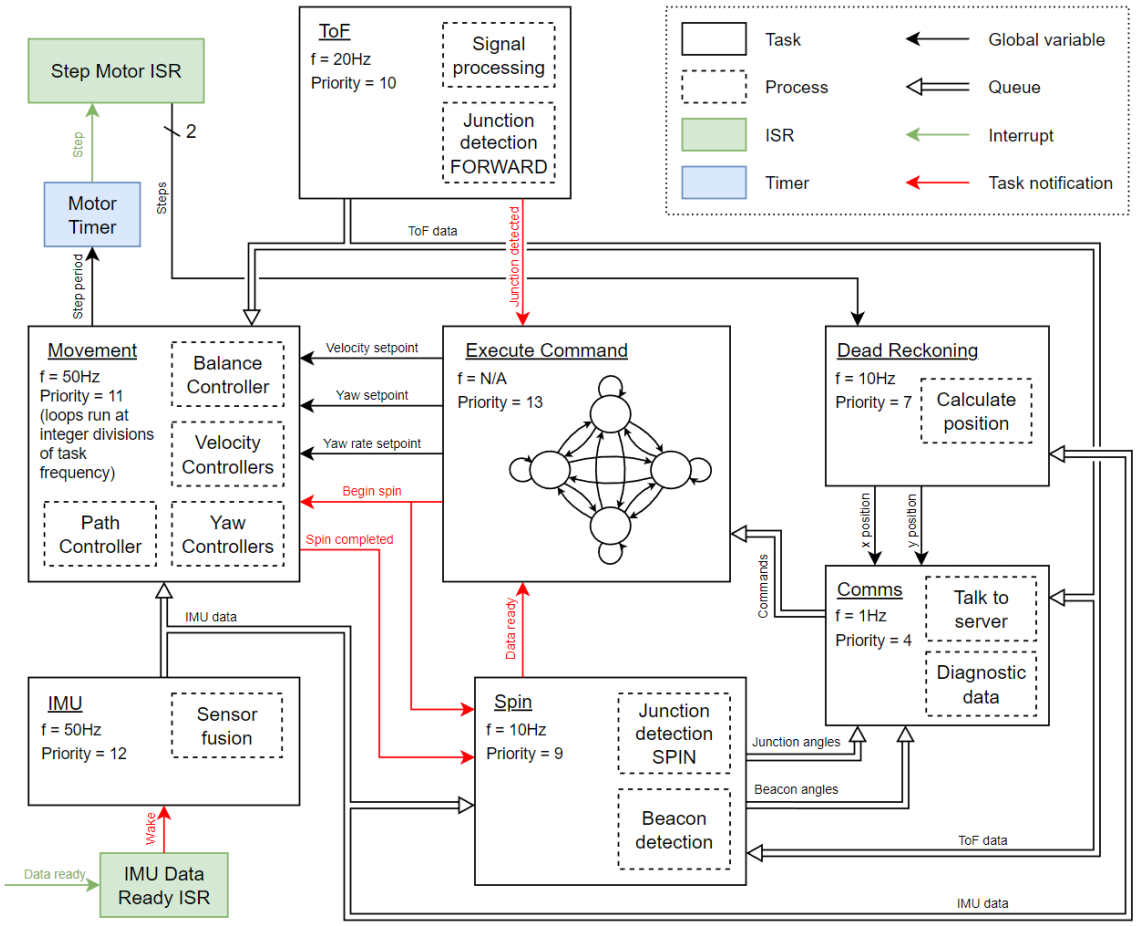


Figure 11 - Robot architecture decomposition and interfaces.

1. Execute Command Task

A state machine was implemented using a queue to control the robot shown in Figure 12.

State/command name	Function
IDLE	Set the speed to 0, balance and wait for new commands.
FORWARD	Move forwards at a set speed until a junction is reached or a wall is hit.
TURN	Turn to a specific angle.
SPIN	Turn 360 degrees, detect beacons and junctions.

Table 3 - Robot command functionality.

Once a command is complete, the next in the queue is executed. If the queue is empty, the robot returns to the IDLE state and waits for new commands. This default state is implemented so a failure from the server does not cause damage. Using these four commands, and oversight from the server which runs the mapping algorithm, it is possible to map the whole maze.

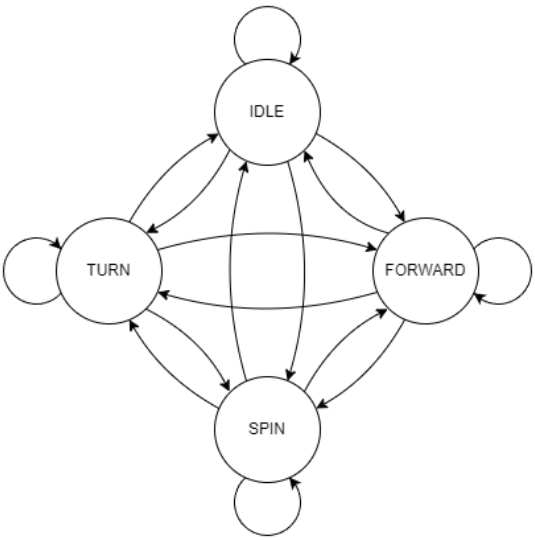


Figure 12 - Robot command state machine.

2. Server Communication Task

Telemetry data is sent to the server and commands are received using HTTP. Data is formatted as a JSON. Request formats can be found in Appendix G.

3. IMU Task

The IMU is sampled, and the data is processed into pitch, pitch rate, yaw, and yaw rate, which are added to a queue used by other tasks.

This task is unblocked by an interrupt service routine (ISR) which is triggered by a data-ready interrupt from the IMU.

4. Movement Task

Contains cascaded control loops, running at set frequencies to manage movement. The control scheme is described in the Balance Control subsection.

5. ToF Task

The ToF sensors are sampled, and the data is filtered and processed before being added to a queue for use by other tasks.

While the current command is FORWARDS, this task detects if there is a junction to the left or right, or if there is an imminent collision and alerts the command task if there is.

6. Spin Task

When the SPIN command is issued, this task is unblocked. It collects and processes data from the FPGA camera system and the ToF sensors to detect the angles of beacons and junctions.

7. Dead Reckoning Task

Odometry from the movement task and the yaw from the IMU task are used to calculate the position of the robot. The inherent drift is compensated for when the robot triangulates its position using the beacons.

Concurrency Issues

Using an RTOS comes with extra concerns regarding the interactions of tasks. Potential failure modes and the methods used to mitigate them are described.

Priority inversion

This occurs when a higher priority task must wait for a lower priority task to give up a resource such as a semaphore or mutex. In the code, the only occurrence of this is that both the ToF sensors and FPGA use I2C (protected by a mutex) and are sampled in different tasks. This was solved by implementing the priority ceiling protocol [7] to prevent the higher priority task from pre-empting the lower priority task while it has the mutex.

Race conditions

These occur when data is modified concurrently by multiple tasks, leading to unpredictable or damaging behaviour. To prevent this, global variables are only driven by one task, and important information such as commands and IMU data are sent in queues.

Other measures

All variables shared between tasks use the volatile keyword and the scope of all variables is limited to as large as it needs to be. The code has been analysed for other concurrency issues such as deadlocks, but no potential occurrences were found.

Balance Control System

Overview

According to Jimenez [8], "the Two Wheeled Automatic Balancing Robot (TWABR) is similar to an inverted pendulum". The system was modelled in two dimensions as the robot could be turned by generating a differential in the rotational speed of the two wheels. The two-dimensional TWABR system is modelled below (Figure 13):

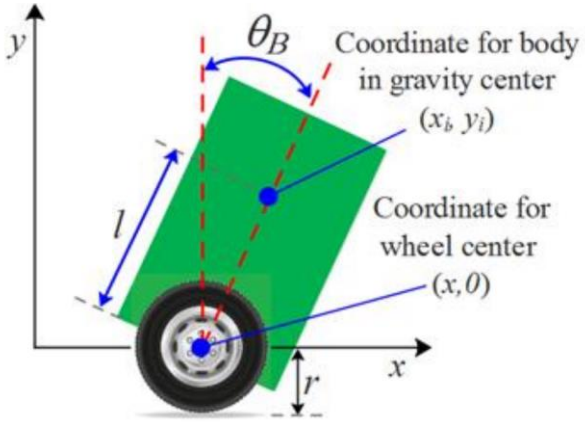


Figure 13 - Physical analysis of TWABR. [8]

From Figure 13, it can be deduced that due to the single controlled variable x, multiple cascaded controllers would be needed.

Several existing controllers, previously implemented for this use case [9], were considered: Linear-Quadratic Regulator (LQR), Fuzzy Control, Pole Placement i.e. Full State Feedback (FSF), Model Predictive Control (MPC), and Proportional–Integral–Derivative control (PID). An explanation of each method is included in Appendix A.

LQR

LQR was considered primarily for its high theoretical performance, given an accurate model representation of the TWABR system. The gain coefficient matrix can be calculated on a powerful computer and hard coded into the microcontroller. Thus, LQR maintains a low computational cost.

Fuzzy Control

Fuzzy control was considered due to its high precision without the need for an accurate mathematical model, whilst maintaining a low computational cost. However, there is an added implementation cost creating and tuning numerous rulesets to get a functional controller.

FSF

FSF has the same pros and cons as LQR, with low computational cost, and requiring an accurate model of the TWABR system. However, FSF cannot be adjusted to optimise for actuator cost or performance but has greater robustness to model inaccuracies.

MPC

MPC would give the most accurate results while providing more robustness against model inaccuracies and non-linearity. However, the computational cost of having to run an optimisation function on the system for every control loop makes it infeasible to implement in this use case.

PID

PID is the easiest to implement but provides the worst performance. It has the advantage of not requiring a mathematical model of the system. There could be added implementation costs of tuning the controller, but this cost can be reduced by obtaining rough values through tuning methods such as the Zeigler-Nichols Method [10] and fine-tuning the coefficients.

Decision criteria

A decision matrix was used to select the optimal method.

The highest priority was to implement a minimally viable control system which would be ready in time for the demonstration. This meant any controllers requiring mathematical models of the TWABR system were heavily penalised as re-characterising the system for every iteration of the robot would be time-consuming.

The high weighting of computational cost was to reduce the load on the microcontroller. This provides ample overhead for server communication and sensor processing which are particularly computationally expensive.

Criteria	**Weight**	**Solution**				
		LQR	**Fuzzy**	**FSF**	**MPC**	**PID**
Computational Efficiency	3	4	5	4	1	5
Implementation Efficiency	2	2	2	3	1	4
Performance	1	4	3	3	5	1
Total		20	22	21	10	24

Table 4 - Control scheme decision matrix.

Cascading PID Controllers

After the decision matrix it was decided to proceed with cascaded PID control. The first controller to be implemented was the pitch controller, which would keep the robot balanced whilst being the foundation for all other controllers. To implement this, motor speed would need to be accurately set.

Motor Controller

The supplied NEMA17 stepper motors and motor drivers step the motor every time a pulse is supplied to the STEP pin on the driver [11]. The size of each step can be altered with microstepping to increase precision. Each step moves the motor $\frac{1.8}{X}°$ where X is the microstepping factor up to 16. Available libraries

such as AccelStepper and the Arduino Stepper library did not have adequate performance, as they caused the control loops to stutter when implemented with the other tasks. This required the creation of a bespoke motor controller. A hardware timer of variable frequency was initialised on the microcontroller that would trigger an ISR every tick. This ISR delivered a pulse to the motor driver and updated the total steps for that motor (for odometry). To keep this ISR fast, the direction was assigned separately. The necessary hardware timer frequency is calculated through the formula $freq. = 200 * microsteps * \frac{speed\ (DPS)}{360}$. The timer is reinitialised with this new frequency. Assuming no steps are skipped, the motor will achieve the set speed.

Speed and acceleration measurements

The linear speed of the robot was estimated using the current setpoint speed of the motors and accounting for any rotation of the robot around the y axis. As both the motor setpoint and angular velocity from the IMU are defined in *degrees per second*, the linear velocity can be calculated:

$$v_{est} = \frac{motorspeed_L + motorspeed_R}{2} + \dot{\theta}$$

A low pass filter was used to reduce the high frequency noise. This value can be converted to linear speed using $v = \frac{v_{est}}{360} 2r\pi\ mm/s$. With $r = 91\ mm$, $v \approx \frac{v_{est}}{360} * 572\ mm/s$.

The linear acceleration was calculated by differentiating the linear velocity.

Tuning Methods

Several tools were used to fine-tune the PID gains from the rough values calculated by Ziegler-Nichols auto-tuning [12]. Serial communication was established to provide real-time debugging graphs (through Serial Studio) and to allow PID gains for a particular loop to be changed without recompilation. The tuning method employed increased K_p until constant oscillations were observed, then increased K_d until the oscillations were as small as possible. This process was repeated until results no longer improved. A small K_i term was then introduced to improve static precision.

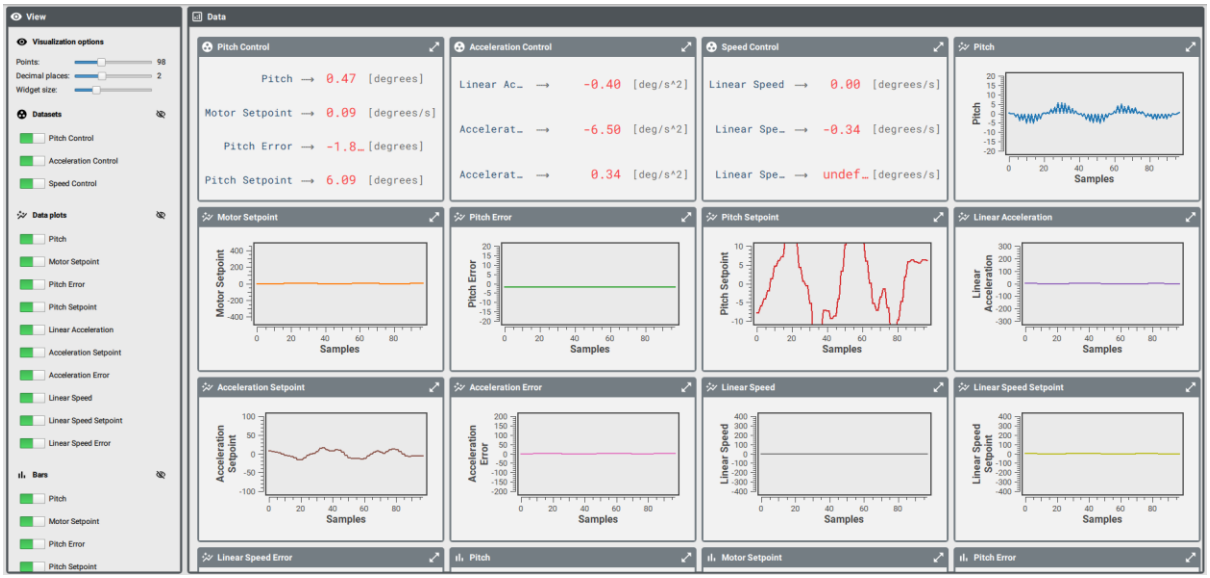


Figure 14 - Real-time graphs used to tune PID gains.

Pitch Controller

Pitch control was implemented using a PID controller to control the acceleration of the motors. Initially, setting the motor speed directly was attempted, but the motor lacked the torque to fulfil the large, almost instantaneous changes in motor setpoint. The controller is run at 50Hz due to the IMU sampling rate.

Initial PID coefficients were obtained by using an autotuning library for Arduino [12], which produced values of $K_p \approx 4, K_i \approx 0, K_d \approx 0.05$. After the fine-tuning process, the PID coefficients were $K_p = 7.00, K_i =$

$0.01, K_d = 0.4$. This satisfied the implementation's performance requirements as the error rarely exceeded more than 1° from the setpoint during testing. The graph in Figure 15 shows the pitch response to an impulse at 250 samples.

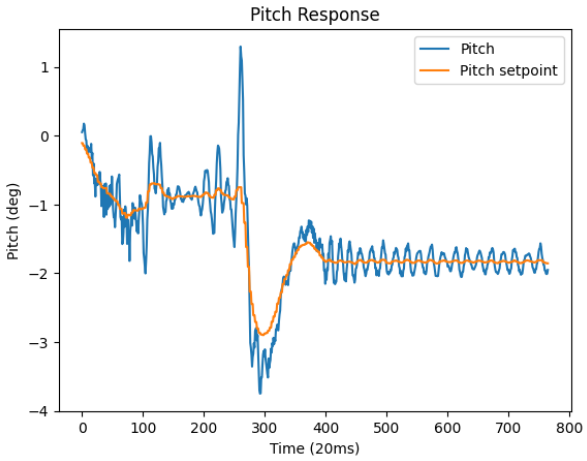


Figure 15 - Pitch response to setpoint.

Acceleration Controller

After analysing the free-body diagram (Figure 13) an expression for linear acceleration (a_{lin}) can be obtained:

$$M_{cw} = mgh \sin\theta$$

$$M_{acw} = Fr = ma_{lin}r$$

To stabilise the robot, $M_{acw} = M_{cw}$ so

$$ma_{lin}r = mgh \sin\theta$$

$$a_{lin} = \frac{gh \sin\theta}{r}$$

Since the robot operates in a small region centred around $\theta = 0°$, the approximation $a_{lin} \approx \frac{gh}{r}\theta$ can be used. Since g, h and r are all constants, linear acceleration can be approximated as directly proportional to the angle deviation from the robot's equilibrium angle. This allows controlling acceleration using a PID controller. During testing it was found that this controller could be run slower than the pitch controller. The lowest frequency without performance degradation was 16.7Hz.

The input for this controller was the linear acceleration, derived above, and the output was a setpoint for the angle controller. The tuning method was then applied giving PID values of $K_p = 0.04, K_i = 0.00, K_d = 0.000025$ which produced the response in Figure 16 to an impulse at 250 samples. Note the increased response time due to the application of an FIR filter.

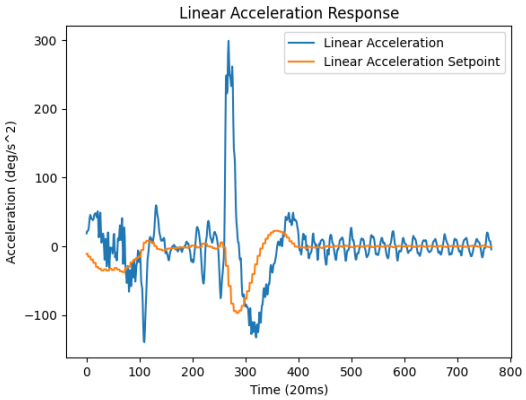


Figure 16 - Linear acceleration response to setpoint.

Speed Controller

The speed controller was implemented as a PID controller with estimated linear speed as its input, and desired linear acceleration as its output. This controller was run as slow as possible without performance degradation (10Hz), to reduce computational cost.

After tuning, PID gains of $K_p = 0.04, K_i = 0.00, K_d = 0.000025$ were used to achieve the response (shown in Figure 17) to an impulse around 250 samples. The initial peak is the system correcting for the placement pitch.

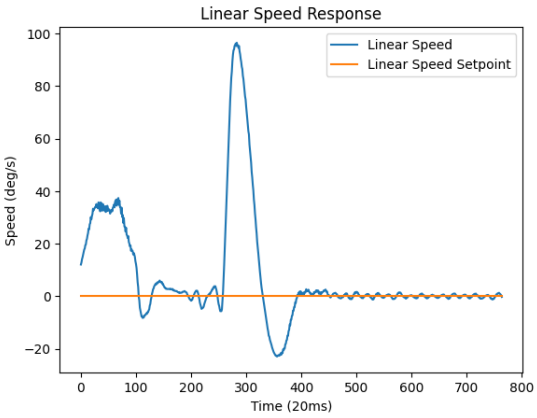


Figure 17 - Linear speed response to setpoint.

Yaw rate and Yaw controller

The yaw and yaw rate could be controlled by inducing a differential in the speed of the left and right wheels. A PID controller was tuned to output motor differentials with the input as yaw rate from the IMU. The yaw controller takes yaw as the input and outputs a desired yaw rate to the yaw rate controller.

Complete Control Architecture

The complete control block diagram (Figure 18) describes all the cascaded control loops. Two sets of cascaded controllers were used: one for controlling the linear movement of the robot and another to control the direction of the robot. Using these in conjunction, two degrees of freedom can be achieved, allowing traversal anywhere on the maze.

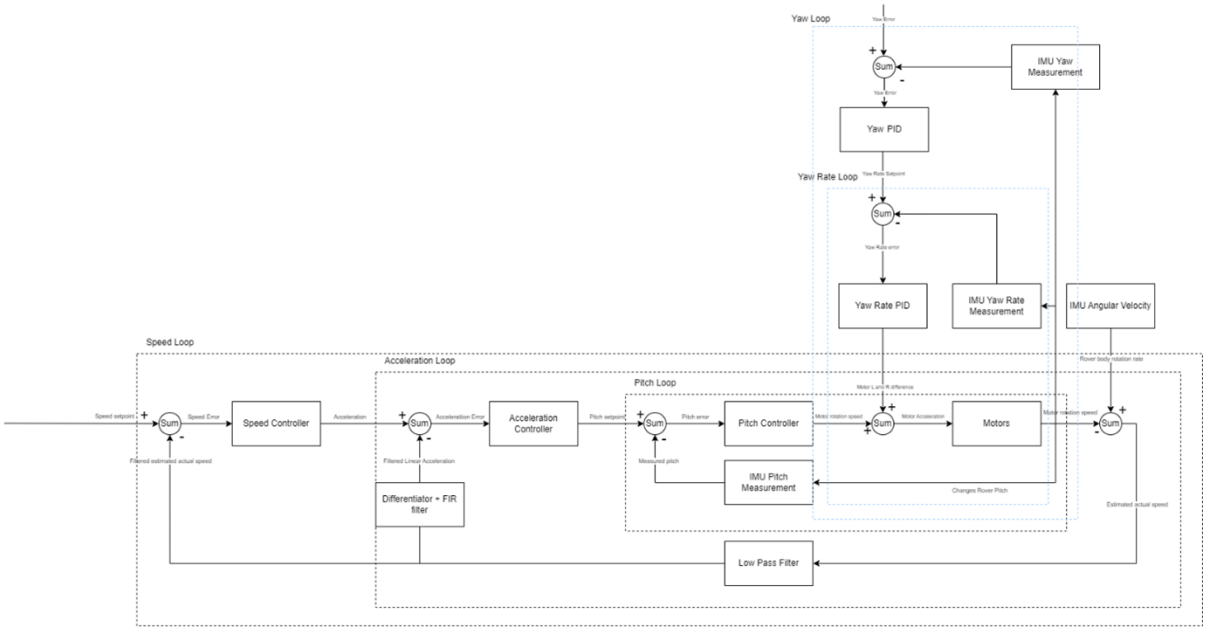


Figure 18 - Control architecture block diagram.

Image Processing

Overview

The robot must use the beacons to triangulate its position. The three beacons' colours are red, yellow, and blue. To detect the colours, the Hue, Saturation, Value (HSV) colour space was used.

FPGA

Pixel detection

Initially, the RGB colour space was used for detection. It showed promising results when detecting non-glowing red, green and blue objects. However, several problems were encountered when detecting glowing beacons:

- Assigning thresholds for luminous colour was difficult as RGB is insensitive to colour variation due to brightness.
- Minimising the effect of noise from the background light source was challenging.

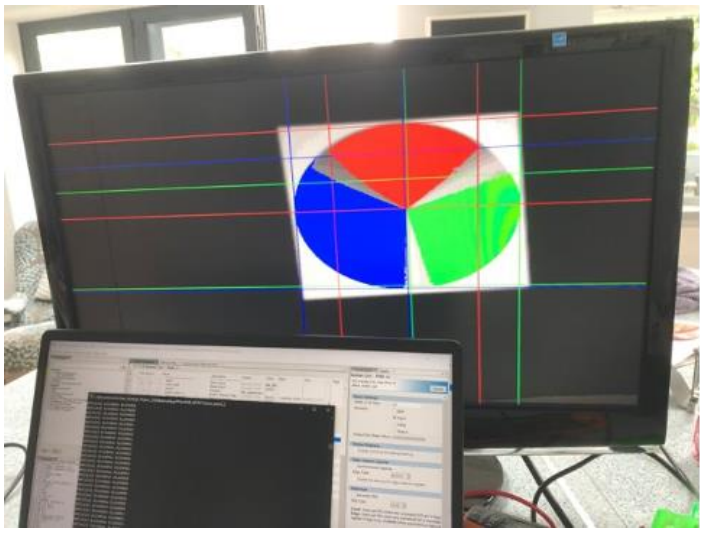


Figure 19 - FPGA detecting red, green, and blue with RGB colour space.

Due to these shortcomings and the following reasons, the HSV colour space was used:

- Wider complexity of colour representation.
- Improved precision for bright lights.
- Even when the lighting conditions and brightness vary, the HSV scale showed improved colour-based segmentation.
- Significantly better at eliminating undesired tints and noise.

To use the HSV colour space the RGB values must be converted into HSV. The formulae used for this conversion are shown below:

$$R' = \frac{R}{255}$$
$$G' = \frac{G}{255}$$
$$B' = \frac{B}{255}$$
$$Cmax = \max(R', G', B')$$
$$Cmin = \min(R', G', B')$$
$$\Delta = Cmax - Cmin$$

$$H = \begin{cases} 0^{\circ} & ,\Delta = 0 \\ 60^{\circ} \times \left(\frac{G' - B'}{\Delta} mod 6\right) & ,Cmax = R' \\ 60^{\circ} \times \left(\frac{B' - R'}{\Delta} + 2\right) & ,Cmax = G' \\ 60^{\circ} \times \left(\frac{R' - G'}{\Delta} + 4\right) & ,Cmax = B' \end{cases}$$

$$S = \begin{cases} 0 & , Cmax = 0 \\ \dfrac{\Delta}{Cmax} & , Cmax \neq 0 \end{cases}$$

$$V = Cmax$$

A module was used which receives RGB values and converts to HSV values [13]. Colours were then detected by setting thresholds for each component.

The threshold values were determined through trial and error. When the camera detects pixels within the thresholds, the detected area will be highlighted with corresponding colour, allowing for easy confirmation.

Beacon detection

A regular spacing between objects does not correspond to the same angle between those objects due to the distortion of the camera lens. To reduce processing, it is sufficient to only detect when a beacon is at the centre of the image, at a known angle. To do this, the average location of all the pixels of each colour can be found, and when this average is near the centre of the image, a beacon has been detected at the robot's yaw.

Figure 20 - Detecting the yellow beacon.

To find the average location, the FPGA finds the sum of all x coordinates and the number of pixels of a colour. These are sent to the ESP32, which applies a threshold and divides the former by the latter. This is because arithmetic division in RTL will cause the synthesizer to instantiate a divider circuit, which requires multiple clock cycles and is hard to implement. The flowchart for this can be found in Appendix B.

The decided strategy requires the robot to detect beacons, at furthest, along the diagonal of the maze (4.3m). The zoom is set to maximum to achieve this.

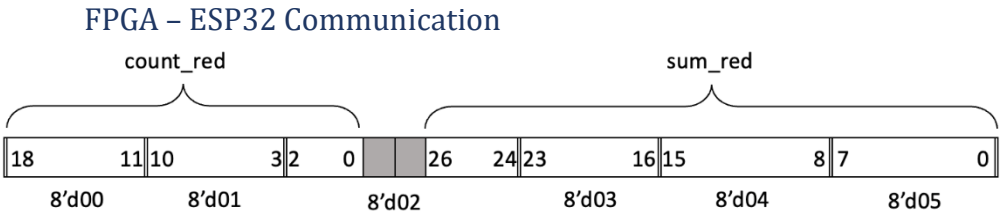


Figure 21 - Register diagram for red

An I2C bus was used to transmit data from the FPGA to the ESP32 because it is fast (unlike UART) and available (unlike SPI). The ESP32 is the bus master and polls the FPGA at regular intervals (10Hz) while the robot is spinning. A driver was written to read the registers and convert the values into useful data for determining the position of the beacons. The full register map and driver flowchart can be found in Appendix C and Appendix D, respectively.

When attempting to communicate with the FPGA, data would frequently be corrupted, and the FPGA would sometimes not respond to being called by its address. It was found after testing that setting the IO standard of the serial clock (SCL) pin to a 3.3V Schmitt trigger [14] fixed the issue. This is because the SCL signal is the clock for the logic in the I2C slave module and a Schmitt trigger is better at detecting the rising or falling edge of the signal, which it was previously missing. This is a result of the Schmitt trigger's hysteresis being immune to the noise on the bus.

Triangulation

Due to dead reckoning being prone to drift, the robot regularly needs to recalibrate its position. To do this, it spins at every junction to find the angles to each of the three LED beacons. The server uses these angles to update the robot's position using triangulation.

Using the yaw angles to each of the three beacons, the server uses standard trigonometric formulae to find the robot's position, given that the locations of the beacons is known. Beacons were placed at the corners of the maze and the system was tested, the results of which are shown in Figure 22.

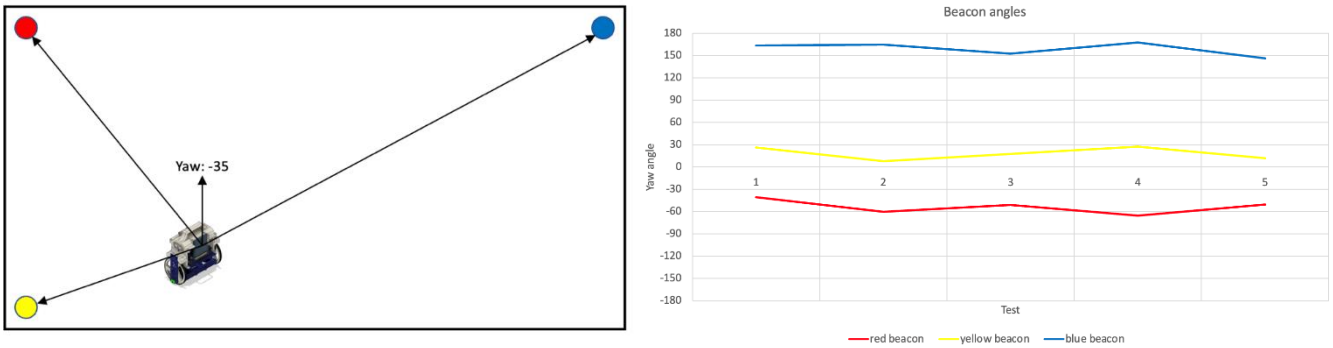


Figure 22 - Beacon triangulation test.

One potential failure mode is if no beacons are detected during a spin. In this scenario, the system is programmed to continue spinning and attempting triangulation until all beacons are detected or a maximum number of attempts is reached.

Another possible failure mode is IMU drift producing inaccurate angle-to-beacon data. The triangulation algorithm compensated for this by using gradient descent. By iterating an offset of the measured angles and its effect on the calculated coordinates, triangulation error reached a local minimum. This numerical method was used rather than an analytical method for two reasons:

1) A global minimum does not need to be found, which reduces implementation and computational complexity.
2) The drift is small enough that the numerical method converges much faster than the analytical method.

Power

Overview

The beacons were designed to be powered by photovoltaic (PV) panels through a DC grid. To turn on the beacons continuously and consistently, various power electronics and control techniques were implemented.

This problem was divided into three main parts:

- Characterising the PV panel at different light intensities.
- Drawing maximum power from the PV panels.
- Maintaining constant brightness in the beacons.

Characterisation

To find the I-V (current-voltage) and P-V (power-voltage) characteristics of the PV panel, a boost switch mode power supply (SMPS) was used to draw different powers from the panel as shown in Figure 23. The current and voltage data was collected for different light intensities and was analysed.

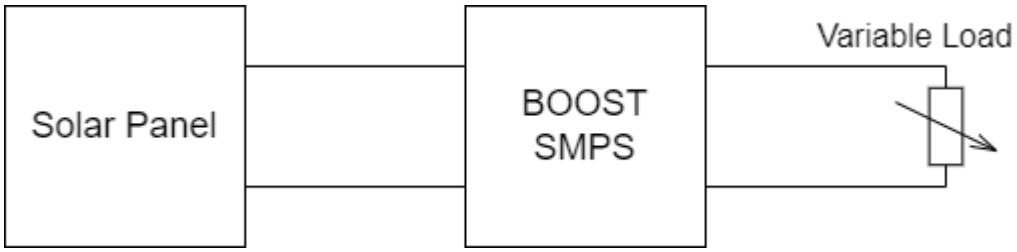


Figure 23 - Structure diagram of characterisation circuit.

The first test was performed by using single PV panel under sunny conditions to get the general shape of the I-V and P-V curves. The results are shown in Figure 24.

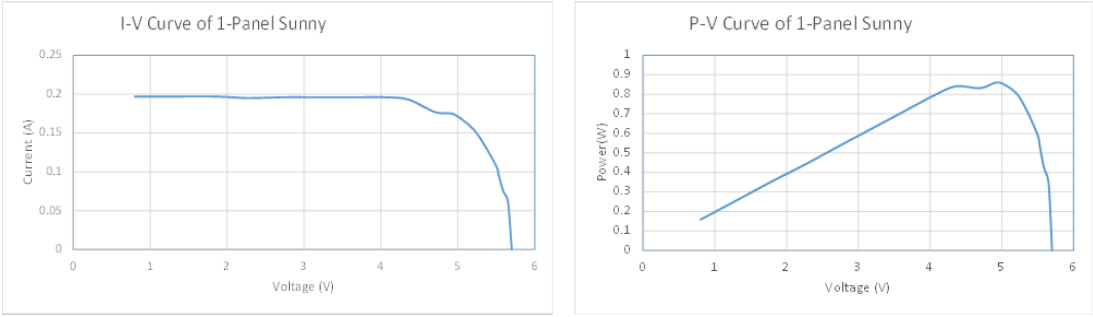


Figure 24 - I-V and P-V characteristics of single PV panel.

The maximum current that can be drawn from a single panel under these conditions is around 200mA. The maximum power point occurs when the input voltage is around 5V.

The second test was carried out using three panels in parallel under three weather conditions: very sunny, relatively sunny, and cloudy, as shown in Figure 25.

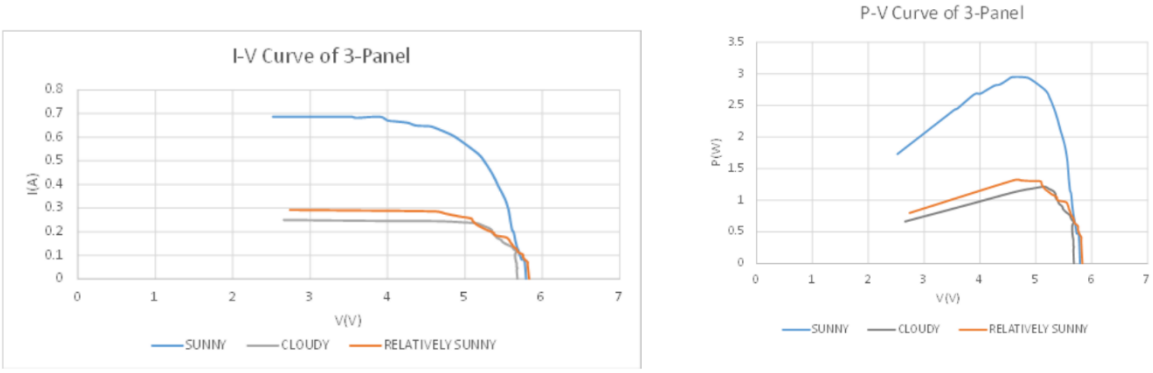


Figure 25 - I-V and P-V Curves of 3 PV Panels

The maximum current was tripled in the sunny case, as was the power. The maximum power points for all three cases appears around 5V and power will drop significantly if there is not enough sunlight.

Further tests using a power supply (in place of the PV panel) were conducted to emulate the conditions of the demonstration. The results and explanations of this experiment can be found in Appendix E.

The beacons were also characterised to find their forward voltages (shown in Table 5 and Figure 26):

Colour	Forward voltage drops (V)
Red	1.78
Yellow	1.83
Blue	2.49

Table 5 - Forward voltage drop of LED beacons.

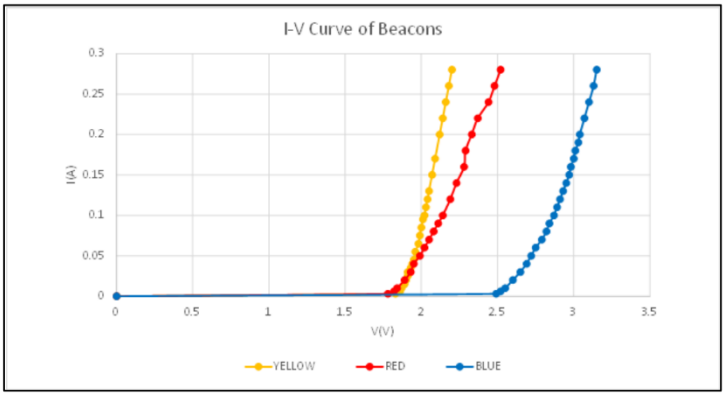


Figure 26 - I-V Characteristics of LED Beacons

The maximum power for all three beacons was limited to 1W (300mA at 3V) to avoid damage. Throughout testing, each beacon needed at least 0.5W of power to enable colour detection using the FPGA camera. An extra SMPS was required to drive each beacon's LED and control the power flow, which will be elaborated upon in the Beacon Driving subsection.

Maximum Power Point Tracking

After the PV panels had been characterised, it was concluded that for each light intensity there exists an operating point that delivers the maximum power. To ensure maximum efficiency, it is essential to operate the panel at this maximum power point. To achieve this, a maximum power point tracking (MPPT) algorithm needed to be applied. Common MPPT algorithms include:

1. **Constant voltage:** A constant voltage reference is set, corresponding to the maximum power output. The SMPS matches the output voltage to the voltage reference by adjusting the duty cycle. This method is simple to apply but gives poor performance when the light intensity varies.
2. **Incremental conductance:** The maximum power point is tracked by comparing the previous operating point to the current operating point. The current and voltage derivatives are calculated, and the duty cycle is adjusted accordingly. Although this method gives the best tracking performance, in this case it is difficult to implement since calculating derivative terms requires high accuracy in the sensing device.

3. **Perturb and observe (P&O):** The duty cycle is constantly varied by a certain amount. Similar to method 2, the operating points are then compared to find their powers. The next duty-cycle change is in whichever direction increased the power. Although this method results in oscillations around the maximum power point, the method itself is implementable, and the performance will not be affected by the oscillations.

Having compared these three algorithms, it was decided to employ the Perturb and observe algorithm due to its simple implementation.

System design

The MPPT system contains three blocks described in Figure 27.

Voltage Limiting

As the voltage limit for the supercapacitor and LED driver is 18V [15] [16], it is reasonable to restrict output voltage of the boost SMPS to 16V as a precaution. If the output voltage is in the safe range, the system will enter the P&O stage.

P&O Algorithm

By comparing both power and voltage samples, the position of the current operating point can be obtained. Owing to the P-V and I-V characteristics of the PV panel, the duty-cycle can be adjusted to make the operating point match the maximum power point (Figure 28).

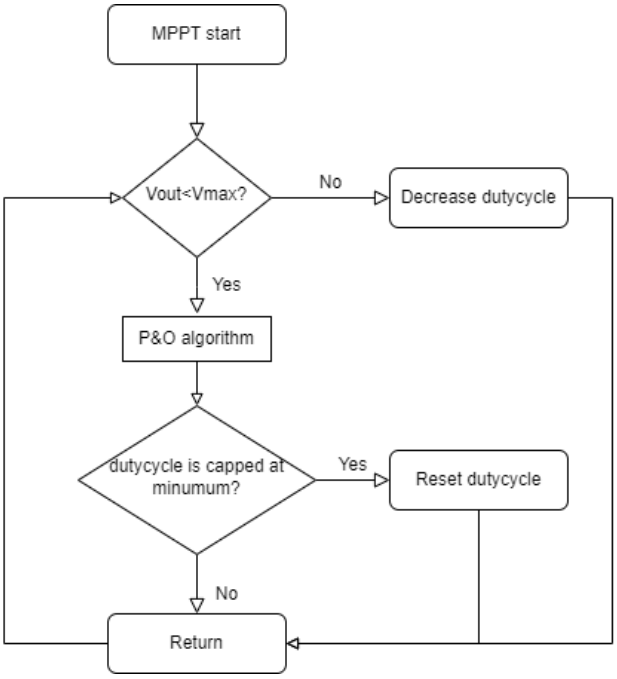


Figure 27 - MPPT System Flowchart

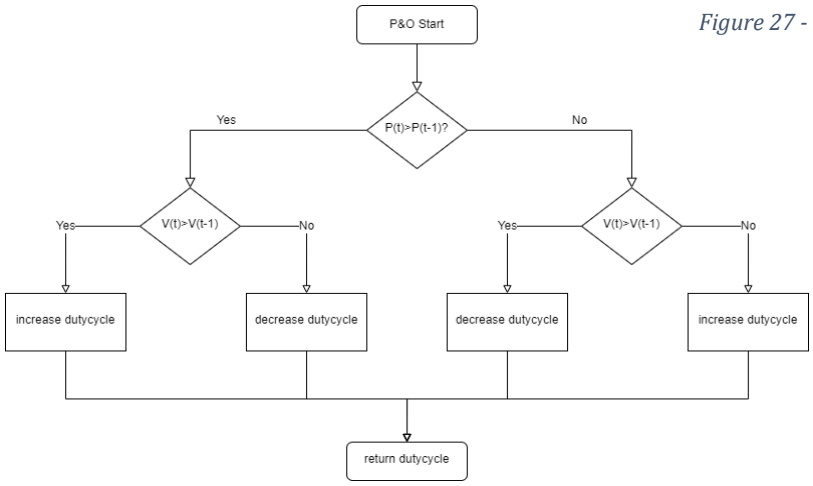


Figure 28 - P&O algorithm

Self-activating algorithm

The issue of this system was that there exist several local maximum power points. To locate the global maximum power point, the self-activating algorithm is needed. This algorithm will record the duty-cycle 1000 samples ago and compare it with the current sample. If both duty-cycles are set at the minimum, the duty-cycle will be reset to its initial value. This enables the system to relocate the maximum power point.

Implementation

Tests were carried out under sunny condition using three PV panels connected in parallel. A constant 3.3W of power (with oscillations) was drawn by the MPPT Boost SMPS as shown in Figure 29.

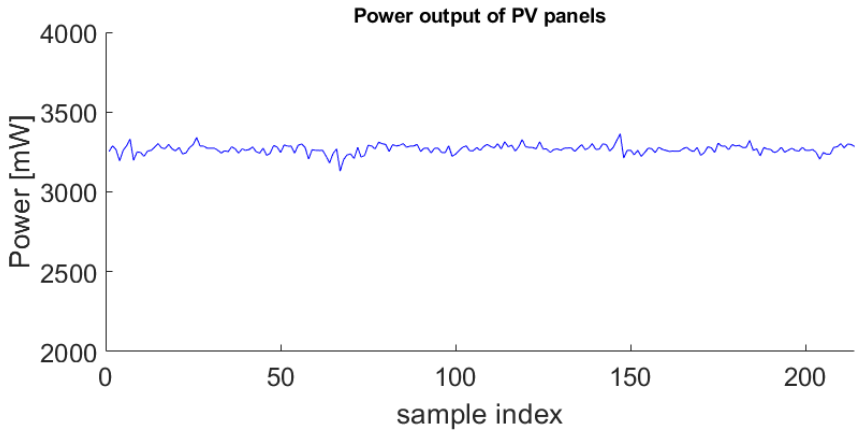


Figure 29 - PV panel output testing results by applying MPPT.

Beacon Driving

As the output from the MPPT SMPS has a maximum of 16V, a buck SMPS is needed to lower the grid voltage and drive the beacon. This buck SMPS also allows control of the beacon power. Three closed loop methods of power control were developed:

Voltage Control

This method requires a fixed voltage reference. The duty-cycle is set to increase based on a proportional relationship if the output voltage is smaller than the voltage reference. A current limit will also be implemented to avoid high current flow into the beacons, as seen in Figure 30.

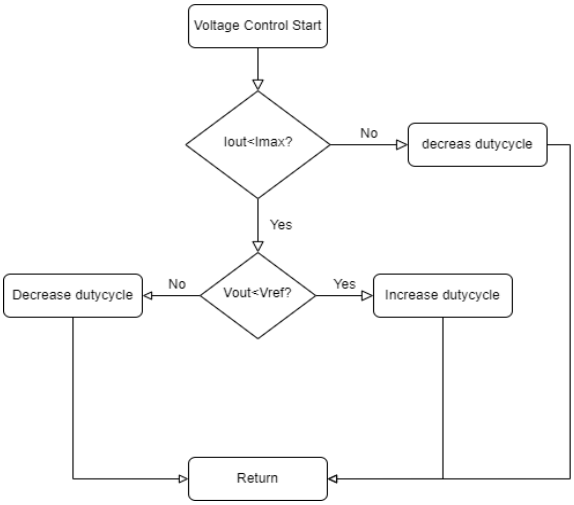


Figure 30 - Voltage control flowchart.

This method has the drawback that the final output voltage will oscillate around the voltage reference. This is because according to the I-V characteristics of the beacons, a small change in voltage will result in a large change in current, and therefore power.

Current control

This method is an improved version of voltage control as shown in Figure 31. By using the output current as the reference, it achieves better control of power flow due to the steep gradient in Figure 26.

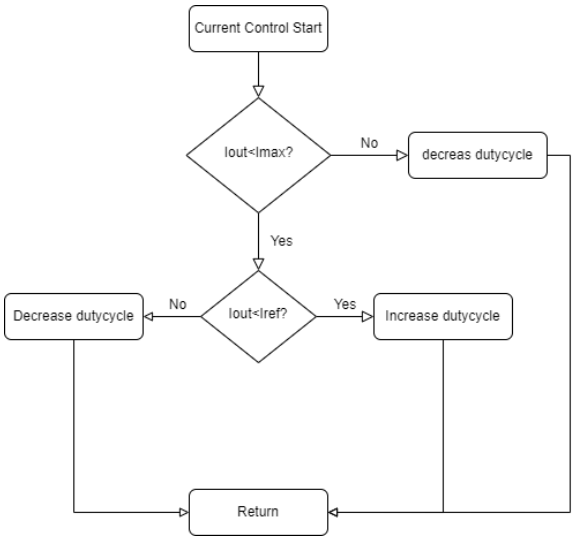


Figure 31 - Current control flowchart.

PID Control:

In this method, current is controlled using a PID controller as shown in Figure 32.

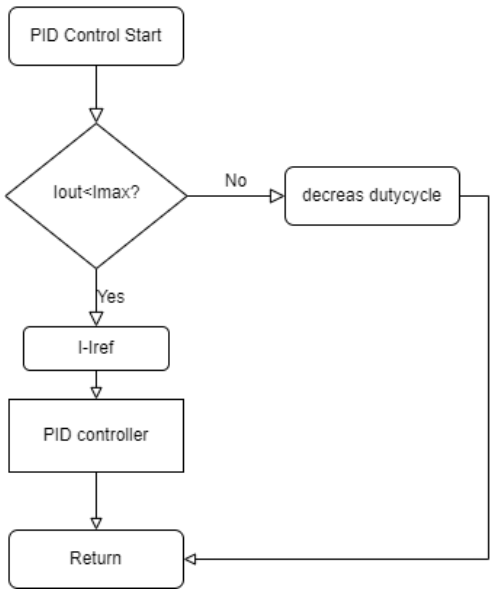


Figure 32 - PID control flowchart.

Implementation

From testing, both current control and PID control were feasible. The final decision was to use PID control as it draws relatively constant power and is constantly on at lower voltage (Table 7).

The current reference was found experimentally for each colour as shown in Table 6 to optimise detection while drawing minimum power.

Colour	Current reference
Red	90mA
Yellow	130mA
Blue	50mA

Table 6 - Optimal current reference of each beacon.

The beacon driver can also carry out wireless communication using Wi-Fi. This will be elaborated upon in the Final Strategy subsection.

DC Grid

As the power of the PV panel is continuously varying, the correct behaviour of the beacons at a low PV panel output power cannot be guaranteed. As a result, extra power needs to be supplied to the DC grid in these conditions. To achieve this, a supercapacitor was connected in parallel with the beacon drivers to store energy in sunny conditions and release it in cloudy ones. A bidirectional SMPS was planned to be connected in series with the supercapacitor to control its charging and discharging.

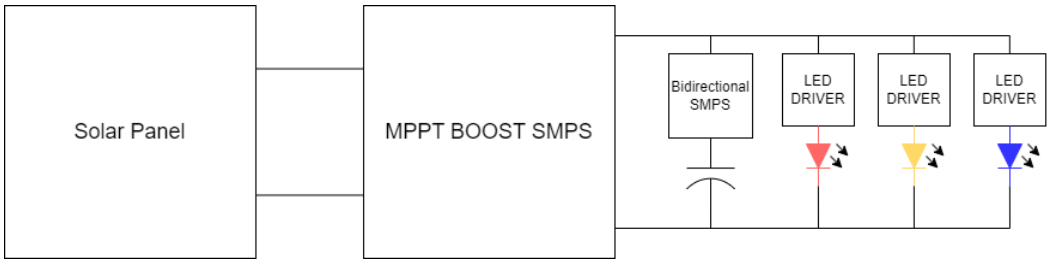


Figure 33 - Structure diagram of the DC grid.

The entire power system was tested at different PV panel output powers and the results are shown in Table 7.

PV panel output power(mW) using current control	Status	PV panel output power(mW) using PID	Status
0	off	0	off
400-500	blinking fast	400-500	blinking fast
800-900	blinking fast	800-900	blinking fast
1300-1400	blinking	1300-1400	constantly on
1800-1900	constantly on	1800-1900	constantly on
1900 above	constantly on	1900 above	constantly on

Table 7 - System testing results with different output power from PV panels.

From these test results, the power required for the beacons to function is approximately 1.5W. Therefore, a power threshold for the bidirectional SMPS is set at 1.5W. If the power at the PV panel output is greater

than the threshold, the bidirectional SMPS should turn on and work as a buck SMPS to charge the capacitor. Otherwise, the supercapacitor will discharge through the bidirectional SMPS, which works as a boost SMPS.

Final Strategy

As the bidirectional SMPS did not arrive on time, the final strategy chosen was to use Wi-Fi to control the power flow. Beacons are only turned on by the server when the robot needs to triangulate its position. This was achieved by controlling the beacon driver's PWM enable signal through HTTP requests. The system will charge the supercapacitor when the beacon drivers are off. A failure mode needs to be considered when the supercapacitor does not have enough stored energy to turn on the beacons when requested. In this case the beacon drivers will then notify the server that they were unable to power the beacons and that the server should retry when the supercapacitor is charged above certain threshold.

Maze Computation

Maze Traversal

To map out the entire maze efficiently, a depth-first search algorithm, known as Trémaux's algorithm, was utilised. Trémaux's algorithm divides the maze into nodes which are placed either at junctions or at the ends of passages and connects adjacent nodes with edges to form a graph. If a node which has previously been visited is visited again, it is treated as a dead end and the robot is told to go back to the previous node.

Other algorithms considered:

- **Wall follower:** sticks to a wall and follows it; rejected due to circling loops continuously.
- **Blind mouse:** chooses a random path at every junction, but unnecessarily repeatedly traverses junctions.
- **Flood fill:** attempts to takes the shortest path and redirects on contact with walls; rejected for not mapping out the full maze.

Initially, Trémaux's algorithm was implemented recursively. However, this did not work as Flask API routing is not compatible with recursive subroutine calls. The solution was to rewrite it as an object-oriented state machine. States are prepended to a queue and outputs are decided based on the current state, prior state, and telemetry data. The object-oriented approach allows for scalability – several robot instances can be instantiated server-side and traverse their own mazes simultaneously using the same algorithm.

Moreover, the algorithm was modified to consider curved passageways and junctions. Visualising these structures to meet Requirement 1.2 demands storing telemetry data more often than Trémaux's algorithm requires, which can result in erroneous "go back" instructions. To compensate, the maze traversal algorithm creates nodes at regular intervals within passageways without updating its prior node. This allows frequent telemetry data to be stored in the tree without affecting the algorithm, allowing a more accurate visualisation.

Several tests were conducted with virtual mazes to test the validity of this algorithm, summarised in Table 8.

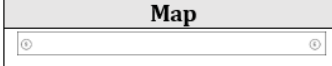
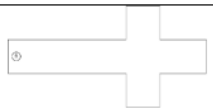
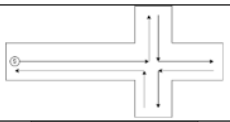
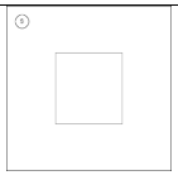
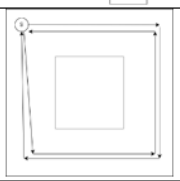
Map	Path taken	Reason for test	Outcome
		Ensure straight passageways can be traversed	Valid, accurate search
		Ensure junctions can be traversed	Valid, accurate search
		Ensure loops are not re-traversed unnecessarily	Loops is re-traversed once – suboptimal, but still valid.
		Ensure the robot can handle curved paths	Valid, accurate search

Table 8 - Test paths and results.

The state machine describing the algorithm can be found in Appendix F.

Maze Solving Algorithm

Once a full tree is produced by the maze traversal algorithm, a modified version of Dijkstra's algorithm is used to find the shortest path between any two points in the maze.
The alternatives considered were:

- **A*:** Similar to Dijkstra's algorithm but uses heuristics to find the shortest path. This was rejected as it may not give the true shortest path.
- **Exhaustive search:** Brute force is used to check every path from start to finish. This method was rejected as it was extremely slow.

Dijkstra's algorithm was accepted as it is adequately fast, guarantees the shortest path and uses the tree established through maze traversal. The shortest path between any two nodes can be calculated by using the Pythagorean theorem, and then Dijkstra's algorithm can be used as necessary. Distances are calculated based on positions, rather than stored distance data as the latter is unreliable. For example, if distance data were stored, it is possible to have a closed loop that does not end where it began.

This algorithm returns the shortest-predecessor graph, which contains an edge between each node and the predecessor which allows it to reach the starting point with the shortest distance. An example of such a graph is below:

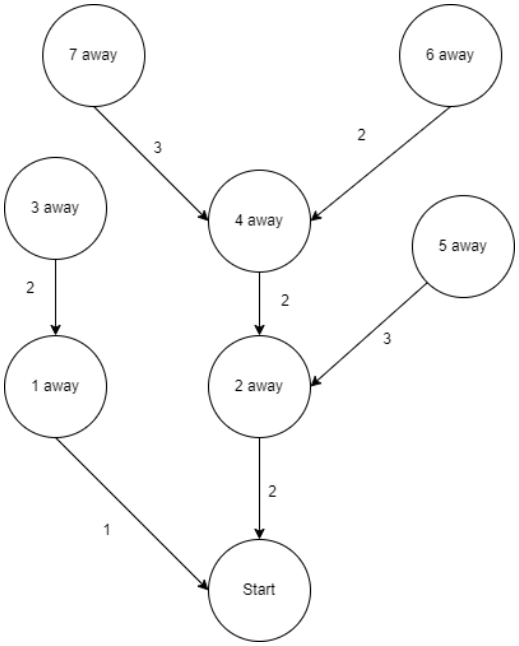


Figure 34 - Sample predecessor graph.

Client-Server model

The group opted for a client-server star topology as it allows for a scalable solution (meeting Requirement 2.3.d). New robots can easily be added to the system and multiple clients can remotely gain access by sending a request to the server.

Server

The server consists of three main instances: the frontend, using React, running on port 80; the database, using MariaDB, running on port 3306; the web server, using Flask, running on port 5000. Communication between the client, server, robot, and database all use TCP, as all information sent must be guaranteed to reach its destination, otherwise the mapping and display algorithms will not work.

The web server uses a Flask framework to build an API, which manages information delivery between the client, robot, and database.

The web server uses the maze traversal state machine to deliver the next instructions to the robot. One endpoint is used to communicate with all robots, allowing an arbitrarily large number of robots to be connected with reduced overhead.

The alternatives considered were:

- **Node.js:** Difficult to integrate maze traversal and solving (since it was written in Python).
- **Django/Tornado:** Lack of experience.
- **Web sockets:** Faster, but generally more difficult to maintain (Requirement 2.3.c).

Furthermore, there were several API endpoints made to serve the frontend. These endpoints query the database to provide the information required for frontend services. These services include displaying all robots (connected and disconnected), displaying sessions, pausing, playing and emergency stopping connected robots, changing robot nicknames, and finding the shortest path. There are also three endpoints for controlling the LED beacons, as they only need to be activated while the robot is spinning to triangulate its position. Error handling has been accounted for through ubiquitous use of exception handling statements on database queries and returning HTTP error code 400 on invalid API requests. Documentation for the API can be found in Appendix G.

The database uses MariaDB, a relational database management system (RDBMS). It stores information about robots, sessions, replay and diagnostic information, and the trees any robot has fully completed. This is needed so that any client can revisit previous sessions and find the shortest path through that maze. Replays, diagnostics, and maze trees are stored with their corresponding session IDs. This allows the frontend to use this data to show a visualisation of the maze, both in real time and as a replay.

The alternatives considered for the RDBMS were:

- **MySQL:** MariaDB is a more optimised fork of MySQL.
- **PostgreSQL:** More appropriate for larger databases.
- **NoSQL (MongoDB):** Doesn't guarantee immediate consistency, which could cause irregular mapping.

An entity-relationship diagram can be found in Appendix H.

Frontend

A user interface (UI) was needed to allow the user to view the final map, the position of the robot and the shortest path.

React was chosen for the frontend as it is simple to use given a basic understanding of JavaScript. It also allows components to be reused, which is more efficient as replays are similar to live mapping (Requirement 2.2).

Design

To ensure a familiar user experience, take advantage of the scalability of the server backend and to meet Requirements 2.3, the following was decided:

- All robots should be displayed, and currently connected robots should be selectable for live mapping.
- Live mapping is autonomous but can be pause and resumed.
- Session replays can be accessed and use standard playback buttons as shown in Appendix I.
- Any two points can be selected and the shortest path between those points will be found and plotted (Requirement 1.2).

For the overall design of the system, usability was the main factor considered. The solution should be simple and intuitive, meaning new users should be able to easily understand navigation of the webpage. The solution should also be robust and reliable, so minimising user input was a key goal to establish ease-of-use and reducing vulnerabilities.

Implementation

When the design was implemented, changes were made from the initial design to allow integration with the server. The navigation between pages for the final implementation is shown in Appendix J.

Home Page

The hub of navigation is the home page. The user has the option of selecting robots or replays. A sliding carousel was implemented, giving a smooth, interactive user experience. This is expanded by overlays when the mouse hovers over a selectable robot or replay as shown in Figure 35, Figure 36 and Figure 37.

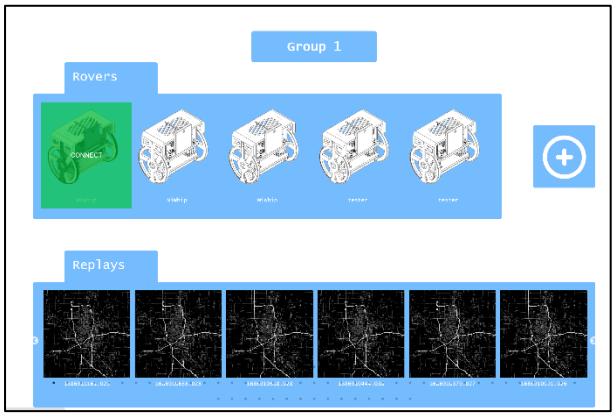


Figure 35 - Mouse hovering over connected robot.

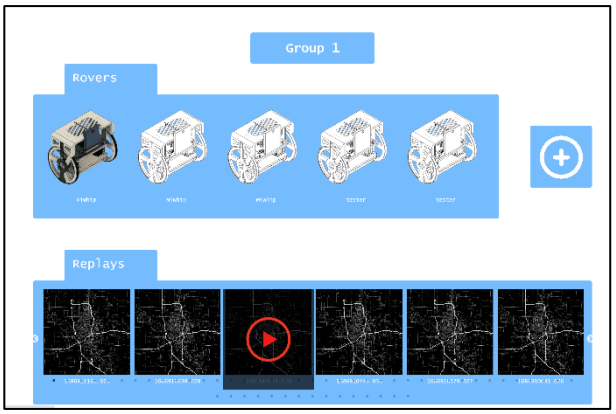


Figure 36 - Mouse hovering over replay.

Connected Page

Previously the "start", "pause", "play" and "view replay" buttons had their own individual pages but were later merged into one "Connected" page as they are very similar, reducing redundant code.

The state machine shown in Appendix J describes how the front end keeps track of what buttons should be displayed and is separate from the state machine in the server used to control the robot. The page assumes the robot starts in the "Start" state. When the user clicks the "Start Mapping" button, a post request is sent to the server. The message is relayed to the robot and mapping begins. The same happens when the client wants to pause the robot. The implementation is shown in Figure 39, Figure 38, Figure 40 and Figure 41.

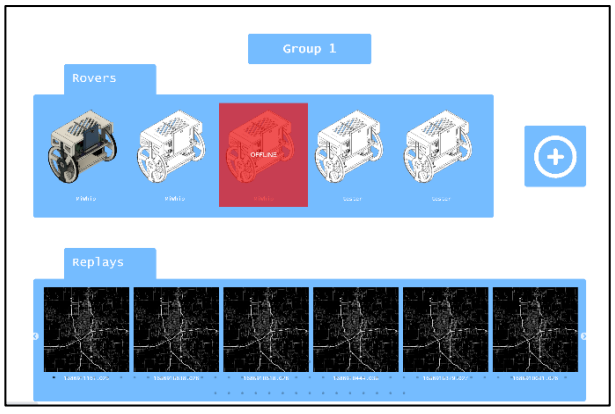


Figure 37 - Mouse hovering over offline robot.

Figure 39 - Connected page (start state).

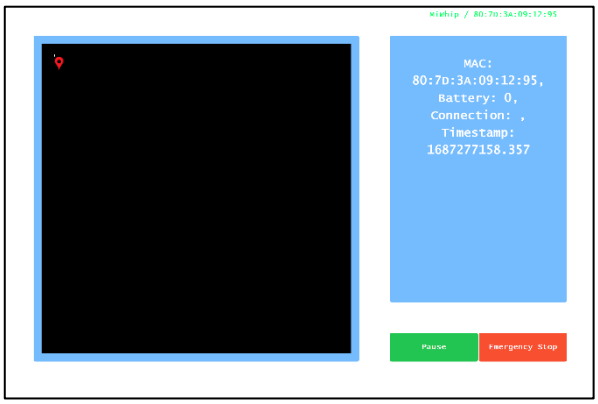


Figure 38 - Connected page (mapping state)

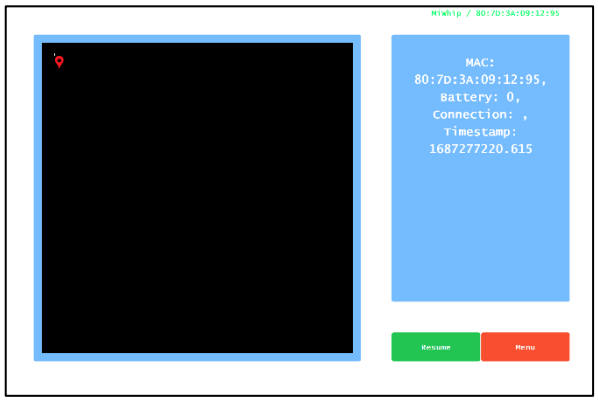


Figure 41 - Connected page (pause state)

Figure 40 - Connected page (finished state).

When the user selects a robot from the home page, diagnostic and replay data polling starts under a unique session ID. The most recent diagnostic data is displayed, and the map is updated. In the failure mode when polling is unsuccessful, the webpage re-requests data from the server a fixed number of times before timing out and the page must be reloaded.

When mapping finishes, the state machine transitions into the 'Finish' state as shown in Figure 40. Clicking the replay button transitions the webpage to the replay page corresponding to the session.

The mapping state, shown in Figure 38, contains an emergency stop button which terminates mapping and puts the robot in an idle state. This is a fail-safe that makes the solution more robust.

Replay Page

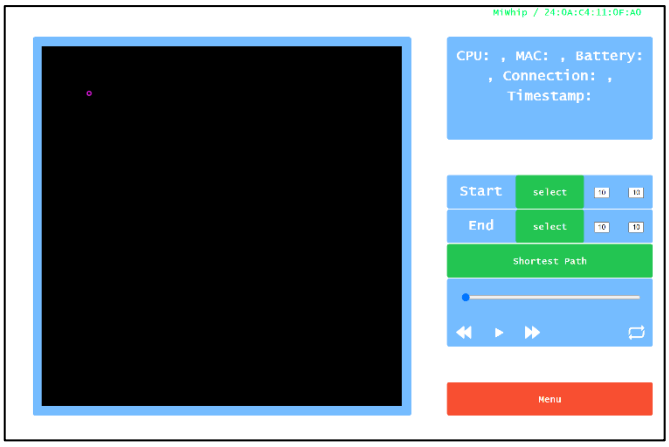


Figure 42 - Replay Webpage

When a replay is selected, the page requests the mapping and diagnostic data for the whole session from the server. The current position in the replay is set via the slider. All playback buttons function as desired and the play button switches to pause or replay when required. The user can also drag the slider, providing a more interactive experience.

Map Visualisation Component

The ToF sensors find the distance to the LED strips on either side of the robot. Readings are taken everywhere in the maze and since the robot always knows its location and orientation, the walls can be drawn to accurately recreate the maze. As the data is fed to the webpage, a small line is drawn for each data point. When all the lines are compiled together, a map complete with curves and junctions will be displayed. An example of this is shown in Figure 43. The ToF sensors are not reliable at distances greater than 500mm so anomalous data can be rejected.

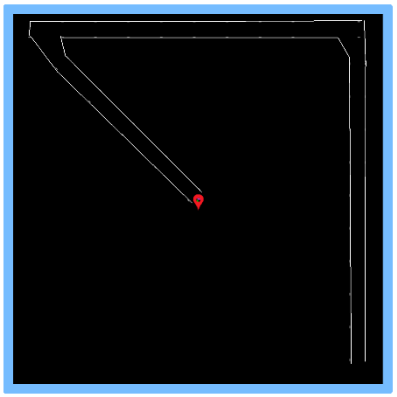


Figure 43 - Complete mapping visualisation (test course).

Replay data is the same as live data except the endpoint updates in real-time as data arrives. The connected page must poll the endpoint for new data. The map is then drawn.

A weakness of this design is its dependency on the two side ToF sensors. In the failure mode that data is not received, or anomalous data is received from the sensor, no line is drawn.

Diagnostic Data Component

Similar to mapping data, diagnostic data is both used in live and historic replays. Diagnostics (as shown in Appendix I) are requested by the webpage from the server. This gives all diagnostic data for the session which can be displayed in both the connected and the replay pages.

Shortest Path Component

The user selects a start and end point on the map. The webpage then requests the shortest-predecessor graph (generated by Dijkstra's algorithm). The shortest path is then drawn using the predecessor graph. Client-side validation is used to meet Requirement 2.2 by snapping to the nearest valid node. As there are many nodes placed along the path, the displayed shortest path will appear as a smooth curved line. Using more nodes than required means the path can follow bends and remains in the centre of the passage. This is shown in Figure 44.

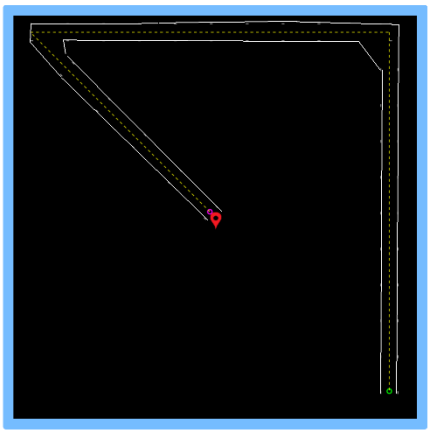


Figure 44 - Complete shortest path visualisation (test course).

Physical Design

It was deemed necessary to use a custom chassis to properly distribute weight, integrate sensors and for other reasons described in the Mechanical Design section. A custom PCB was also designed.

PCB

Overview

A custom printed circuit board (PCB) was designed to integrate all the electrical components of the project. This also helped to reduce both size and weight while improving reliability by decreasing the number of easily unpluggable connectors.

Prototype

A prototype of the circuit board was first created using stripboard to test the connections and compatibility of the components. It also allowed testing and tuning to continue while the PCB was being manufactured.

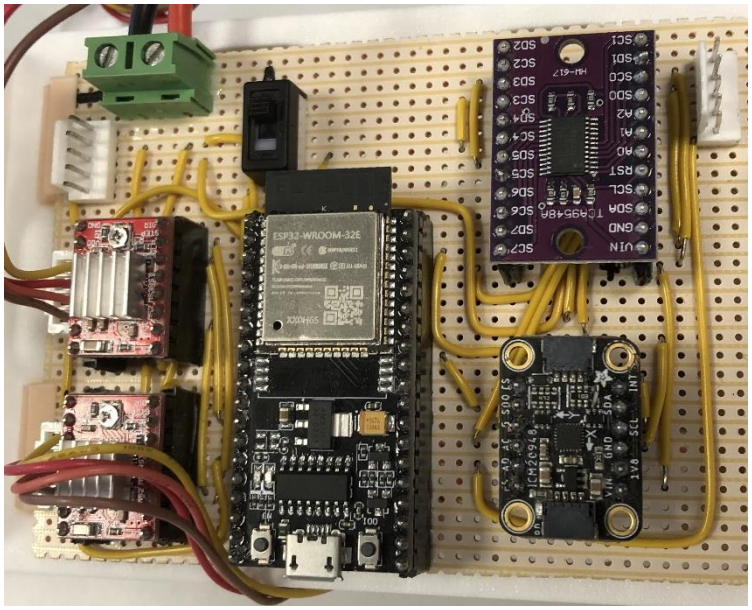


Figure 45 - Prototype circuit board.

Several lessons were learned from this prototype:

- Six of the ESP32 pins are used by the module's onboard flash and can't be used as GPIO.
- 4 pins are input only, and only 2 of them support interrupts.
- Several other pins do not support interrupts.
- The TX0 and RX0 pins are used for the USB communication and therefore cannot be used.
- SPI frequencies greater than 1MHz can cause interference in the I2C lines next to it if improperly handled.

Schematic

With the lessons learned from the stripboard prototype, designing the PCB could then begin. The software used was KiCad.

The first step was creating a schematic which can be seen in Appendix K. This required careful selection of components considering budget, size, functionality, power constraints and many other criteria.

A new part that had to be added compared to the stripboard prototype was the power supply since the PCB is required to power both the ESP32 microcontroller unit (MCU) and the FPGA, as well as other components. This required two voltage rails, 5V for the FPGA, and 3.3V for the MCU, sensors and other components. The maximum observed currents drawn by all the components of the robot were measured and the results are shown in Table 9.

Component	Max Current (mA)	Voltage (V)	Max Power (mW)
ESP32	240	3.3	792
FPGA	220	5.0	1100
Motor drivers	8	3.3	26
Motors	~2000	7.2	14400
Sensors + other ICs	<10	3.3	33

Table 9 - Components and corresponding maximum currents, voltages, and powers.

The results show that to allow a large safety margin, both the PCB's 3.3V rail and 5.0V rail must be able to support up to 1A of continuous current, and the motor connections must be able to support up to 2.5A each.

A dual synchronous buck SMPS capable of taking either the battery or USB voltage and supplying the needed voltages was decided upon. This is because of their high efficiency and low drop out voltage. The part chosen was the Texas Instruments TPS54295RSAT [17] for the following reasons:

- Low cost.
- Efficient integrated field effect transistors (FETs).
- High current output (2A per rail).
- Protection features:
    - Over-current limiting
    - Over and under voltage protection
    - Thermal protection.
- High frequency (700kHz) allowing for smaller components.

As this part could not be tested before implementing it, lots of care was taken when consulting the datasheet to make sure the circuit met the manufacture's requirements for optimal performance.

Additional features include:

- Power and status LEDs.
- Reset and boot buttons.
- Jumpers to select different stepping and microstepping modes.
- Additional I2C ports.
- Additional power ports.

These allowed for easier debugging and for the design to remain flexible even if the requirements were changed or new sensors needed to be added.

Most components are surface mount devices (SMD) due to their small size and abundance compared to older through hole (THT) parts. Passive components are mostly in the size 0402 due to availability.

Stackup

It was decided to use a four-layer PCB because they are comparable in price to two-layer boards but are much simpler to route. This is because the inner layers can be dedicated plane layers, reducing the amount of routing needed. The two inner layers were both chosen to be ground planes to provide optimal electromagnetic interference (EMI) performance [18]. Signals are then routed on the outer layers and power is poured in the remaining spaces.

layer	Material Type	Thickness
Layer	Copper	0.035mm
Prepreg	7628*1	0.2104mm
inner Layer	Copper	0.0152mm
Core	Core	1.065mm
inner Layer	Copper	0.0152mm
Prepreg	7628*1	0.2104mm
Layer	Copper	0.035mm

Figure 46 - JLCPCB's JLC04161H-7268 stackup. [19]

The stackup has controlled impedance which is required to meet the 90Ω differential impedance required for USB.

Layout and Routing

The first step was creating the outline of the board. This involved measuring the FPGA with callipers to ensure a good fit between the FPGA and the PCB. This outline was then tested by 3D printing it and placing it on top of the FPGA as shown in Figure 47. Once it was confirmed that the part fit and the holes lined up, component placement could begin.

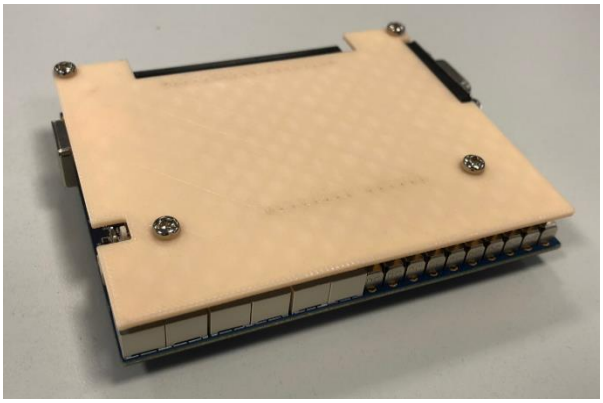


Figure 47 - 3D printed test outline.

The highest priority parts were placed first. These are ESP32 module which requires clearance around the antenna (50x20mm) and the IMU which needs to be located as centrally as possible. Next, the connectors were placed where they would need to be accessed, followed by the larger components such as the motor drivers and fuse. Finally, the smaller integrated circuits and their accompanying passive components were placed in the remaining space.

The buck SMPS was laid out to minimise loop area, which reduces inductance and resistance, and therefore improves the efficiency and responsiveness of the regulator [19].

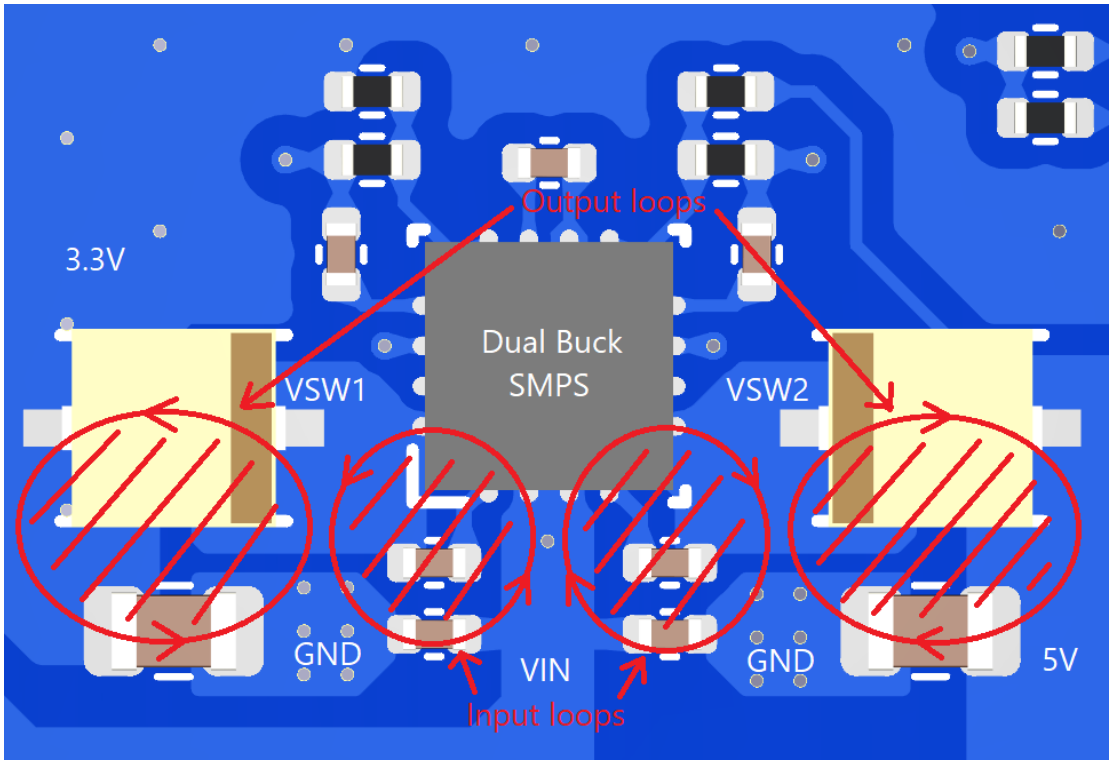


Figure 48 - Buck converter layout

The 90Ω USB differential pair's trace-width and spacing was calculated based on the stackup and then implemented as shown in Figure 49.

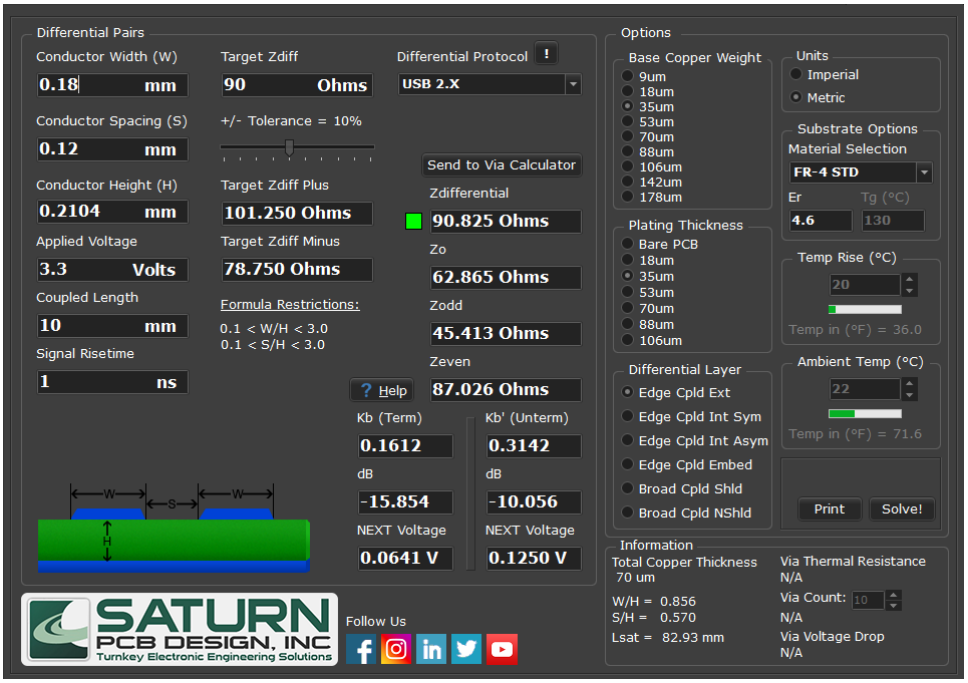


Figure 49 - Saturn PCB differential pair calculator.

Testing and Conclusion

The PCB was ordered and assembled. A few minor issues were present, but these were fixed with jumper wires. The schematic (Appendix K) was updated to reflect the changes.

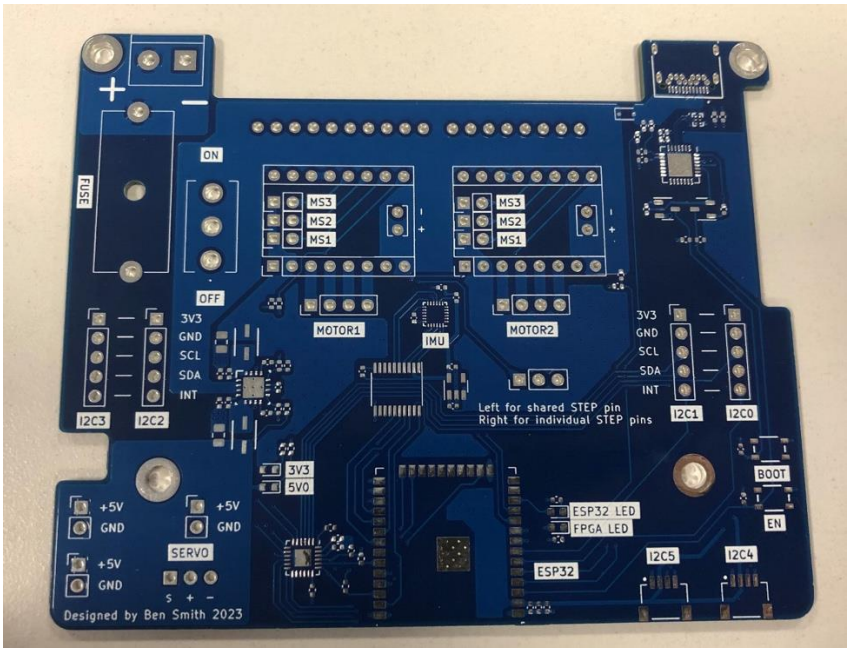


Figure 50 - The actual PCB.

Mechanical Design

Overview

The objective for the chassis was to fulfil several requirements. The chassis must be:

- **Modular**: Due to the continuous integration approach, e.g. if the angle of the ToF sensors was to be changed, the entire design shouldn't have to be reworked.
- **Robust:** Testing often involves the robot falling over and hitting the ground. The robot would need to be constructed in a way that protects any vulnerable components inside (Requirement 2.2).
- **Consistent:** All components in the robot should be rigidly mounted to prevent movement between tests as this could affect the results (Requirement 2.1).

Initial Chassis

The original chassis was not used for several reasons, firstly its low modularity. To attach or adjust new components, holes must be drilled, or components would have to be mounted in a non-rigid way. Secondly, the chassis was not supplied until the 2nd week, by which time the second iteration was already fabricated.

Iterative Design

First Iteration

The first iteration was heavily inspired by the B-Robot by jjRobots [20]. It was created to test 3D printing design tolerances. This version was used to decide sizes for screw holes, tolerances for parts which slot together, and testing the design of the wheels. The wheels were 3D printed and wrapped in GT2 timing belt, improving friction, which is typically low for parts printed in PLA.

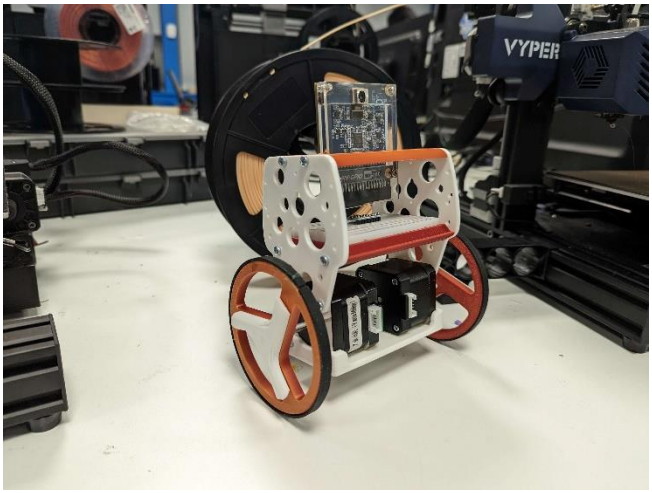


Figure 51 - First robot iteration

Second Iteration: Electronics Testbed

The second iteration was designed to allow testing of the electronics that would be used on the final design (excluding the FPGA and camera). This design continued using the belt-covered wheels of the first iteration, whereas everything else was redesigned. This version included easy access to four shelf-mounted breadboards for simultaneous testing and debugging. Two magnetically attached arms allowed for ToF sensors to be mounted when needed. The side panels had openings to route power cables to the PSU and wires to the ToF sensors.

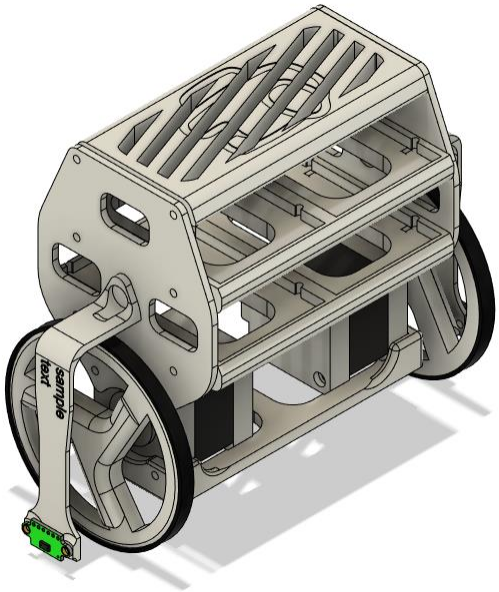


Figure 52 - Second robot iteration

Third Iteration: Integration Testbed

Since the second iteration balanced successfully, it was decided to start integrating everything on a single robot. This chassis was widened to accommodate the battery, and the removeable breadboard trays were scrapped in favour of a sliding stripboard mount that allowed for easy removal. The ToF mounts were screwed instead of attached magnetically. While this iteration allowed integration of everything onto a single robot, it had two main flaws:

- The battery was difficult to remove for charging.
- The wheels eventually started spinning around the motor shaft due to wear and tear.

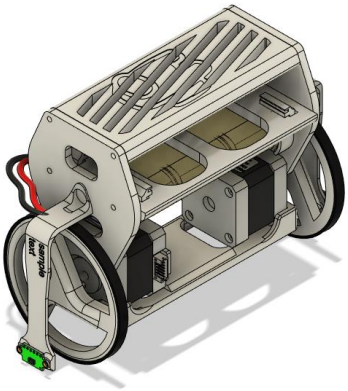


Figure 53 - Third robot iteration

Fourth Iteration: Fully Integrated Robot

The next iteration was intended to be the final version. It was understood that enlarging the wheels and moving the battery below the motors would make the battery easy to access while lowering the centre of mass and making the robot easier to balance. This was achieved using a compliant snap-fit mechanism (shown in Figure 54). The motors were recessed into the mounting plate to further lower the centre of mass.

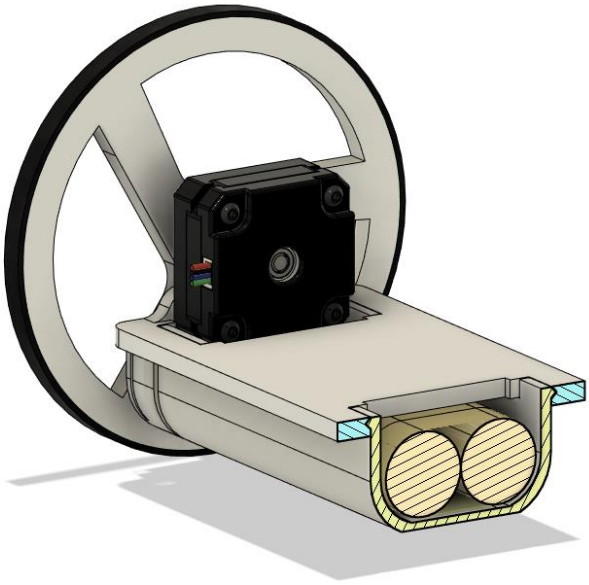


Figure 54 - Snap-fit mechanism and recessed motors.

This version allowed for the FPGA and PCB to be mounted in the main shelf which could be easily accessed by removing the magnetically attached lid. This was useful for tuning the motor driver potentiometers and flashing configurations onto the FPGA. As the ToF mounts of the previous versions were not rigid enough, this iteration used 3mm acrylic over the previously used PLA.

While the control performance of this robot was excellent, shortly after its completion Requirement 1.3 was expanded to bar passively balancing robots, requiring this iteration to be abandoned.

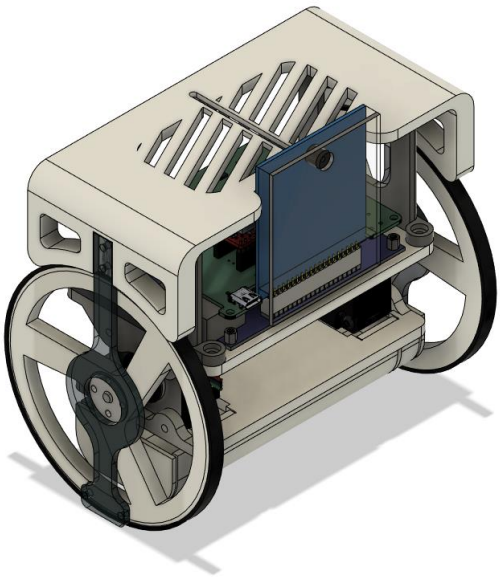


Figure 55 - Fourth robot iteration

Fifth Iteration: Final Version

As the robot could not be passively balanced, this iteration raised the battery as high as possible. Several techniques developed from the previous iteration were used, such as the recessed motor placement to make the robot 12mm shorter. The ToF mounts were updated to be thicker and more rigid to resist tension from the connectors. The easy access to the electronics bay and battery were maintained by having the lid double as the battery mount. This required only the removal of a single part for access to the battery and electronics. A cutout was made for a USB-B port to allow access to the FPGA for re-flashing of firmware.

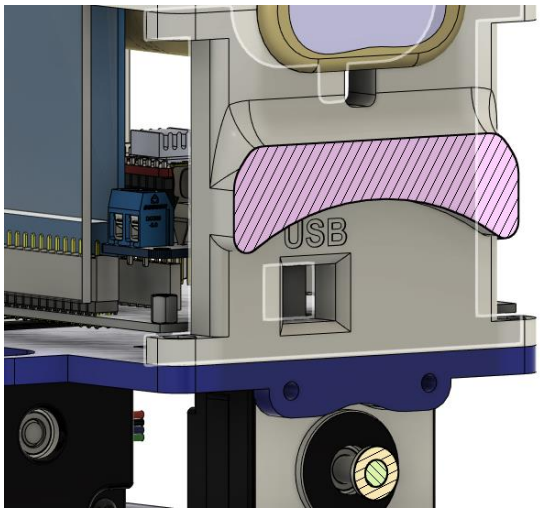


Figure 56 - USB-B cutout for FPGA access

To increase robustness, a bumper was added to the front and back to ensure that when the robot fell during testing, the ESP32 antenna on the back and the USB-C port on the front would not be damaged. The FPGA tray was designed to provide the camera with an exact height of 140mm above ground level which was determined as the optimal height to detect the beacons. The camera angle can also be adjusted ±12 degrees by adjusting the size of the mounting spacers.

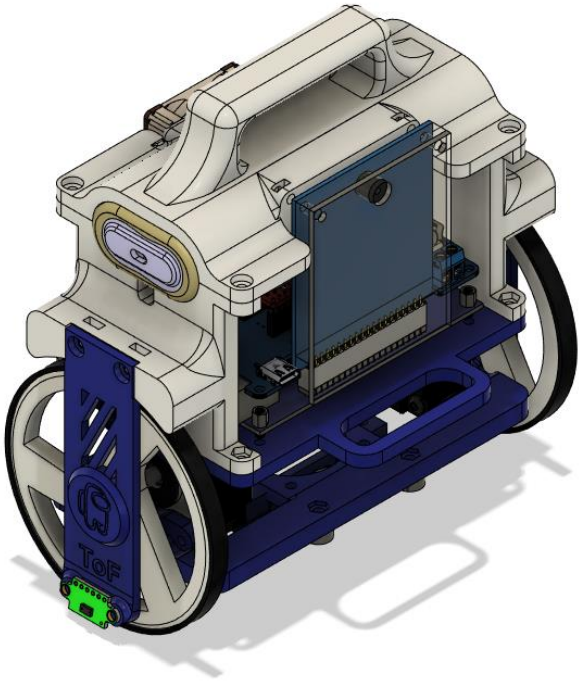


Figure 57 - Final robot iteration

Thermal Considerations

There are several sources of heat that needed to be considered. The motors, while limited to 1A by the A4988 motor drivers, deliver maximum torque at 2A. To use the motors to their maximum capacity, TMC2208 motor drivers were installed. The motors themselves are a source of heat and, while the datasheet specifies that they can operate at 80°C above ambient temperature, the motor mounts will become deformed at such high temperatures. Finally, it was observed the microcontroller would become very hot when running the control code simultaneously with server communication via Wi-Fi. The microcontroller and electronics compartment were cooled using a single fan to deliver airflow which increases heat dissipation [21]. A heatsink was applied to each motor and a fan was fixed to the motor mount to protect against the deformation of the plastic.

Material Choices

The main considerations for material choices were availability and thermal properties. PLA was the first material choice as it was readily available for 3D printing in the lab. However, PLA begins to warp at 50 degrees [22] which was observed in the PLA motor mounts (Figure 58). For this reason, PETG was used for the heat-sensitive parts such as the motor mounts, which has similar availability and cost to PLA.

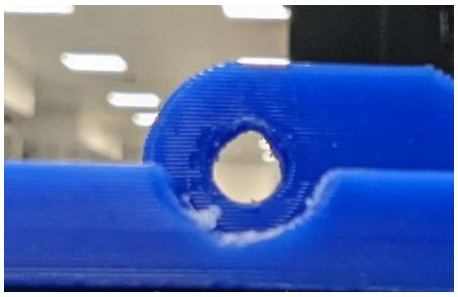


Figure 58 - Plastic deformation of motor mounts

PETG starts deforming at 85 degrees [22] which the motors should never reach with active cooling.

Whole System Testing

Each subsystem has been tested extensively in its relevant section. This section will focus on testing the entire system's performance and the interaction of subsystems.

The first time the Wi-Fi and the control systems were enabled simultaneously, unusual behaviour was noticed (Figure 59).

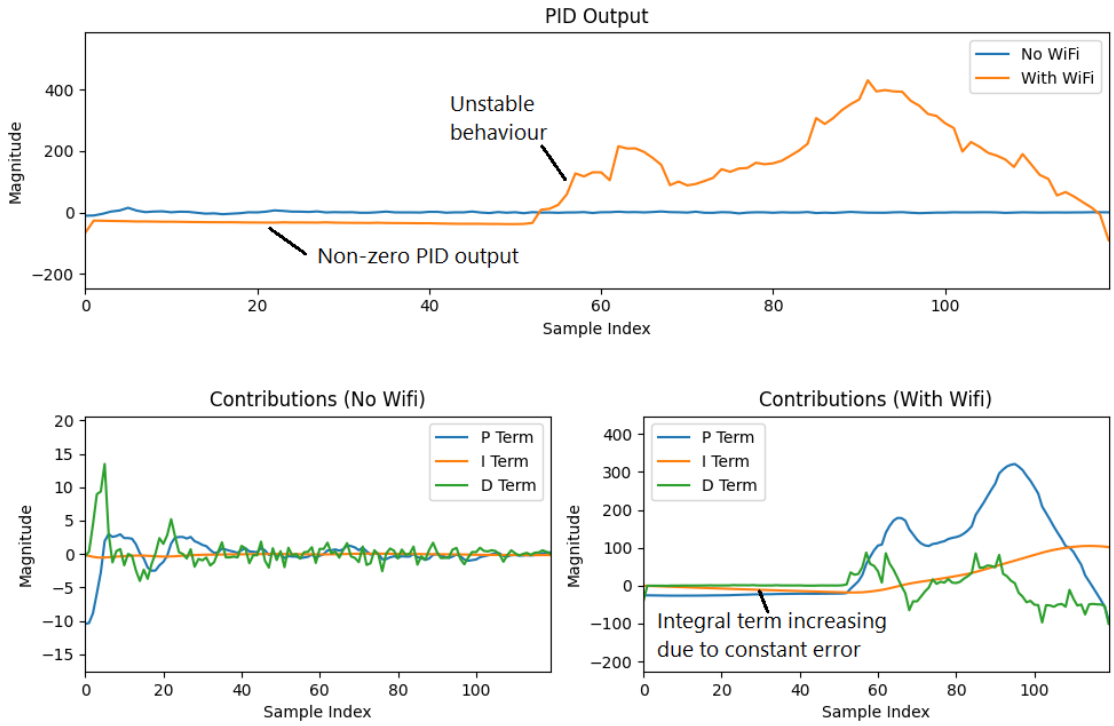


Figure 59 - Controller response with Wi-Fi communication

After the Wi-Fi was configured, the controller would immediately become unstable. After much testing, it was discovered that this was due to the IMU being initialised and outputting one reading before the Wi-Fi began. This led to a large integral error making the system unstable. To fix this, the boot order was changed.

The following tests are used to test robot-server and client-server communication.

The robot was tested to follow a passage, dictated by the server. The figures below show the path it took. This process was repeated for many passages, with curved or angled paths, with results showing adequate performance.

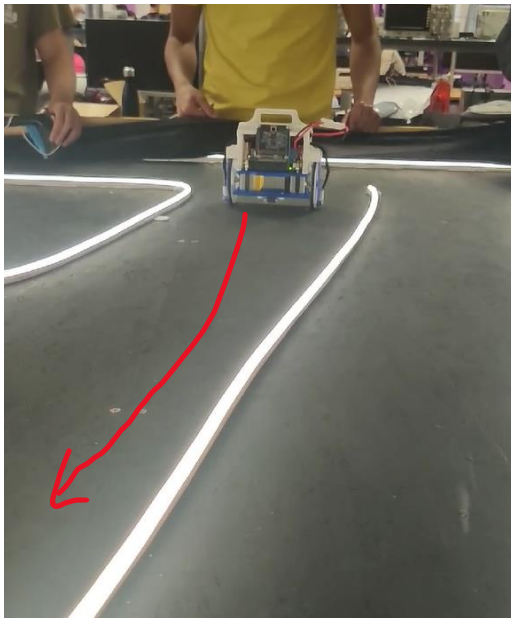


Figure 61 - Direction of robot movement (before).

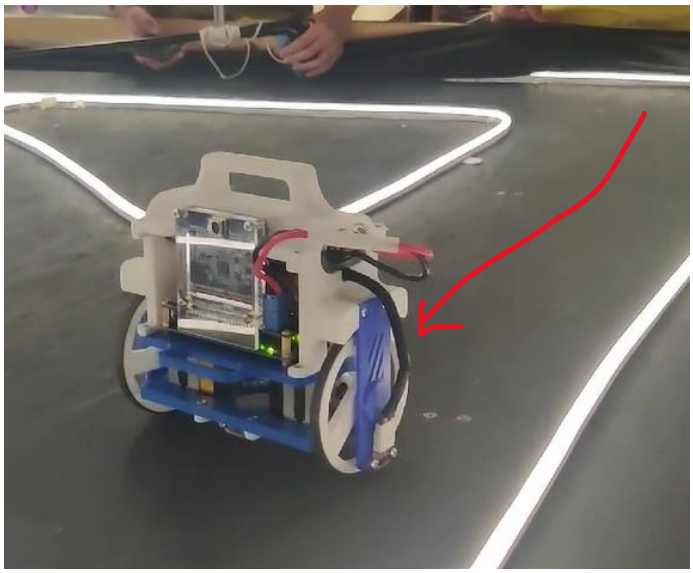
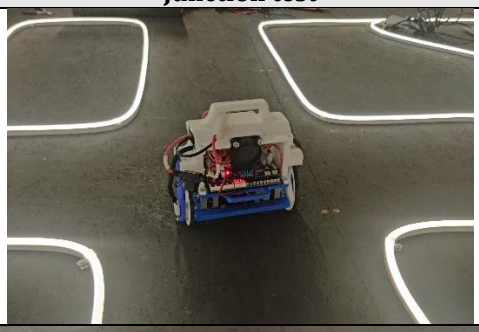
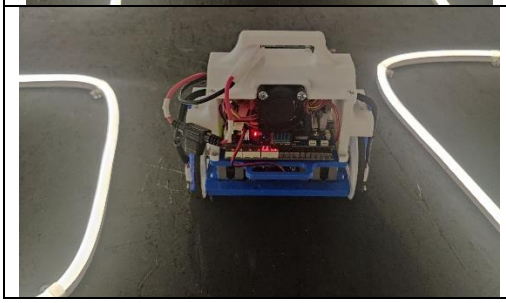


Figure 60 - Direction of robot movement (after).

To test junction detection, the robot was placed at a junction and instructed to spin. It then responded to the server with angles to the junctions. Absolute angles are not important, only the difference between angles.

Junction test	Angles returned
	[-136.55, -57.25, -8.11, 90.25] VALID
	[72.01, -112.02] VALID

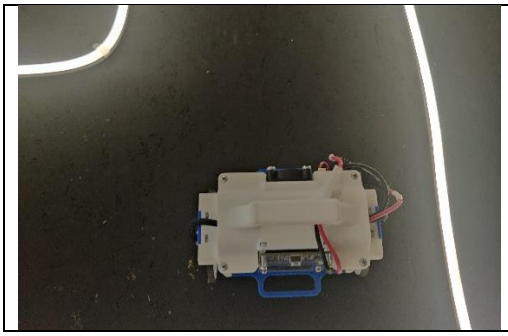
	[-93.29, -45.0] INVALID [-92.88, 17.22] VALID

Table 10 - Testing results of junction test.

The robot does not spin on the spot due to interactions between the linear speed controller and angular rate controller. This affects the position of the robot, so dead reckoning must also update the position during spins.

A replay of the robot passage movement was stored and shown in Figure 62. The direction was incorrect, due to a sign error in visualisation, which was quickly fixed.

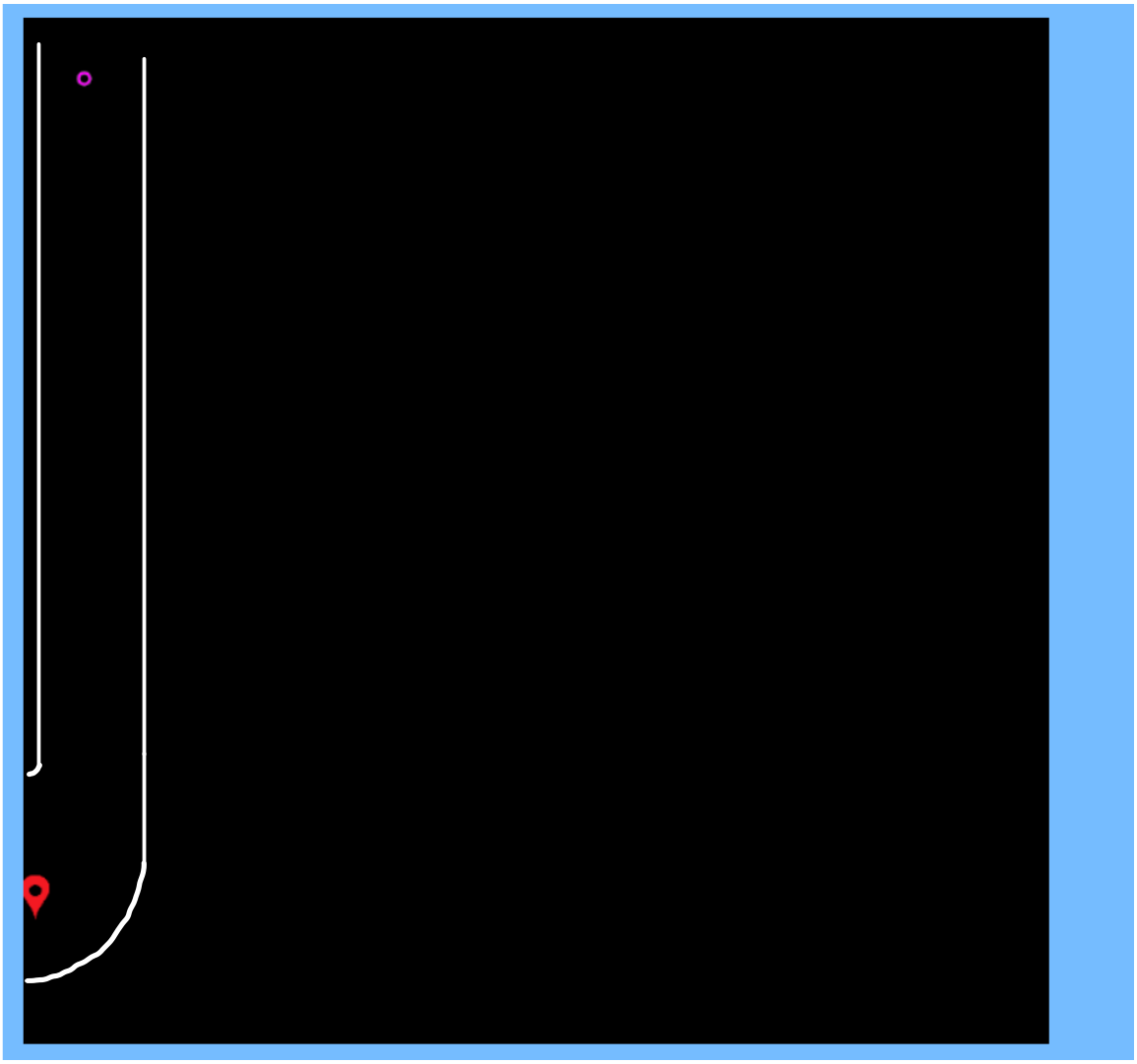


Figure 62 - Visualisation of robot movement.

Final Package

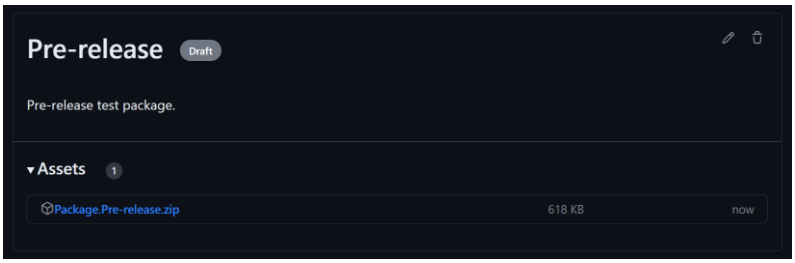


Figure 63 - Pre-release package

A final build package was created containing all the files necessary to build and run the robot. This was done so a customer could easily deploy the whole system in any environment. The exact contents of the package are listed below:

- Microcontroller code binaries
- FPGA programmer object file
- CAD files
- PCB Files
    - Gerber files needed for manufacturing the PCB.
    - Interactive bill of materials (BOM) for easy assembly (Figure 64).
- Guide on how to access the webpage.
- BOM

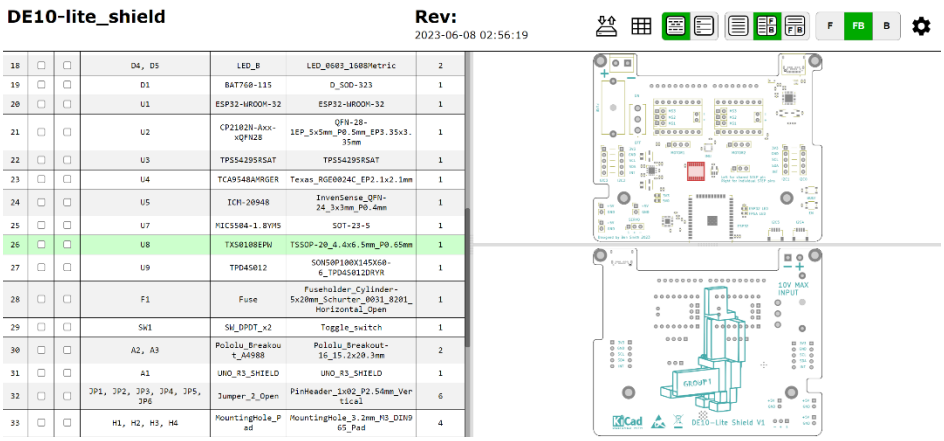


Figure 64 - Interactive bill of materials.

Conclusions

The goals set out in the Requirements section were evaluated:

1. Functional requirements:
   1.1 This was achieved using ToF sensors on the sides and front of robot, to prevent collision and provide information. Velocity, path and heading control were also implemented to meet this requirement.
   1.2 This was achieved through Trémaux's and Dijkstra's algorithms and frontend maze visualisation software using React.
   1.3 This was achieved through a pitch controller based on the physical properties of the robot and using the knowledge learned in the Control Systems module. This controller was tuned through numerical methods and trial and error.

2. Non-functional requirements:
   2.1 The server maze mapping algorithm interacts with the robot's systems without oversight and runs constantly until the maze is completely mapped.
   2.2 The main failure modes have been accounted for, both at a subsystem level and through full system integration and testing. Efficient physical construction and robust error handling software has been built to further meet this requirement.
   2.3 Coded for
      a. When designing the user interface, usability was the foremost consideration. This was done using a simple, intuitive interface reminiscent of other applications a user would be accustomed to.
      b. A modular design at every level was used to allow every system and subsystem to be tested both individually and together.
      c. All the code has been commented extensively and all subsections are well documented on the GitHub.
      d. The use of cloud infrastructure as well as the client-server model allows for an arbitrarily large number of users. The integration of all modules into a final package for easy deployment also helps meet this requirement.

3. Implementation restrictions:
   3.1 The following permitted means of locating the robot within the maze were used:
      a. Dead reckoning was used between triangulations with the beacons.
      b. Periodically the beacons were used to triangulate the position of the robot and update the current location.
   3.2 This was achieved through the DC grid involving an MPPT SMPS, a supercapacitor, and beacon drivers which were Wi-Fi controlled to achieve optimal energy distribution.
   3.3 The FPGA camera system was used to locate the beacons.

Future Work

Potential improvements to the robot include:

- **FPGA camera lens:** The field of view of the camera could be improved by using a lens. This was briefly tested, but the image processing requirements to unwrap the distorted image were too high to implement in time.
- **Machine learning:** A convolutional neural network could be implemented to better detect beacons and other maze markings. This would require better hardware resources and knowledge that the team does not possess.
- **Simultaneous localisation and mapping (SLAM):** Using more sensors and more sophisticated processing techniques, it is possible to build a highly accurate map of the robot's surroundings. This would reduce errors in dead reckoning.
- **LQR control:** Using this control paradigm in the robot could lead to increased efficiency and better battery life, since actuation cost could be minimised.
- **Continue mapping:** On unexpected disconnection from server, the robot could reconnect and continue mapping from where it left off. This could be achieved by saving the robot's state and incomplete maze tree on disconnection and later retrieving it on reconnection.
- **Bidirectional SMPS:** The bidirectional SMPS could be implemented to achieve better control of power flow in the DC grid by storing power in the capacitor.
- **MPC:** This would allow for the best control performance. To achieve this, a more powerful CPU would be needed to run MPC in real time. [23]
- **Live video:** Video is sent from the robot to the server via Wi-Fi. This would allow for server-side computer vision and a live camera feed could be displayed to the end user.
- **Over the air updates:** The robot could request the latest version of the firmware from the server on startup and update itself if its version is behind.

Bill of Materials

PCB

Name	Quantity	Notes	Unit Price
PCB	1		£2.00
CP2102N-A02-GQFN28	1	USB to UART bridge	£1.68
ESP32 WROOM 32E	1	MCU	£3.50
TPS54295RSAT	1	Dual buck SMPS	£2.14
TCA9548AMRGER	1	I2C Multiplexer	£1.96
ICM-20948	1	IMU	£7.50
BAT760,115	5	Diode	£0.27
MIC5504-1.8YM5-TR	1	1V8 LDO Regulator	£0.16
TXS0108EPW	1	Level shifter	£1.24
TPD4S012	1	USB ESD Protection	£0.45
31.8201	1	Fuse	£0.42
Pin Headers	8		£0.00
Pin Sockets	5		£0.00
Molex Connectors	4	I2C Connector	£0.07
JST SH	2	I2C Connector	£0.10
Buttons	2		£0.08
Toggle Switch	1		£0.00
Screw Terminal	1		£0.20
USB-C	1		£0.42
1.5kΩ	3		£0.00
2.2kΩ	4		£0.00
3.3kΩ	1		£0.00
4.7kΩ	4		£0.00
10kΩ	15		£0.00
22.1kΩ	2	RC0402FR-0722K1L	£0.00
33kΩ	1		£0.00
73.2kΩ	1	RC0402FR-0773K2L	£0.01
124kΩ	1	ERJ2RKF1243X	£0.03
10nF	4		£0.00
100nF	8		£0.00
1uF	1		£0.00
4.7uF	1		£0.00
10uF	6		£0.00
47uF	2		£0.00
100uF	2		£0.00
2.2uH	1	IHLP1212BZER2R2M11	£0.18
3.3uH	1	IHLP1212BZER3R3M11	£0.18
LED Red	2		£0.00
LED Blue	2		£0.00
Total			**£24.07**

Chassis

Name	Quantity	Notes	Unit Price
PLA Filament	0.12	1kg price	£10.00
PETG Filament	0.05	1kg price	£13.00
tmc2208	2		£8.56
GT2 Belt	0.57	price/m	£0.95
Wheel Coupler	2		£1.10
ToF	3		£1.17
4010 fan	2		£2.50
Small Heatsink	3		£0.15
Motor Heatsink	2		£1.29
M3x6 Screws	8		£0.00
M3x8 Screws	16		£0.00
M3x12 Screws	4		£0.00
M3 Nuts	28		£0.00
Total			**£33.29**

The total bill of materials cost is £57.36, under the £60 budget.

References

[1] H. Merdan, *ELEC50008 - Engineering Design Project 2,* London, 2023.

[2] R. James, "I2C Device Discussion," 26 May 2014. [Online]. Available: https://www.i2cdevlib.com/forums/topic/153-official-dmp-documentation-is-released-by-invensense/. [Accessed 19 June 2023].

[3] xioTechnologies, "Fusion," GitHub, 30 March 2022. [Online]. Available: https://github.com/xioTechnologies/Fusion. [Accessed 19 19June 2023].

[4] SparkFun, "SparkFun_ICM-20948_ArduinoLibrary," GitHub, 28 February 2021. [Online]. Available: https://github.com/sparkfun/SparkFun_ICM-20948_ArduinoLibrary. [Accessed 19 June 2023].

[5] ST Microelectronics, *VL53L0X Datasheet,* Geneva, 2022.

[6] Adafruit, "Adafruit_VL53L0X," GitHub, 23 September 2016. [Online]. Available: https://github.com/adafruit/Adafruit_VL53L0X. [Accessed 19 June 2023].

[7] S. Hymel, "Introduction to RTOS - Solution to Part 11 (Priority Inversion)," DigiKey, 12 February 2021. [Online]. Available: https://www.digikey.co.uk/en/maker/projects/introduction-to-rtos-solution-to-part-11-priority-inversion/abf4b8f7cd4a4c70bece35678d178321. [Accessed 19 June 2023].

[8] F. Jimenez, I. Ruge and A. Jimenez, "Modeling and Control of a Two Wheeled AutoBalancing Robot: a didactic platform for control," in *Engineering, Integration, and Alliances for a Sustainable Development. Hemispheric Cooperation for Competitiveness and Prosperity on a Knowledge-Based Economy: Proceedings of the 18th LACCEI International Multi-Conference for Engineering, Education and Te*, Virtual, 2020.

[9] S. A. Bin Junoh, "TWO-WHEELED BALANCING ROBOT CONTROLLER DESIGNED USING PID," Universiti Tun Hussein Onn Malaysia, Johor, 2015.

[10] J. Bennet, A. Bhasin, J. Grant and W. C. Lim, "PID Tuning via Classical Methods," LibreTexts Engineering, 26 February 2014. [Online]. Available: https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)/09%3A_Proportional-Integral-Derivative_(PID)_Control/9.03%3A_PID_Tuning_via_Classical_Methods. [Accessed 19 June 2023].

[11] Schneider Electric, *Stepper Motor NEMA 17,* PBCLinear.

[12] jackw01, "arduino-pid-autotuner," GitHub, 6 June 2017. [Online]. Available: https://github.com/jackw01/arduino-pid-autotuner/commits/master. [Accessed 20 June 2023].

[13] ProgrammerSought, "RGB to HSV (Verilog)," ProgrammerSought, 2018. [Online]. Available: https://www.programmersought.com/article/31054814413/. [Accessed 19 June 2023].

[14] FPGA4FUN, "I2C slave (method 1)," fpga4fun.com, 7 August 2007. [Online]. Available: https://www.fpga4fun.com/I2C_2.html. [Accessed 19 June 2023].

[15] Cornell Dubilier, ""DSM254Q018W075PB"," Cornell Dubilier, [Online]. Available: https://www.mouser.co.uk/ProductDetail/Cornell-Dubilier-CDE/DSM254Q018W075PB?qs=rQFj71Wb1eXTUSitfenP%2Fw%3D%3D. [Accessed 19 June 2023].

[16] DIODES INCORPORATED, "AP62300/AP62301/AP62300T," January 2021. [Online]. Available: https://www.diodes.com/assets/Datasheets/AP62300_AP62301_AP62300T.pdf. [Accessed 19 June 2023].

[17] Texas Instruments, "TPS54295RSAT," Texas Instruments, 12 May 2018. [Online]. Available: https://www.ti.com/product/TPS54295/part-details/TPS54295RSAT. [Accessed 19 June 2023].

[18] Altium, *[LIVE] How to Achieve Proper Grounding - Rick Hartley - Expert Live Training (US),* Ohio: Altium, 2020.

[19] P. Lab, *Switching Regulator PCB Design - Phil's Lab #60,* YouTube, 2022.

[20] JJRobots, "The B-Robot Evo (the self balancing robot)," JJRobots, 2021. [Online]. Available: https://jjrobots.com/projects-2/b-robot/. [Accessed 20 June 2023].

[21] Deckingman, "HOT FILAMENT AND COLD COFFEE," 18 Aprial 2017. [Online]. Available: https://somei3deas.wordpress.com/2017/04/18/stepper-motor-and-electronics-cooling/. [Accessed 19 June 2023].

[22] J. O'Connell, "How Warm Can You Go?," All 3DP, 9 February 2021. [Online]. Available: https://all3dp.com/2/pla-petg-glass-transition-temperature-3d-printing/. [Accessed 20 June 2023].

[23] J. S. Y. S. Nils Keller, "Predictive Control of a Two-Wheeled Balancing Robot," ETH Zurich, Zurich, 2023.

[24] J. Wu and W. Zhang, "Design of fuzzy logic controller for two-wheeled self-balancing robot," *Proceedings of 2011 6th International Forum on Strategic Technology,* pp. 1266-1270, 2011.

[25] T. Greene, *2. Photo-Voltaic Energy,* London: Imperial College London, 2020.

[26] JLCPCB, "Multilayer high precision PCB's with impedance control," JLCPCB, 2023. [Online]. Available: https://jlcpcb.com/impedance. [Accessed 19 June 2023].

Appendices

Appendix A: Alternative Controllers

LQR

LQR was considered due its high performance as it is a form of optimal control.

LQR is used to calculate the gain matrix K which allows the system to reach its desired state. LQR is a type of optimal control as it seeks to optimise a cost function:

$$\int_{t_0}^{t_1} (X^T QX + U^T RU)dt$$

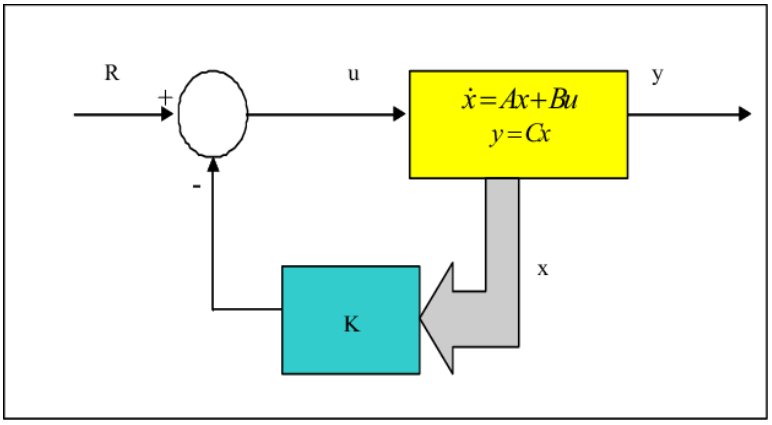


Figure 65 - LQR feedback.

Where X is the characteristic of the system as derived from the free body diagram [8], while U is the matrix of inputs necessary to achieve the desired state. Q and P are values defined by implementation that allow the weighting of the cost function to favour low actuation cost or high performance. U is given by

$$U = -KX$$

LQR requires the value of K that gives the minimum cost function to be calculated. The value of K can be calculated by solving the following Riccati differential equation.

$$A^T P(t) + P(t)A - P(t)BR^{-1}B^T P(t) + Q = -\dot{P}(t)$$

This value of K only needs to be calculated once assuming the model does not change and the values of the desired setpoint do not deviate too greatly from the linearisation point hence the computational cost on the microcontroller is not very high. However, the time needed to recharacterize the physical model every iteration would cause unreasonable implementation time.

Fuzzy Control

Fuzzy control has been used to successfully simulate a TWABR before by Junfeng Wu, [24] and was described as having "good performance in maintaining stability, yielding short settling time and also low overshoot". Fuzzy logic controller provides higher robustness when compared to the FSF method by Bin Junoh [9]. As shown in the diagram below, fuzzy logic separates the measured input values into discrete fuzzy input sets which are then matched against a rule base to output the fuzzy logic output set that is defuzzified to produce the necessary outputs.

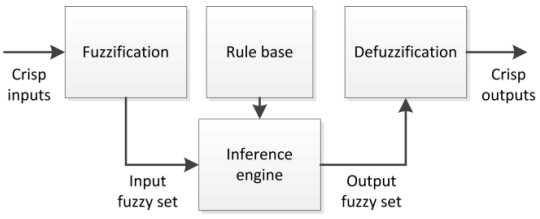


Figure 66 - Fuzzy Control block diagram.

Full state feedback

FSF is similar to LQR with the outputs of the system being fed through a gain matrix instead of the state variables in the system. Therefore, FSF can work without all the state variables being known. FSF allows the placement of the controller's poles anywhere in the left half-plane allowing modification of the

characteristics of the controller while maintaining stability. The gain matrix $\boldsymbol{K}$ is calculated by solving the equation,

$$|s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{BK})| = 0$$

where the poles are already known so that $\boldsymbol{K}$ can be determined.

This calculation would only have to be done once after which the $\boldsymbol{K}$ values can be hard coded into the microcontroller resulting in a low computational cost but requiring an accurate model of the TWABR system.

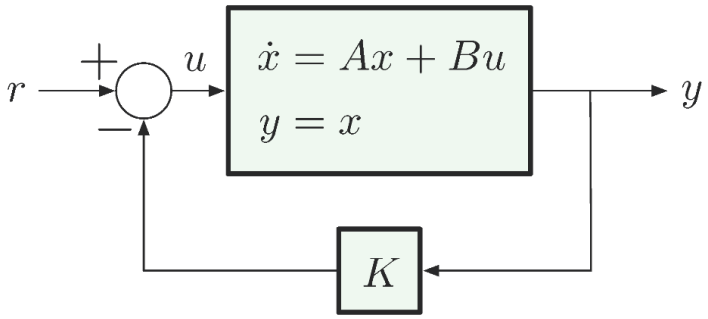


Figure 67 - FSF block diagram.

Model Predictive Control

MPC uses a mathematical model of the physical system to simulate and optimize a discretised control response across a receding horizon. The first response value is then applied to the system and the optimization occurs again; this is repeated. This approach leads to a very high performance against other methods especially when a linearised system is taken further from its linearised point. The main drawbacks would be the high computational cost from running the optimization and simulation in real time on the microcontroller. There is also an added implementation cost from characterising the TWABR model.

PID Control

PID control seeks to minimize the error between the setpoint and measured value by outputting a control value that is a weighted sum proportional to the error itself, the integral of the error and the derivative of the error. The computational cost of a PID controller is low as there are no complex models being simulated in real time. The implementation cost is due to the tuning needed to make the controller functional, the brunt of which can be achieved by using a tuning algorithm such as the Ziegler-Nichols method and fine tuning the gains until the desired response is reached. Changes in the physical design do not force tuning again as the change in PID gains is usually proportional to the magnitude of the physical change (e.g., changing the battery height would not require a drastic retuning of the controller).

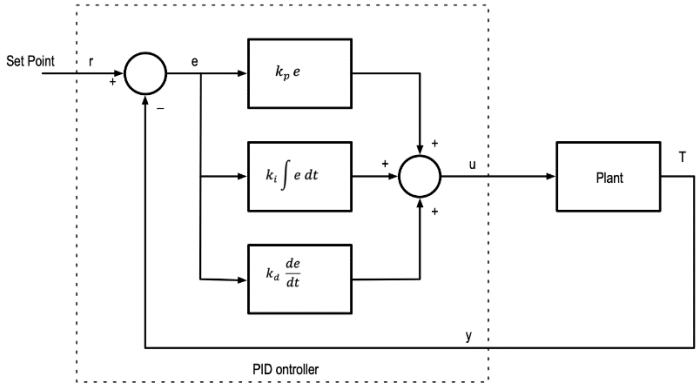
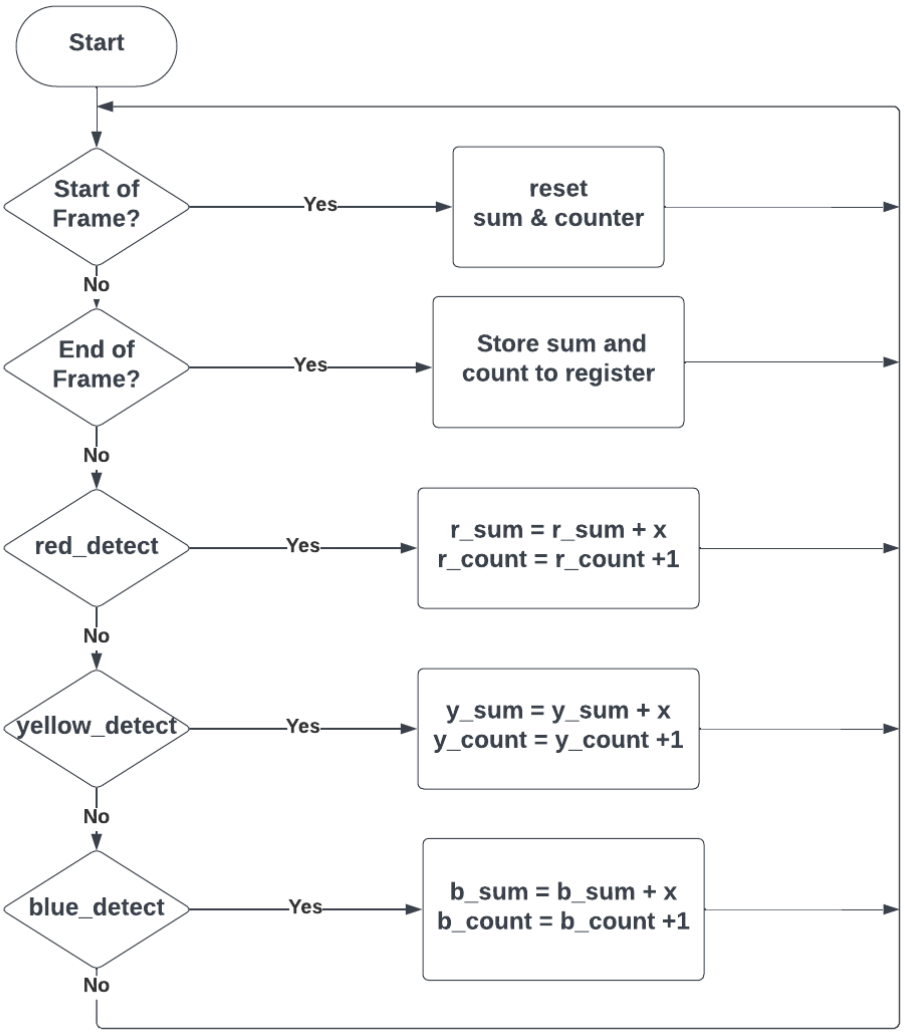


Figure 68 - PID controller block diagram

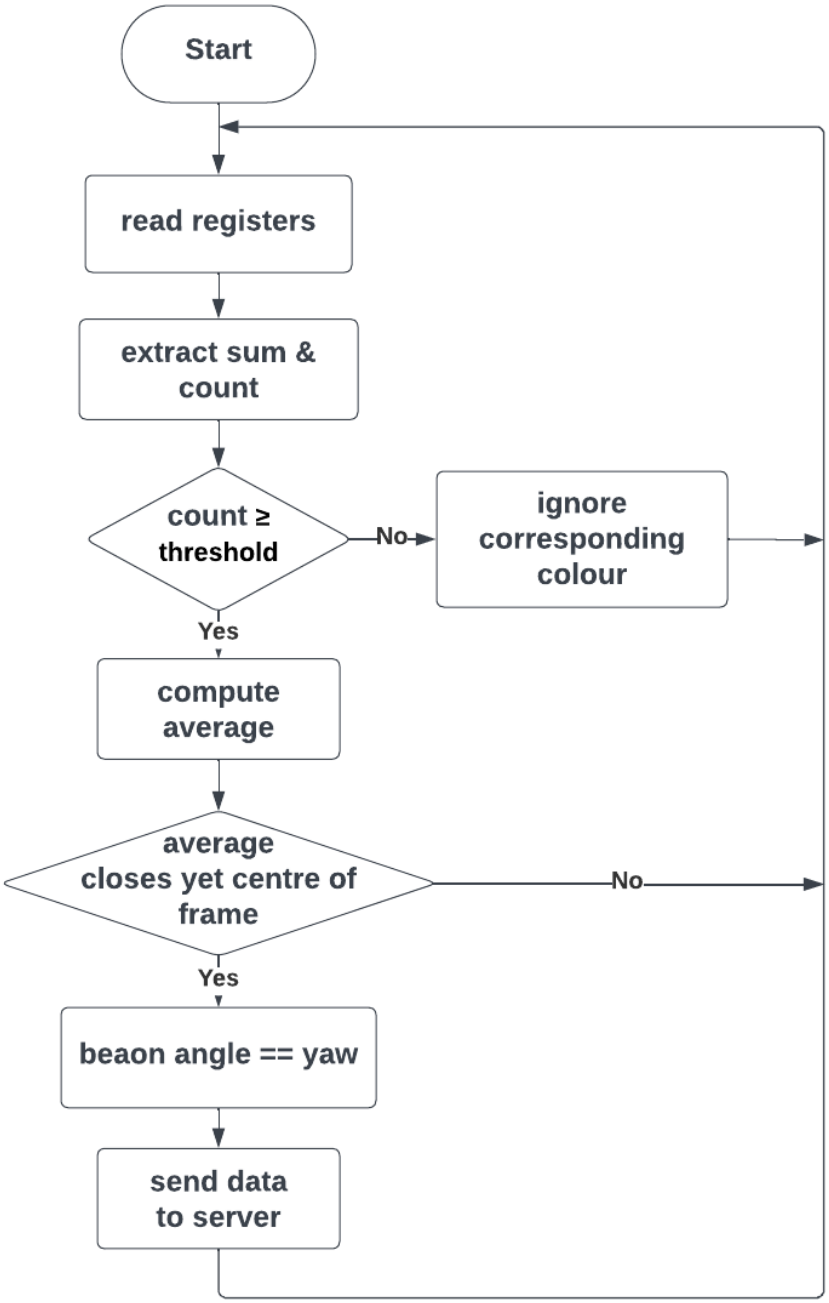
Appendix B: Beacon Detection Flowchart



Appendix C: FPGA Register Map

ADDR	Name	Type
8'd00	count_red[18:11]	Read only
8'd01	count_red[10:3]	Read only
8'd02	count_red[2:0], 2'b00, sum_red[26:24]	Read only
8'd03	sum_red[23:16]	Read only
8'd04	sum_red[15:8]	Read only
8'd05	sum_red[7:0]	Read only
8'd06	count_yellow[18:11]	Read only
8'd07	count_ yellow[10:3]	Read only
8'd08	count_ yellow[2:0], 2'b00, sum_ yellow[26:24]	Read only
8'd09	sum_ yellow[23:16]	Read only
8'd10	sum_ yellow[15:8]	Read only
8'd11	sum_ yellow[7:0]	Read only
8'd12	count_ blue[18:11]	Read only
8'd13	count_ blue[10:3]	Read only
8'd14	count_ blue[2:0], 2'b00, sum_ blue[26:24]	Read only
8'd15	sum_ blue[23:16]	Read only
8'd16	sum_ blue[15:8]	Read only
8'd17	sum_ blue[7:0]	Read only

Appendix D: FPGA Register Driver Flowchart



Appendix E: PV Panel Modelling

To completely model the PV panel, emulations using a power supply unit (PSU) were conducted.

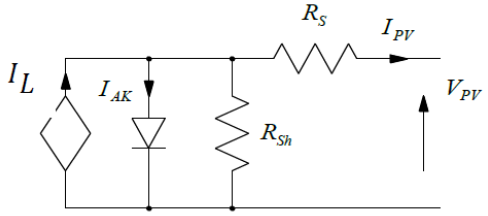


Figure 69 - Equivalent model of a PV cell. [26]

The basic conceptual model of PV panel consists of current source series with resistor, parallel with a diode and a shunt resistor. The metallisation resistance R_s will cause a voltage drop when current pass through it. Parallel resistance R_{sh}, caused by the material configuration of PV panel, will draw significant current when there is enough voltage applied across [25]. (Figure 26)
In this case, a simpler model of the PV panel was applied. The model will only consist of a parallel and series resistor but no diode, both powered by PSU. (Figure 70)

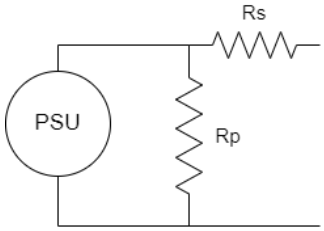


Figure 70 - Simpler model of a PV cell.

Two cases were emulated: sunny and relative sunny.

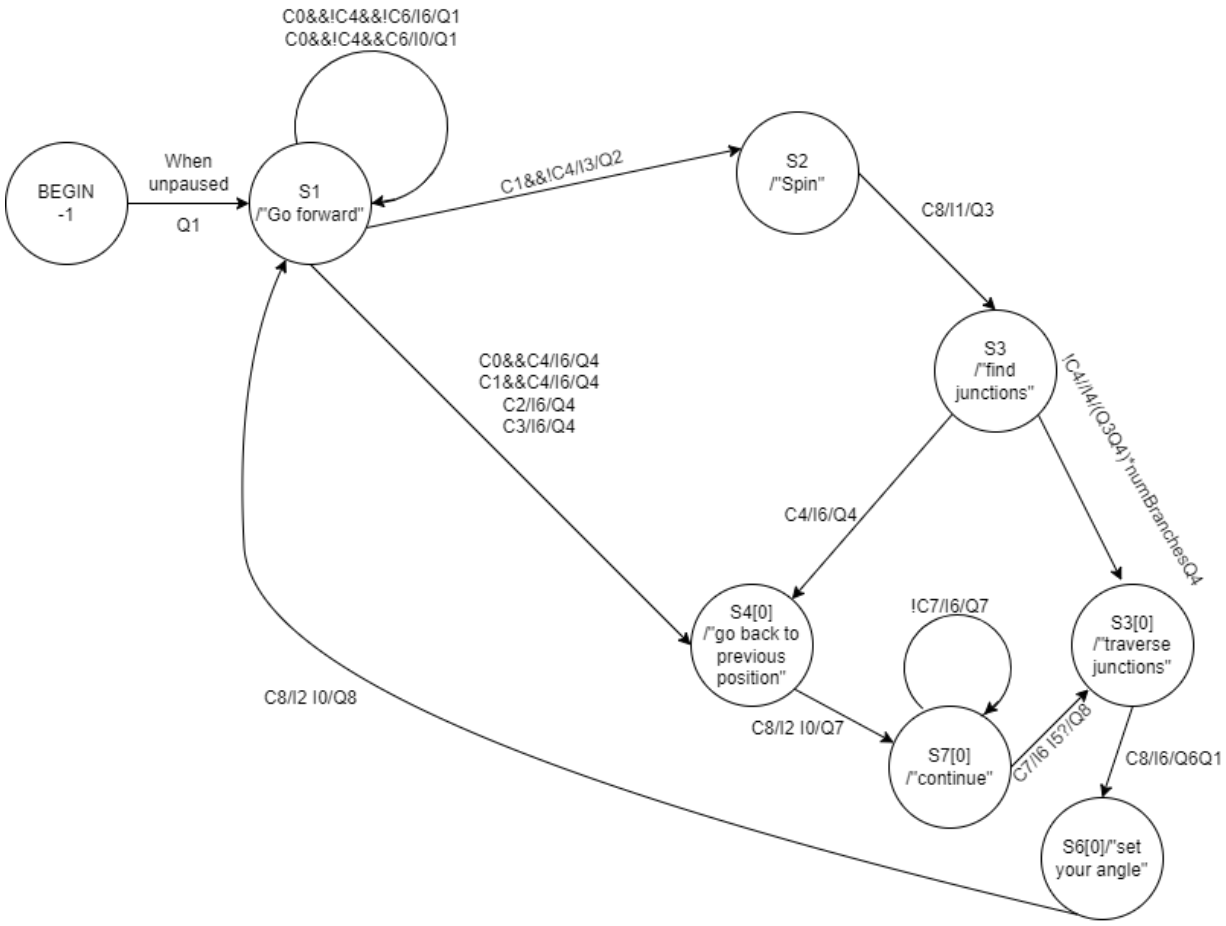
Figure 71 - I-V and P-V characteristics of emulated PV panel

The emulated parallel resistance and series resistance can be obtained from the reciprocal of the gradient shown in IV curve as listed in Table .

Case	R parallel	R series
Sunny	74	1.3
Relative sunny	400	3

Table 11 - Equivalent series and parallel resistances of emulated PV panel.

Appendix F: Maze Solving State Machine



Condition/Instruction/Queue
C0: AT PASSAGE
C1: AT JUNCTION
C2: AT DEAD END
C3: AT EXIT
C4: ALREADY VISITED CURRENT POS
C5: PREVIOUSLY AT PASSAGE
C6: PREVIOUSLY AT JUNCTION
C7: REACHED DESTINATION
C8: ALWAYS

I0: GO FORWARD
I1: SPIN
I2: ANGLE
I3: IDLE
I4: UPDATE POSITION
I5: FINISH
I6: NO INSTR

Q1: prepend S1
Q2: prepend S2
Q3: prepend S3
Q3[0]: prepend S3[0]
Q4: prepend S4[0]
Q6: prepend S6[0]
Q7: prepend S7[0]
Q8: prepend nothing

At each step, remove first item in queue, go to that state, and continue as necessary. Arrows show common transitions, not deterministic transition.

Appendix G: Server API Documentation

Endpoint	Sample request	Sample response (200 OK)	Sample response (400+ not OK)	Notes
/rover	POST {"diagnostics": {"battery":100,"connection":100}, "MAC":1234567, "nickname":"MiWhip", "timestamp":10000, "position":[0,0], "whereat":1, "orientation":90, "branches":[20, 170, 310], "beaconangles":[100,300,250], "tofleft":10, "tofright":10, "battery":100}	{"next_actions":[0, 1, 2, 220, 3, 4, 260, 180, 5], "clear_queue":false} All possible actions are given in sample, realistically only 1 or 2 given at a time. Step – 0, spin – 1, angle – 2, idle – 3, update position – 4, done – 5. Clear queue is used in emergency stop – rover completes carrying out last action and stops, cannot restart rover.	{"error":"Incorrectly formatted request: missing MAC"}	ESP to server communication only. For whereat, 0 is passage, 1 is junction, 2 is dead end, 3 is exit. branches & beaconangles are array of identified angles. All angles in degrees from "north" clockwise. When a rover first connects, it is automatically paused, and must be manually started.
/client/allrovers	GET ~	[{"MAC":1234567, "nickname":"MiWhip", "connected":true, "sessionid":12}, {"MAC":1111111, "nickname":"Lightning McQueen", "connected":false}]	~	Returns list of all rovers. If a rover is connected, its sessionid will be given. If a rover is in the database but not connected, no sessionid will be given.

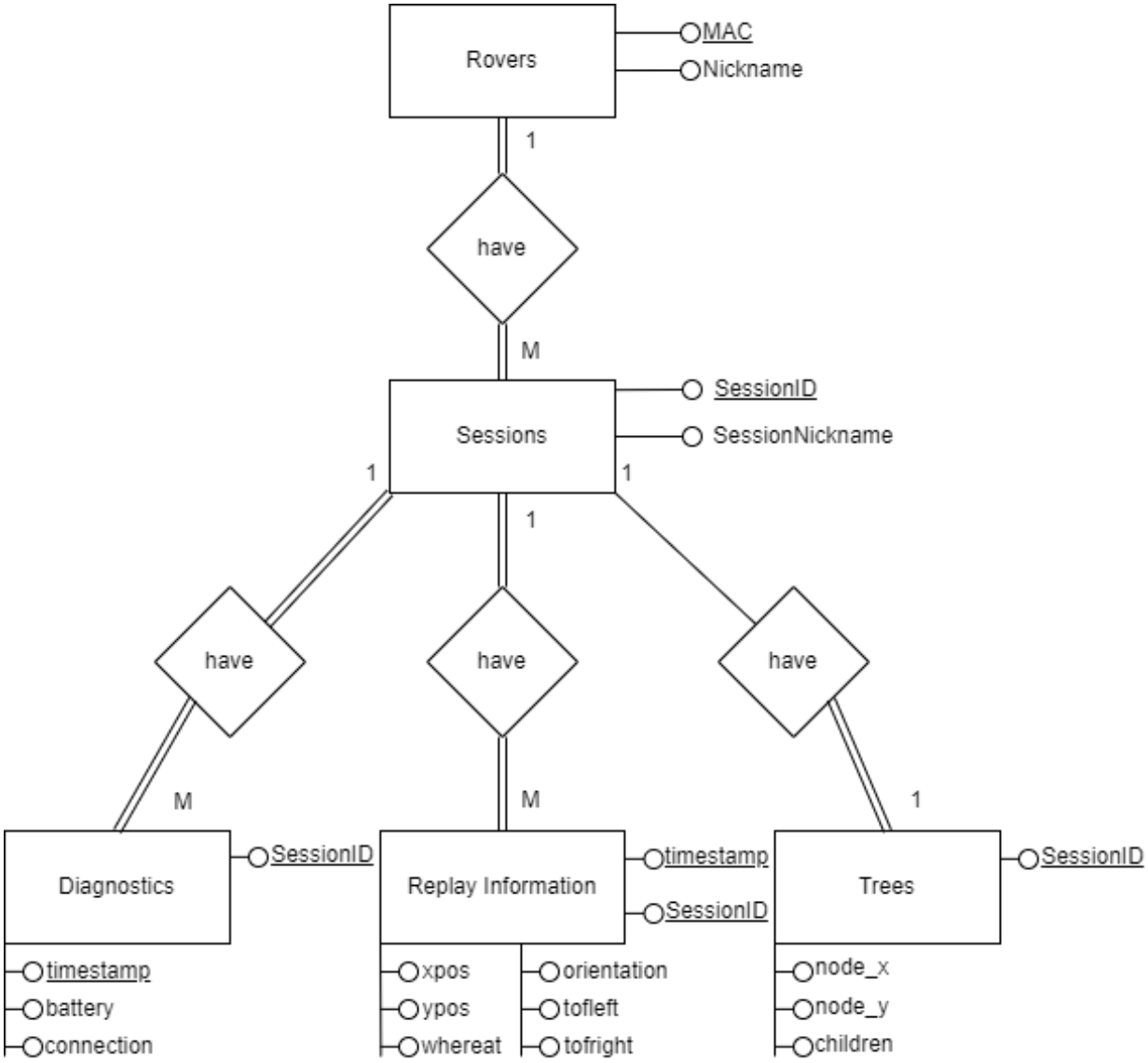
/client/replay	GET
	{"sessionid":23}
		{
		1000:[0, 0, 0, 90, 30, 30],
		2000:[100, 0, 0, 90, 35, 30],
		timestamp:[xpos, ypos, whereat, orientation, tofleft, tofright]
		}
			{"error":"Incorrectly formatted request: missing sessionid"} OR
			{"error":"Session does not exist"}
				Returns all information needed to make a replay on the client side

/client/sessions	GET ~	[{"MAC":1234567, "sessionid":23, "nickname":"Lightning McQueen"}]	~	Returns all session ids, corresponding MAC addresses and nicknames
/client/diagnostics	GET {"sessionid":"10"}	[{"MAC":"1234567", "timestamp":10000, "battery":100, "connection":100}]	{"error":"Incorrectly formatted request: missing/invalid MAC"}	Returns all logged diagnostic data for given session
/client/pause	POST {"MAC":"1234567"}	{"success":"successfully paused rover"}	{"error":"Selected rover does not exist, or is not currently connected"} OR {"error":"Incorrectly formatted request: missing MAC"}	Stops a rover from continuing its way through the maze. Idempotent.
/client/play	POST {"MAC":"1234567"}	{"success":"successfully played rover"}	{"error":"Selected rover does not exist, or is not currently connected"} OR {"error":"Incorrectly formatted request: missing MAC"}	Allows a paused rover to continue making its way through the maze. Idempotent.
/client/sessionnickname	POST {"sessionid":23, "sessionNick":"wr pace"}	{"success":"successfully changed session nickname"}	{"error":"Incorrectly formatted request: missing sessionid or sessionNick"}	Changes session nickname.
/client/rovernickname	POST {"MAC":1234567, "nickname":"YourWhip"}	{"success":"successfully changed rover nickname"}	{"error":"Incorrectly formatted request: missing MAC or nickname"}	Changes rover nickname

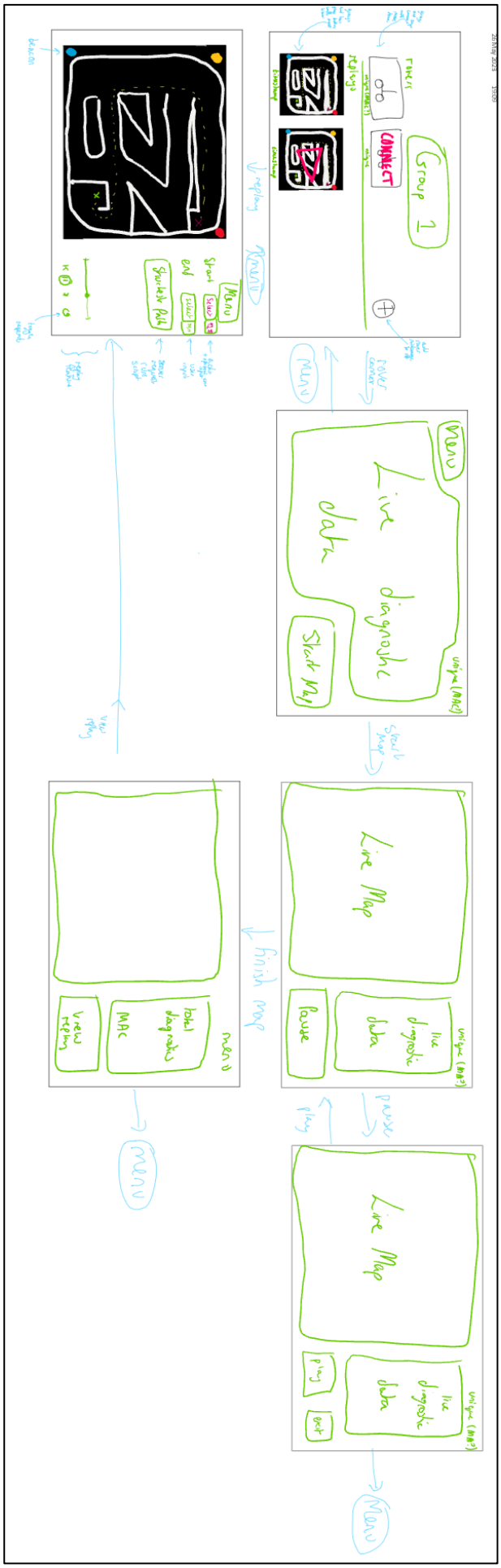
/client/shortestpath	POST
	{"MAC":1234567, "start":[0,0]}
		{
		1:{mapsto: -1, xcoord: 0, ycoord: 0},
		2:{mapsto: 1, xcoord: 100,ycoord: 100},
		3:{mapsto:3}, xcoord=50, ycoord=50
		}
			{"error":"Incorrectly formatted request: missing MAC or start"} OR
			{"error":"Incorrectly formatted request: invalid MAC address"}
				Returns shortest path predecessor graph, ie key is current node, mapsto is next node to travel to to ensure shortest path. Start node has mapsto -1.

/client/estop	POST {"MAC":1234567}	{"success":"estopped"}	{"error":"Missing MAC address"} OR {"error":"Invalid MAC address"}	Emergency stops given rover. Cannot be restarted, must be disconnected and reconnected.
/led_driver/<color> color: red/blue/yellow)	POST {"current":103, "voltage":1.8}	{"success":"received data", "switch":1}	~	Controls switching beacons on/off depending on whether they are being used for a spin or not.

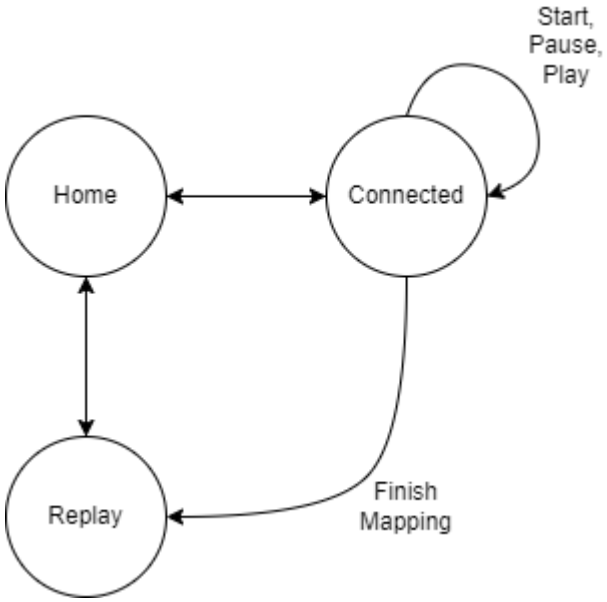
Appendix H: Database Entity-Relationship Diagram



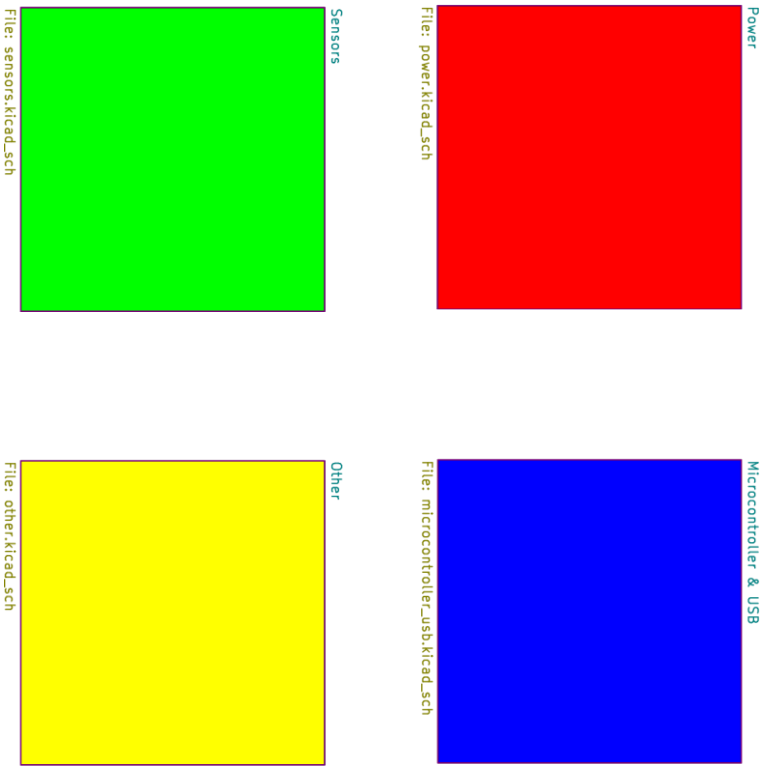
Appendix I: UI Designs



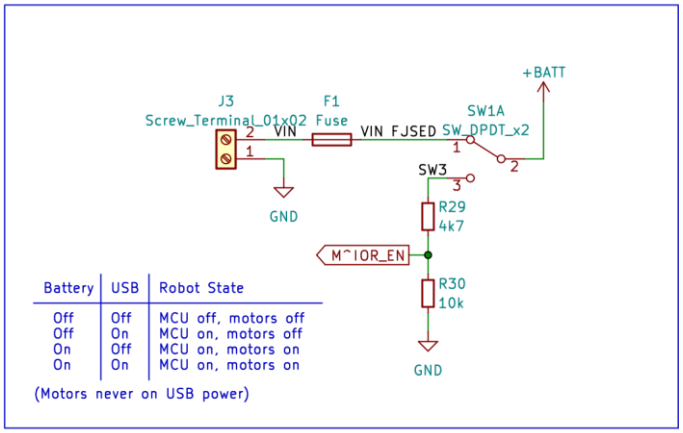
Appendix J: Webpage Navigation State Machine



Appendix K: PCB Schematic



Battery Input and Switch

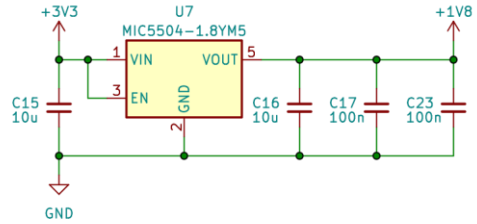


Battery	USB	Robot State
Off	Off	MCU off, motors off
Off	On	MCU on, motors off
On	Off	MCU on, motors on
On	On	MCU on, motors on

(Motors never on USB power)

1.8V Regulator

Low drop out (LDO) regulator that supplies 1.8V power to the IMU

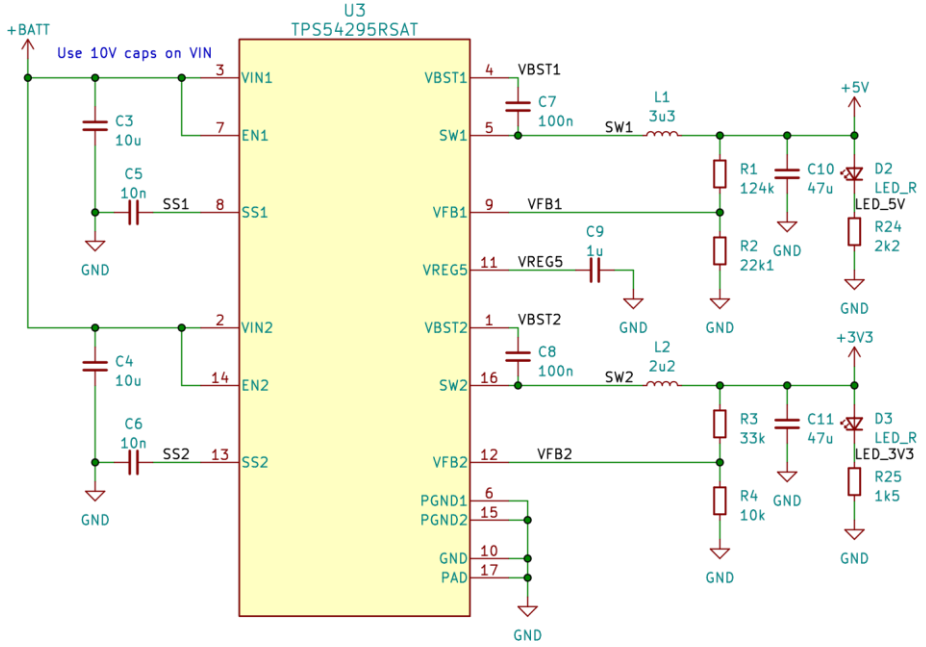


5V Headers

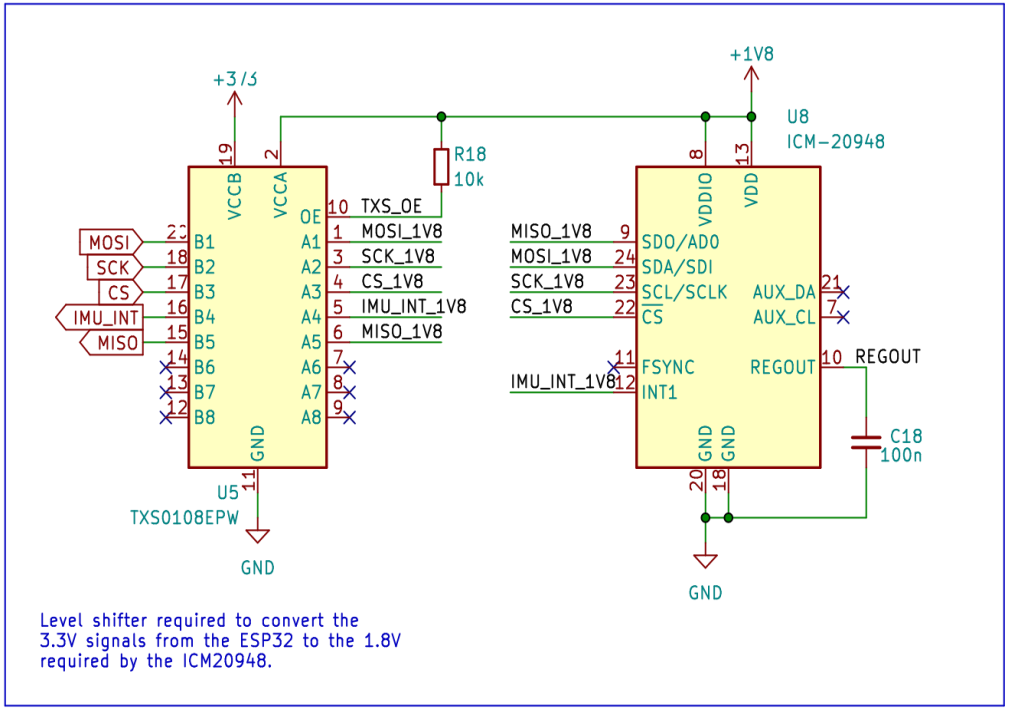
Headers to power FPGA and servo/fan

Buck Converter

3.3V and 5V outputs, 2A per rail



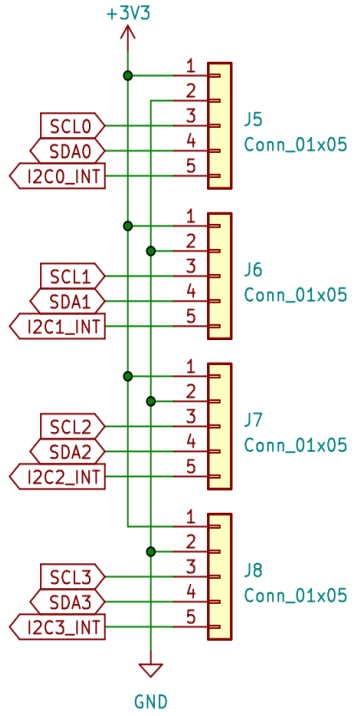
IMU & Level Shifter



Level shifter required to convert the
3.3V signals from the ESP32 to the 1.8V
required by the ICM20948.

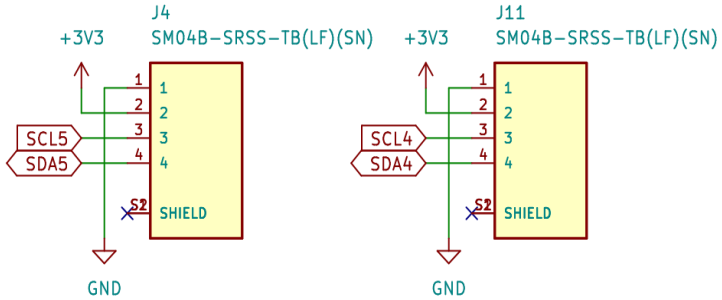
I2C Connectors (Molex)

With interrupt pin.

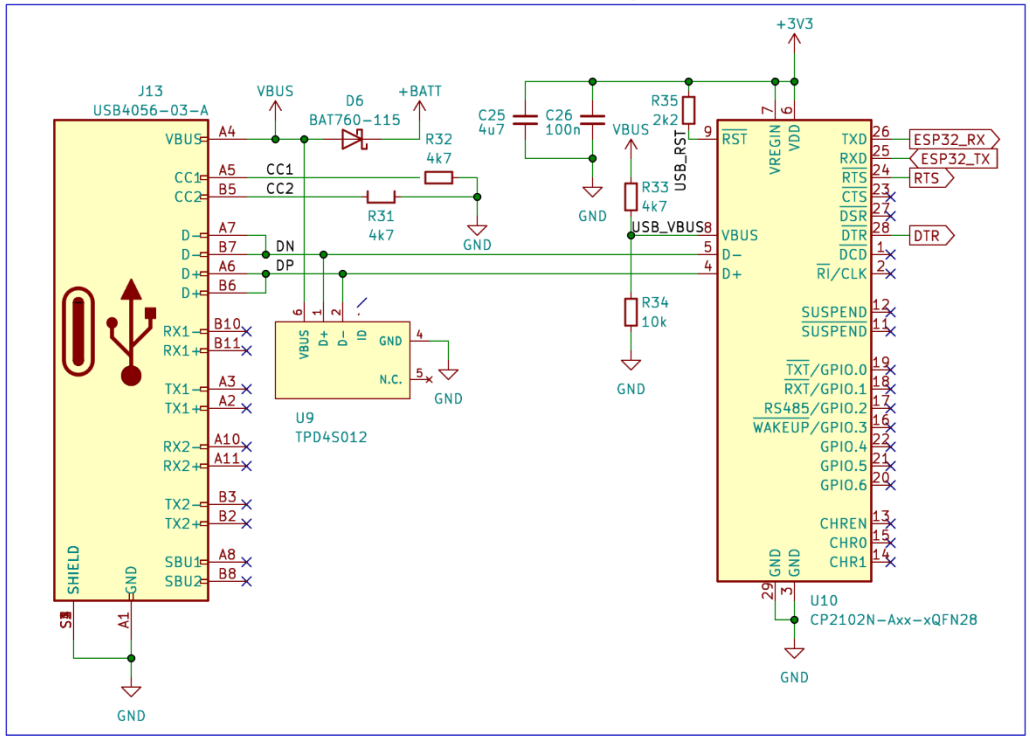


I2C Connectors (JST SH)

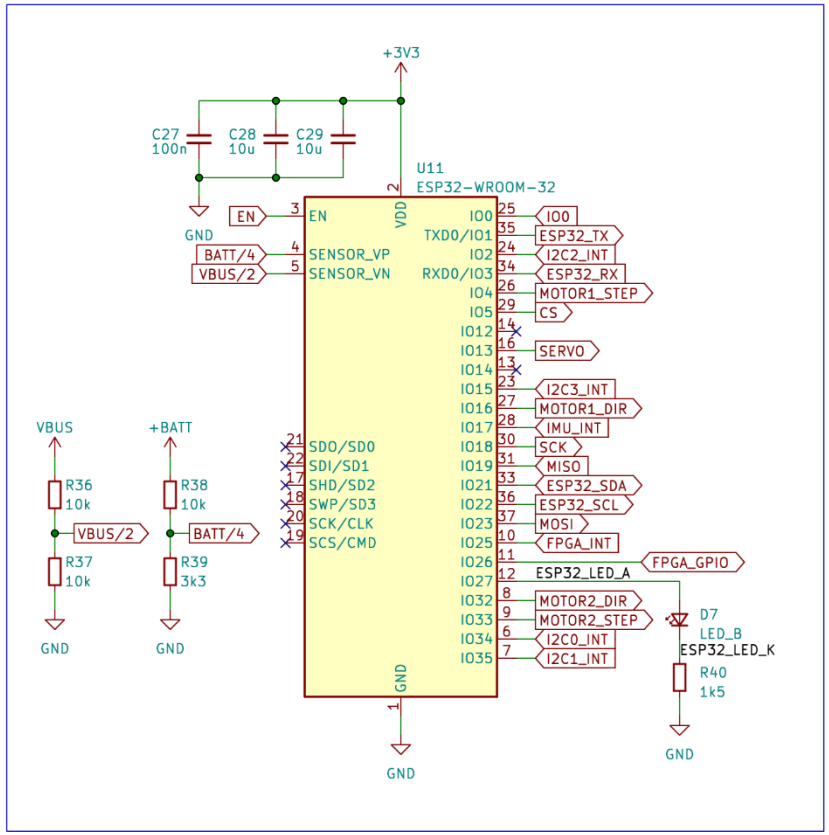
Also called STEMMA QT or QwiiC connectors. No interrupt pin.



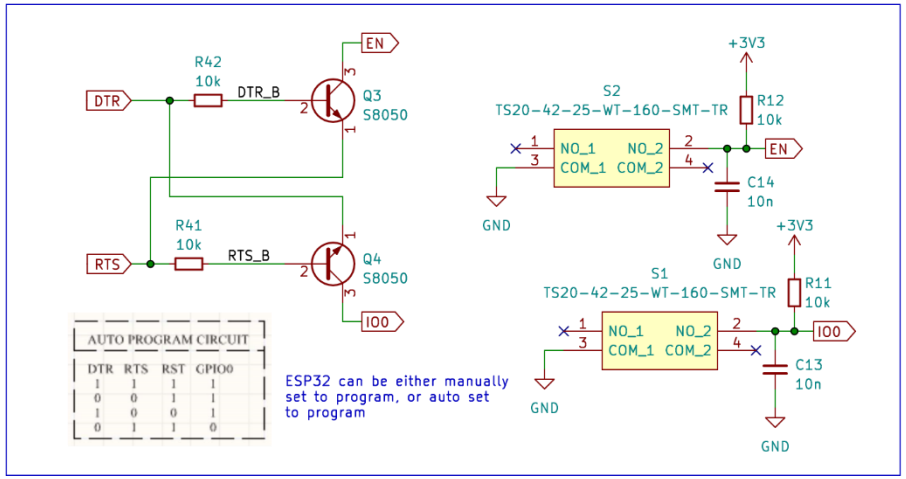
USB-C Connector and USB to UART Bridge



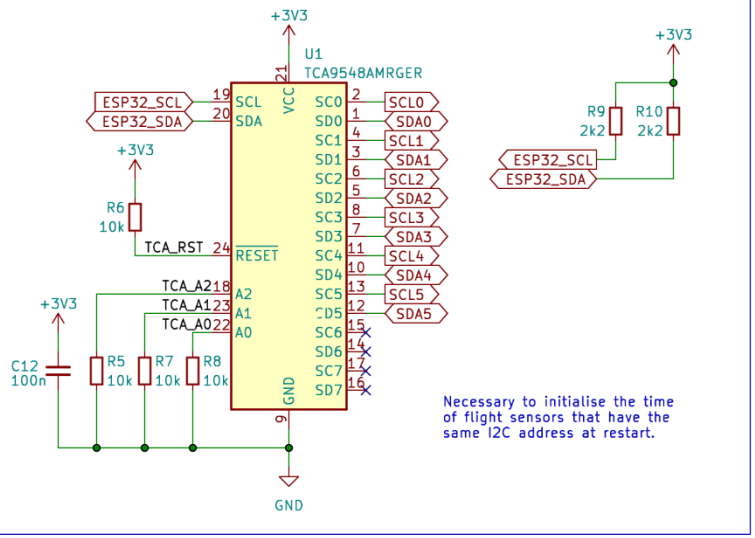
ESP32



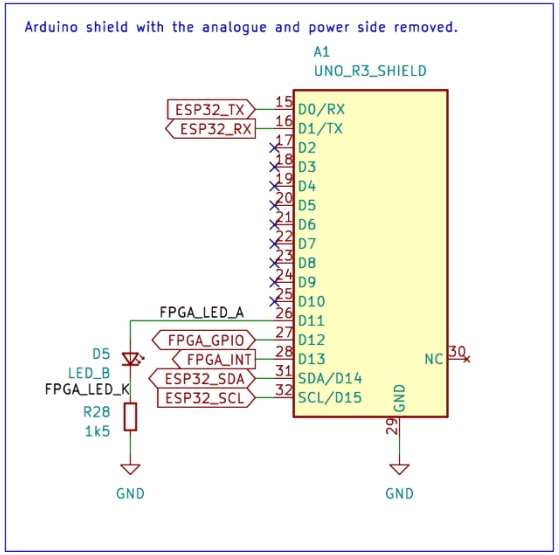
Reset and Boot



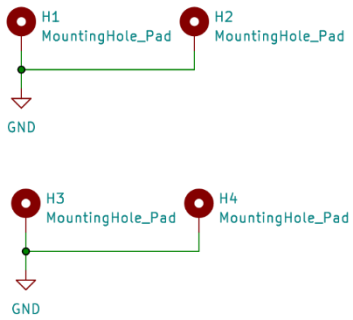
I2C Multiplexer



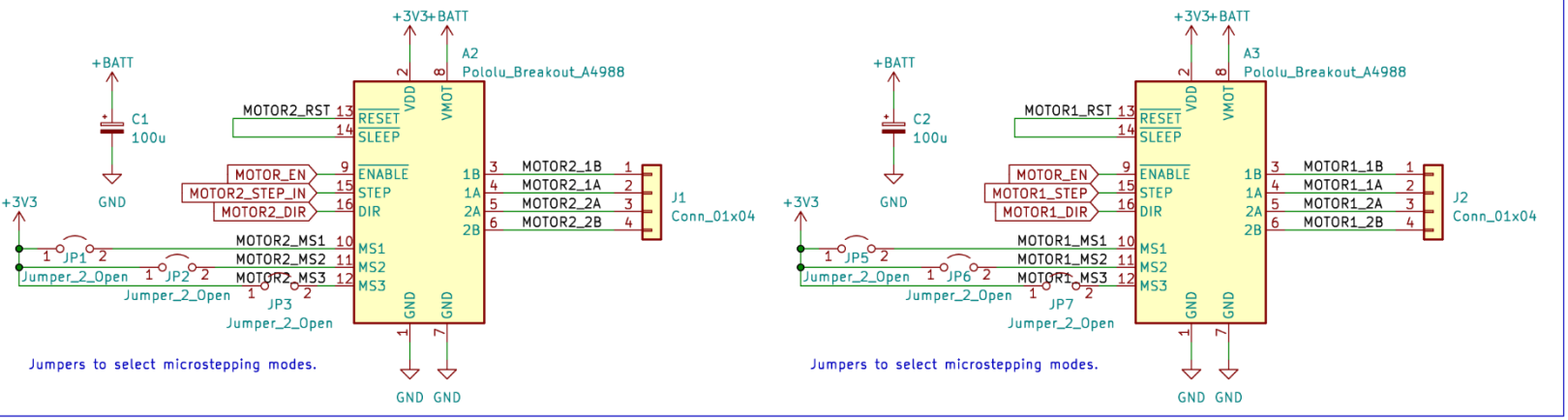
FPGA Connections



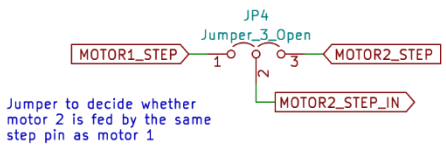
Mounting Holes



Motor Drivers



Motor STEP Selection Jumper



Appendix L: Completed Chassis

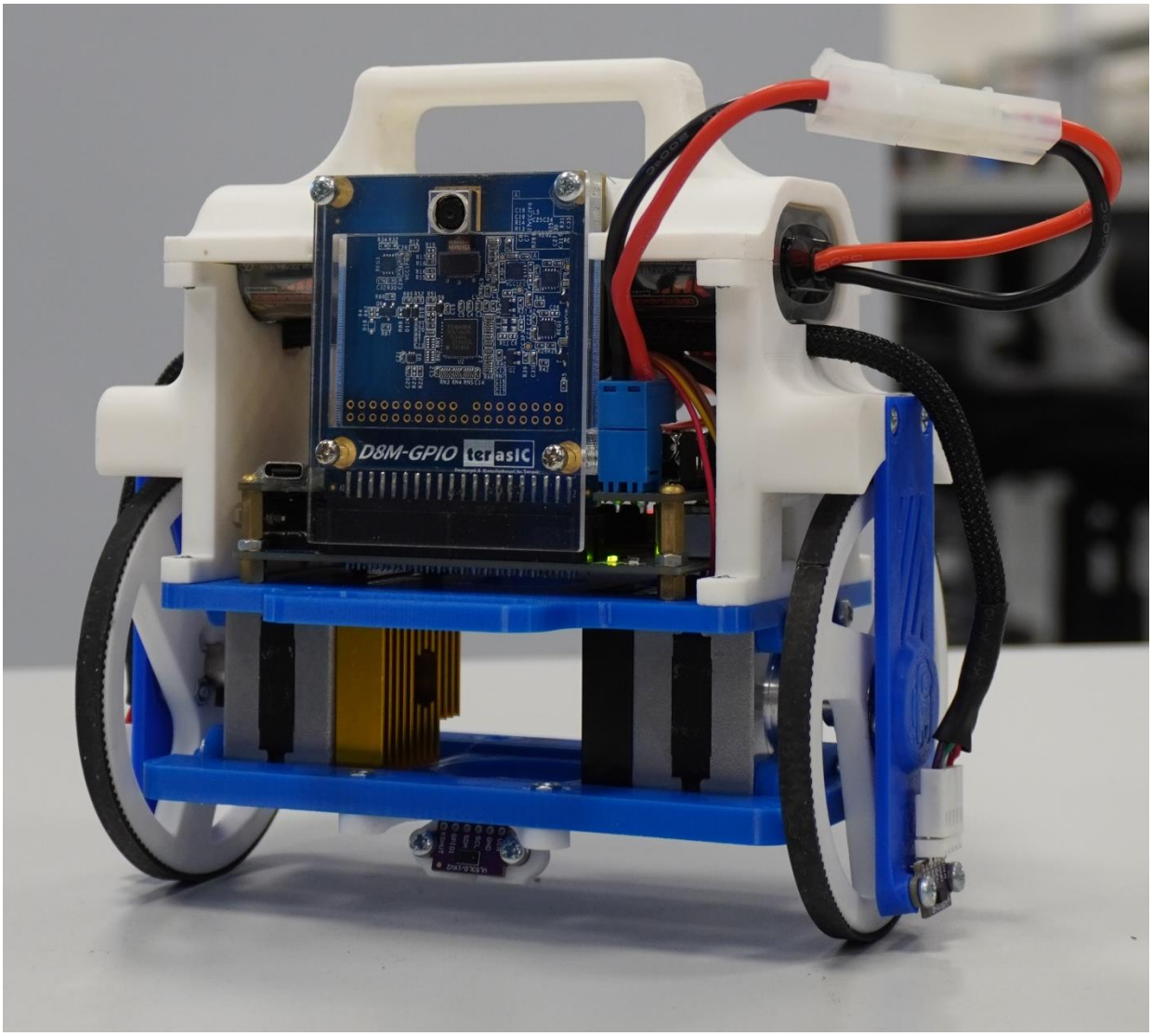


Table of Figures