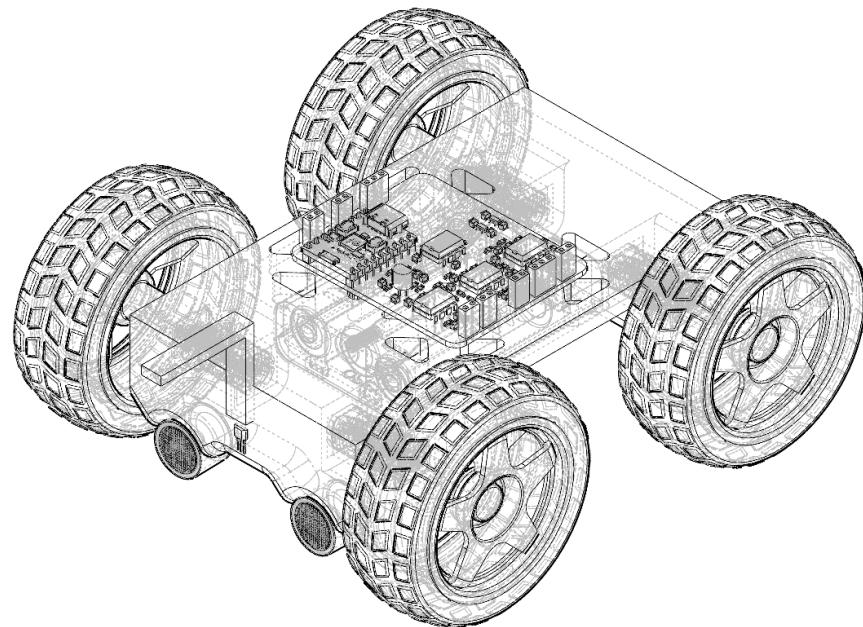


EEE1 Group Project: Seaboat

Name	CID
Archie Eltherington	02568251
Hema Karthikeyan	02556359
Carys Leung	02565398
Wen Li	02569586
Kevin Song	02622710
Simonas Zalatorius	02416877

Word Count: 8565



June 2025

Contents

1	Introduction	3
1.1	Project Management & Planning	3
1.2	Product Design Specification	4
1.2.1	Performance	4
1.2.2	Environment	4
1.2.3	Materials	5
1.2.4	Size & Weight	5
1.2.5	Testing	5
1.3	Signals	5
1.3.1	Infrared	5
1.3.2	Ultrasound	5
1.3.3	Magnetic field	6
1.3.4	Radio	6
1.4	Microcontroller	6
2	Sensors	7
2.1	Infrared	7
2.1.1	Previous Designs	7
2.1.2	Photosensor	8
2.1.3	Photodiode Biassing	8
2.1.4	Photodiode AC Coupling	8
2.1.5	Amplification	9
2.1.6	Analogue to Digital Conversion	9
2.1.7	Decoupling of Voltage References	9
2.1.8	Frequency Measurement	13
2.2	Magnetic Field	13
2.3	Ultrasound	14
2.3.1	Outline of Task & Design Iterations	14
2.3.2	Choosing a Sensor	15
2.3.3	Amplification	15
2.3.4	Envelope Detector & Comparator	16
2.3.5	Obtaining the Name of the Duck	19
2.4	Radio	19
2.4.1	Antenna	19
2.4.2	Radio Front End	20
2.4.3	Process	20
2.4.4	Software	20
3	Control & Web Design	20
3.1	Architecture	20
3.2	Sensor reading	22
3.2.1	Digital Frequency Measurement for Infrared and RF	22
3.2.2	Digital Hall Effect Measurement for Magnetic field	23
3.2.3	Decoding ASCII in UART package for Ultrasound	24
3.3	Motor control	25
3.4	Web page	26
4	PCB	27
4.1	Considerations	27
4.2	Schematic	28
4.3	Implementation	29
5	Body	31

5.1	Drivetrain	31
5.2	Body Modules	32
5.2.1	Motors	32
5.2.2	Motor Driver	33
5.2.3	Power	34
5.3	Chassis	35
5.3.1	Sensor placement	35
6	Further Development	38
6.1	Ultrasound	38
6.2	Web app	38
6.3	PCB	38
6.4	Chassis	38
7	Conclusion	39
8	Appendix	40
8.1	GitHub Repository	40
8.2	Full Size PCB Layout Pictures	40
8.3	Chassis Prototype Pictures	44
8.4	LLM Prompt and Response, Model: Deepseek R1 Qwen Distilled Running on T3chat	47

1 Introduction

1.1 Project Management & Planning

Effective project management was critical to the success of our group project. At first, we immediately divided the workload into two main teams: hardware and software.

Within the hardware team, the responsibilities were further broken down by sensor type. Each team member was assigned to one of the sensing components, including radio, infrared, ultrasonic and magnetism. This division allowed for focused development and parallel progress across different components of the project.

For the software and motor control components, two team members collaborated closely to ensure seamless integration. This pairing allowed for the efficient development of a web-based motor control interface, which provided enhanced responsiveness and control reliability.

To ensure timely progress, we established clear deadlines for the ordering of hardware components early in the project. This enabled hardware teams to begin testing and prototyping as soon as the parts arrived. Our planning process included both short-term weekly goals that are set during our weekly in-person meetings or through WhatsApp, and long-term milestones, visualised and tracked using a Gantt chart. This approach helped align immediate tasks with the overall objectives of the project and important deadlines.

Task\Time	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Find + Order sensors						
Web UI						
Body CAD design						
Design sensor circuit						
Test sensor circuit						
Find + Order motors						
Test motors						
Combine sensor and web UI						
PCB design						
Test rover in Imperial Lake						
Interim presentation						
Reflections on professional practice						
Project report						
Interview, Demo + Competition						

Figure 1: Gantt chart

After receiving insightful feedback during our interim presentation, we realised that there was a significant imbalance in progress between the hardware and software teams. In response, we took time to reflect as a group and decided to rebalance workloads by temporarily reallocating resources. Team members whose tasks were completed began to assist in other areas of the project, particularly with sensor integration and debugging.

This shift marked a transition to a more collaborative and flexible working style. As team members completed their initial responsibilities, they stepped in to help other areas, lending ideas and technical support to address bottlenecks. This cooperative workflow not only accelerated development, but also improved overall system cohesion and team synergy.

Our structured yet adaptable planning strategy ultimately allowed us to maintain a steady momentum, adapt to challenges in real time, and deliver a more cohesive rover design.

1.2 Product Design Specification

The design and development of our amphibious rover were guided by key specifications derived from both the project brief and the practical constraints encountered during prototyping. These specifications ensured that the rover could reliably detect and identify ducks in a controlled demo environment while meeting size, weight, and usability requirements. In the following, the design specifications are detailed under the categories of performance, environmental constraints, materials, size and weight, and testing.

1.2.1 Performance

A primary requirement of the project was the ability to accurately identify various types of ducks by both name and species. Each duck's name was transmitted via ultrasonic signal as a UART packet, which was received by the rover's ultrasonic transducer. The incoming signal was filtered, amplified, and routed to UART2 of the ESP32 microcontroller. The data is then decoded from hexadecimal to ASCII characters to reconstruct a 4-character identifier, always beginning with a '#' and ending with a null character.

For species detection, we implemented infrared, radio frequency, and magnetic sensors. The system was designed to successfully classify a duck's species using any two of the three available sensing methods. This redundancy improved reliability and allowed for continued operation even if one type of sensor failed or underperformed in certain environmental conditions.

The rover was also designed for high manoeuvrability using a four-wheel drive system powered by high-RPM motors. This allowed for smooth navigation through the arena, particularly around obstacles and narrow passages. In terms of usability, the rover was controlled through a web-based joystick interface, which also displayed real-time feedback on duck names and species. This user interface was designed to be intuitive, allowing us to make quick decisions during the demo.

1.2.2 Environment

The demo environment imposed several physical constraints that directly influenced our design. The rover had to be capable of passing through a lakeside entrance that was not wider than 250mm and underneath an arch bridge with a maximum clearance of 150mm and a minimum of 110mm. To accommodate these limitations, our design maintained a compact profile that remained within these dimensions without compromising functionality.

In addition to size constraints, the rover was required to weigh less than 750 grams. Ducks classified near the lakeside would not communicate if approached by a heavier rover, so maintaining a lightweight design was essential to ensure the successful identification of all types of ducks. These environmental restrictions were continuously referenced throughout the design and iteration phases to ensure compatibility with the final demo setup.

1.2.3 Materials

The rover chassis was constructed using 3D printed PLA. To reduce overall weight while maintaining structural integrity, we printed with a low infill percentage. The chassis design also featured holes, which served two purposes: minimising material use and allowing clean, organised cable routing.

PLA was selected for its balance between printability, weight, and mechanical strength, as well as its availability. This material choice also allowed for rapid prototyping and modification, enabling us to iterate the design rapidly and efficiently.

1.2.4 Size & Weight

Our goal was to keep the rover compact, lightweight and, ideally, square in layout to support an even weight distribution and agile movement. A square chassis improved turning performance and ensured better clearance while navigating the demo environment.

The choice of switching from the Metro M0 Express microcontroller to the smaller and lighter ESP32 helped us further reduce the rover's size and weight. This decision also improved integration with our sensor systems due to the additional UART ports and wireless capabilities of ESP32. The overall weight reduction contributed to faster movement, improved energy efficiency and ensured we remain within the sub-750g requirement for duck interaction.

1.2.5 Testing

The testing followed an iterative process, starting with basic functional checks of the movement of the rover and control via the web interface using a joystick. Once the rover's mobility was completed, we tested the placement of sensors to ensure optimal positioning and reduce interference for reliable detection of ultrasound, magnetic fields, infrared, and radio signals.

The CAD design was revised multiple times, in particular to accommodate the ultrasonic sensor mounted on the underside of the chassis. These changes were tested in successive prototypes to ensure signal clarity and accessibility. Each test informed further refinements, leading to a final design that balanced stability and sensor accuracy in real-world conditions.

1.3 Signals

To accurately determine the characteristics of each duck, the rover must detect and interpret several types of electromagnetic signals emitted from specific locations on the duck. Each signal presents unique detection challenges, which are summarised below.

1.3.1 Infrared

An infrared signal pulsing at a characteristic frequency is emitted from the body of the duck. The pulse frequency correlates with the species.

Detection challenges include:

- The infrared power is relatively low, making detection more difficult
- Ambient light, particularly from AC-powered lighting, introduces interference which may overlap with the duck's signal
- The signal has a narrow pulse width of $50\ \mu s$, requiring a fast response sensor and minimal filtering to preserve frequency accuracy

1.3.2 Ultrasound

The duck's name is transmitted using an ultrasonic signal at a carrier frequency of 40 kHz, modulated by amplitude-shift keying. This signal is emitted from the body of the duck and must be detected in close proximity using an ultrasonic transducer.

Detection challenges include:

- The signal attenuates rapidly with distance and is quite susceptible to noise, which requires the receiver to be placed close to the duck
- Demodulation is required to extract the binary signal, which involves thresholding the amplitude using a comparator
- The comparator threshold is highly sensitive and difficult to tune due to varying signal amplitudes and ambient noise

1.3.3 Magnetic field

A permanent magnet embedded in the duck's head generates a static magnetic field, orientated either "up" or "down" to indicate species.

- Field strength decreases significantly with distance, requiring close proximity for reliable detection
- Some sensor, such as reed switches, are binary and may not provide the directional sensitivity needed

1.3.4 Radio

The duck emits an AM-modulated radio frequency signal, also characteristic of its species. This signal is emitted from the body of the duck and must be detected with a tuned antenna.

- The signal amplitude is low and requires amplification prior to demodulation
- The antenna design must be tuned accurately to the carrier frequency to ensure sensitivity and selectivity

1.4 Microcontroller

During the project, our approach to microcontroller selection and implementation evolved significantly as we refined both our technical requirements and hardware constraints. Initially, we conducted most of our testing using the ESP32-WROOM-32 module. This platform was chosen for its accessibility, performance, and rich set of features. In particular, the ESP32-WROOM-32 offers dual-core processing, which enabled us to decouple critical functionalities: running the web server on one core while managing motor control on the other. Additionally, it provided ample GPIO pins and substantially more memory and processing power compared to our initial alternative, the Metro M0 Express.

The Metro M0 Express, while suitable for basic prototyping, is significantly less powerful than the ESP32 platform. With only a single-core processor, 32 KB of RAM, and 256 KB of flash storage (paired with an external WiFi module), it proved inadequate for handling concurrent operations efficiently. In contrast, the ESP32-WROOM-32's dual-core architecture (with 514 KB RAM and 448 KB flash) allowed for a more scalable and responsive system, particularly valuable in development and testing phases.

As we progressed toward a final prototype, we re-evaluated our hardware priorities, specifically with regard to size, power efficiency, and cost. This led to the decision to switch to the ESP32-C3 SuperMini. Despite having only a single core, the module offered sufficient computational performance for our optimised software, which no longer required task separation into individual cores. We had identified that pinning web and sensor/motor control to separate cores was inefficient and unnecessary for our use case. Consolidating these functionalities onto a single core not only simplified the design but also reduced overhead.

Furthermore, the ESP32-C3 SuperMini features a significantly smaller footprint and fewer GPIO pins, which fits our finalised design requirements while offering a much more compact and lightweight solution. Compared to the earlier Metro M0 Express paired with an external WiFi module, the ESP32-C3 SuperMini is much more efficient in terms of size, integration, and power consumption.

2 Sensors

This section discusses the hardware implementation of the sensors circuit design while "Sensor Reading" revolves around the software side.

2.1 Infrared

According to the provided project requirements, only two out of the four species of ducks emit an infrared pulse signal at a frequency ranging from 135Hz to 1 kHz. Hence, our goal with the infrared sensor was to develop a means to detect the presence of an infrared signal and measure its frequency.

2.1.1 Previous Designs

Initial designs for the infrared front-end revolved around the architecture employed by some commercial IR receivers for remote control applications. Specifically, the analogue photosensor signal is first converted into a voltage, then amplified, filtered, and converted into a digital signal [1] [3]. Each design mostly differed in the implementation and ordering of the individual stages. For example, the amplification stage may utilise inverting or non-inverting amplifier configurations, and the stages may be AC or DC coupled. One such early design is illustrated in **Figure 3**:

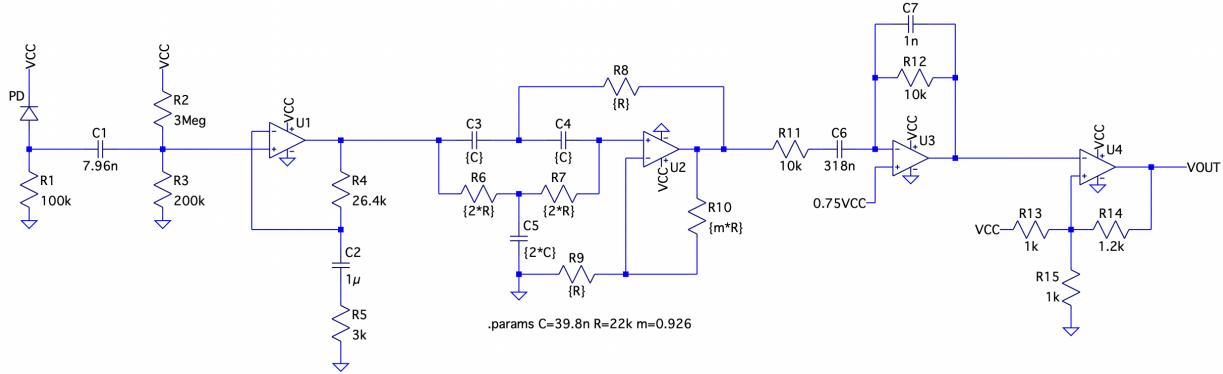


Figure 2: Initial Infrared front-end design

This design was focused on reducing noise in the input signal as much as possible prior to the analogue-to-digital conversion. The Twin-T notch filter was configured to remove noise at 100 Hz caused by room lighting. However, due the filter's DC gain of 1.8, the doubling of the bias in the input signal necessitated different biassing voltages for each stage. After testing showed that the 100 Hz noise was insignificant compared to the desired pulse signal, the notch filter was ultimately removed from the final design.

Likewise, the active bandpass filter was also found to be redundant. An additional active bandpass filter was utilised in multiple designs to achieve a sharper roll-off of -40 dB per decade in both the low and high frequency regions. However, after receiving feedback during the interim presentation suggesting that the bandpass filter was unnecessary, it was confirmed that the circuit was functional without it.

After repeated simplification, the final design (**Figure 3**) was left with three stages: **Photodiode Biassing and AC Coupling, Amplification and Digital Conversion**.

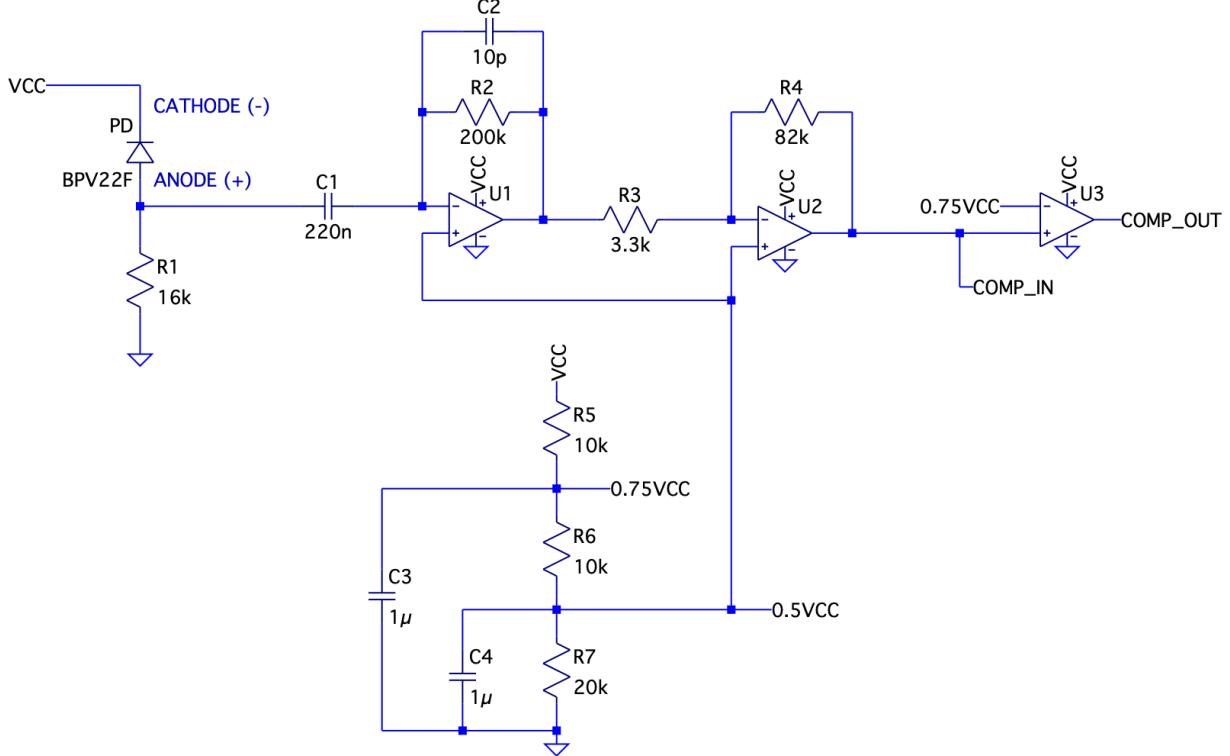


Figure 3: Final Infrared front-end design

2.1.2 Photosensor

To counteract external interference from sunlight and room light, the BPV22F photodiode was chosen as the photosensor for its tight matching with the transmitted IR signal. The photodiode exhibits peak sensitivity at the 950 nm wavelength of the transmitted signal, and its optical filter has a narrow passband of 870 nm to 1050 nm. In addition, the large viewing angle of 60° permits greater misalignment of the duck and photodiode.

2.1.3 Photodiode Biassing

A photodiode can be modelled as a current source in parallel with a parasitic capacitance, known as the junction capacitance C_j [6, p. 2]. Since the junction capacitance and the load resistance form a first-order low-pass filter, the rise and fall times of the output signal depends on the time constant $R_L C_j$. A longer time constant produces more rounded edges in the output waveform, which can cause rapid transitions in the comparator output at the end of the analogue stages.

Hence, to improve the frequency response of the infrared sensor, a reverse voltage of approximately -3.3V is applied to the photodiode. The reverse bias widens the depletion region of the photodiode and reduces C_j from 70 pF to 25 pF compared to the zero bias [7, p. 3].

2.1.4 Photodiode AC Coupling

During testing, it was found that clouds passing overhead can create low-frequency changes in the ambient light intensity, which in turn shifts the DC level of the photodiode current. For this reason, the photodiode is AC coupled to the amplification stage. In the input stage, the RC network formed by R1 and C1 divides the photodiode current into two branches. Photocurrents above the RC cutoff frequency mostly flow into the input of the amplifier, while lower frequency and DC currents are diverted into ground via the resistor.

Since the frequency of the IR pulse signal ranges from 135 Hz to 1 kHz, the cutoff frequency was set to 45 Hz to minimise the attenuation of the useful signal.

2.1.5 Amplification

The amplification stage is used to increase the amplitude of the input signal to a sufficient level for the comparator stage. First, a transimpedance amplifier (TIA) converts the current output of the photodiode into a voltage. Although a simple resistor achieves the same functionality, the TIA was preferred because of its lower input impedance for the same gain. It can be shown that the input impedance of the TIA is approximately the feedback impedance divided by the open-loop gain of the op-amp:

$$V_{out} = A_{OL}(V_+ - V_-) \approx -I_{in}Z_f$$

Assuming the non-inverting input is grounded, $V_+ = 0$ and $A_{OL}V_{in} \approx I_{in}Z_f$. Therefore,

$$Z_{in} = \frac{V_-}{I_{in}} \approx \frac{Z_f}{A_{OL}}$$

When a 10 pF capacitor is used in the feedback network, the TIA produces a -3 dB bandwidth of 79.6 kHz. In contrast, a simple 200 $k\Omega$ resistor only achieves 31.8 kHz at a junction capacitance of 25 pF.

Because the output voltage of the TIA was too small to trigger the comparator, an inverting amplifier is used to provide an additional gain of +28 dB. The use of DC coupling between the two amplifiers eliminates the need for an AC coupling capacitor.

2.1.6 Analogue to Digital Conversion

Finally, a comparator (U3) converts the output of the amplification stage into a digital signal accepted by the digital input of the ESP32. Since the comparator input voltage ranges from approximately $\frac{1}{2}V_{CC}$ to V_{CC} , the threshold has been set halfway to $0.75V_{CC}$.

2.1.7 Decoupling of Voltage References

The decoupling capacitors, C3 and C4, were added to ensure circuit stability. Prior to their inclusion, high-frequency oscillations occurred at the comparator input. As a result, the comparator output switched rapidly between "low" and "high" at a rate of 530 kHz, rendering it unusable for frequency measurement.

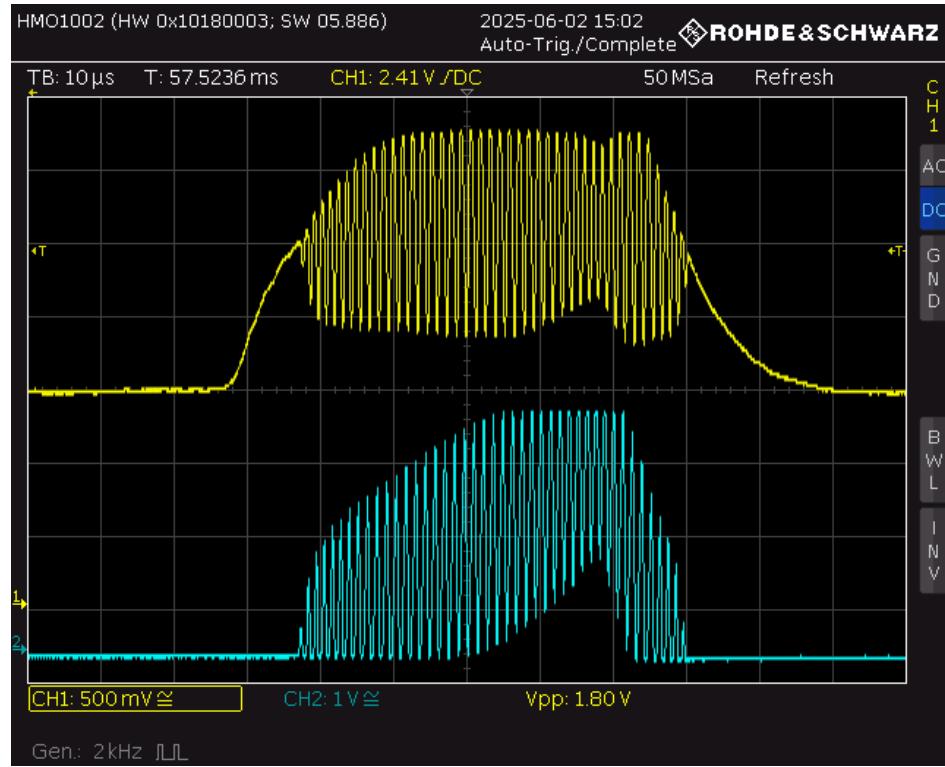


Figure 4: High-frequency oscillations in comparator input and outputs

The cause of the oscillation was suspected to be parasitic capacitance between the resistor ladder and the op-amp inputs, which creates a positive feedback loop. This hypothesis was supported because oscillations were also present on the $0.75V_{CC}$ and $0.5V_{CC}$ outputs of the resistor ladder (**Figure 5**), despite the extremely low input bias currents of the op-amps. Since virtually no *DC* current should be drawn from the resistor ladder, it must be capacitively coupled to one of the op-amp's terminals.

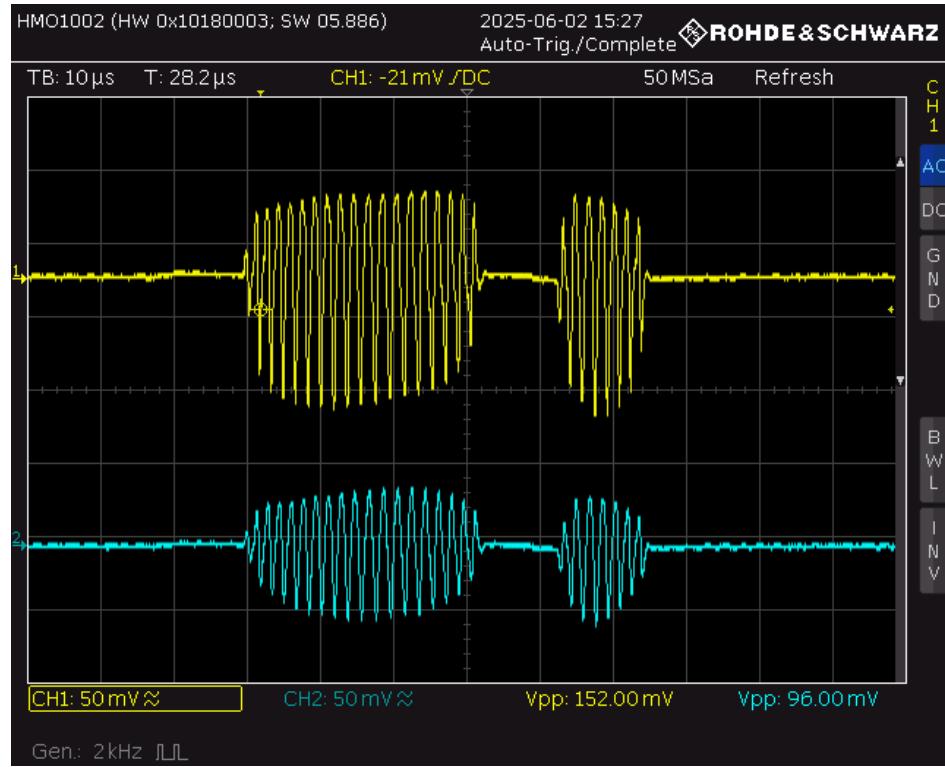


Figure 5: Oscillation present on $0.75V_{CC}$ and $0.5V_{CC}$ references

To further investigate the problem, each op-amp in the circuit was supplemented with the manufacturer's input capacitance model [Microchip Reference]. The subsequent LTSpice simulation exhibited oscillations resembling those of the real-life circuit, albeit at a higher frequency of 1.5 MHz versus 530 kHz. **Figure 6** shows the oscillations on the comparator input.

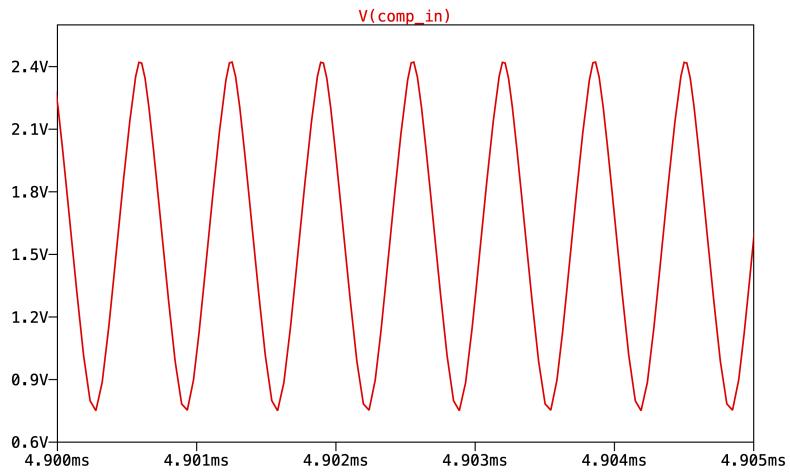


Figure 6: LTSpice simulation showing similar oscillatory behaviour at comparator input

By connecting a $1\mu F$ capacitor from each output of the resistor ladder to ground, the oscillations were completely eliminated, as evidenced in **Figure 7**.

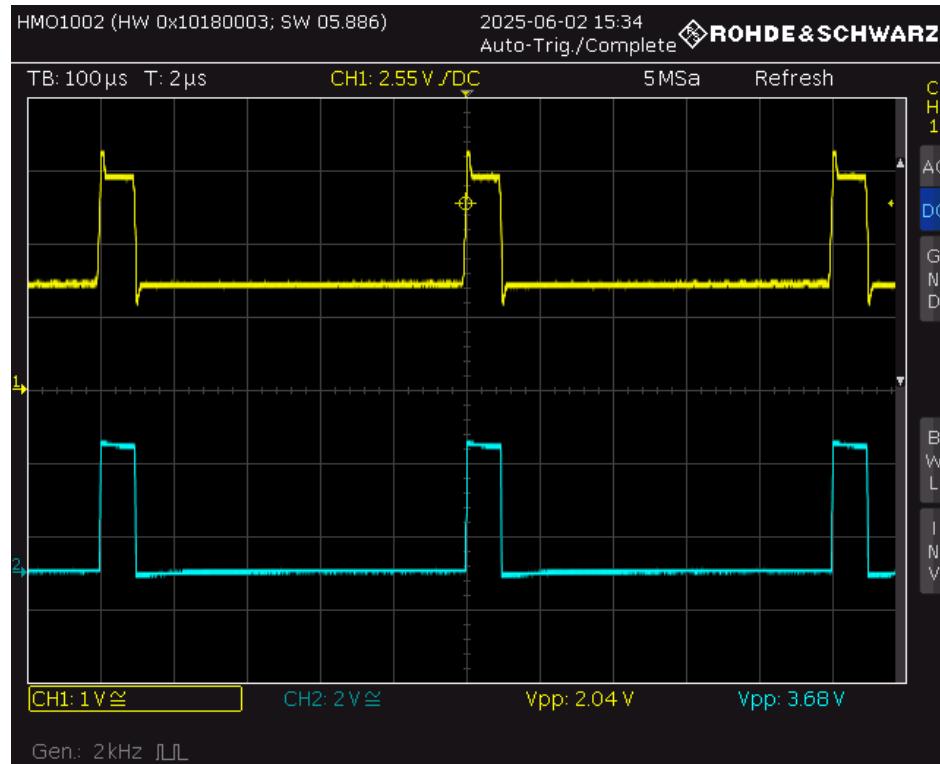


Figure 7: Instability eliminated after adding decoupling capacitors to resistor ladder

2.1.8 Frequency Measurement

A software solution was used to determine the frequency of the IR pulse signal. Since the implementation covers both Infrared and Radio detection, it is discussed later in the section "Sensor Reading".

2.2 Magnetic Field

To detect the orientation of a magnetic field, specifically whether it is pointing "up" or "down", we evaluated and implemented a hall effect sensor based solution. Initial testing was carried out using the hall effect sensor onboard the ESP32-WROOM-32 module. However, it was found that the onboard sensor exhibited a very low sensitivity. The sensor readings only showed a noticeable change when the magnet was in direct contact with the metallic lid of the ESP32 module. This made the onboard hall sensor unsuitable for our application.

To improve detection reliability, an external analogue hall effect sensor was selected. Unlike digital hall sensors, which have fixed internal thresholds for magnetic field detection, the analogue variant provides continuous output values, allowing us to define custom thresholds and reliably distinguish between "up" and "down" magnetic field directions in our use case. Taking these considerations into account, we chose the A1324LUA-T hall sensor for its low noise, linear analogue output, and temperature-stable quiescent voltage [8], which makes it ideal for sustained operation during our demonstrations.

Implementation of the hall sensor was straightforward, requiring only a $10k\Omega$ resistor to act as a pull-up resistor. This ensured a defined logic level at the output, making it less susceptible to noise and resulting in a more stable signal.

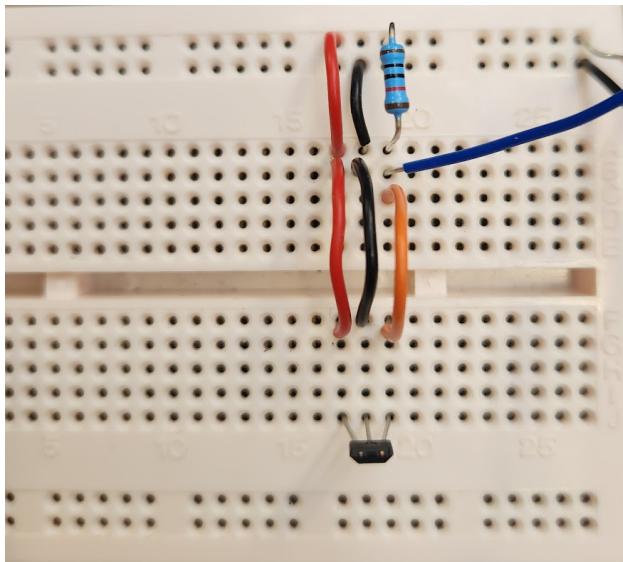


Figure 8: Circuit design on breadboard

We empirically defined two thresholds for classification of the magnetic field in the duck's head:

Up: $V_{OUT} > 2.8V$

Down: $V_{OUT} < 2.2V$

N/A: $2.2V < V_{OUT} < 2.8V$ (No significant magnetic field detected)

These thresholds were chosen on the basis of the analogue sensor's output range and field behaviour to avoid false positives. The margin between the upper and lower thresholds ensures that minor fluctuations do not cause incorrect state changes.

2.3 Ultrasound

2.3.1 Outline of Task & Design Iterations

To determine the duck's name, we first needed to detect the ultrasonic signal given out by the duck, demodulate the detected signal into a binary signal, and decode the binary signal that are sent via UART packets in ASCII.



Figure 9: Outline of stages of name detection

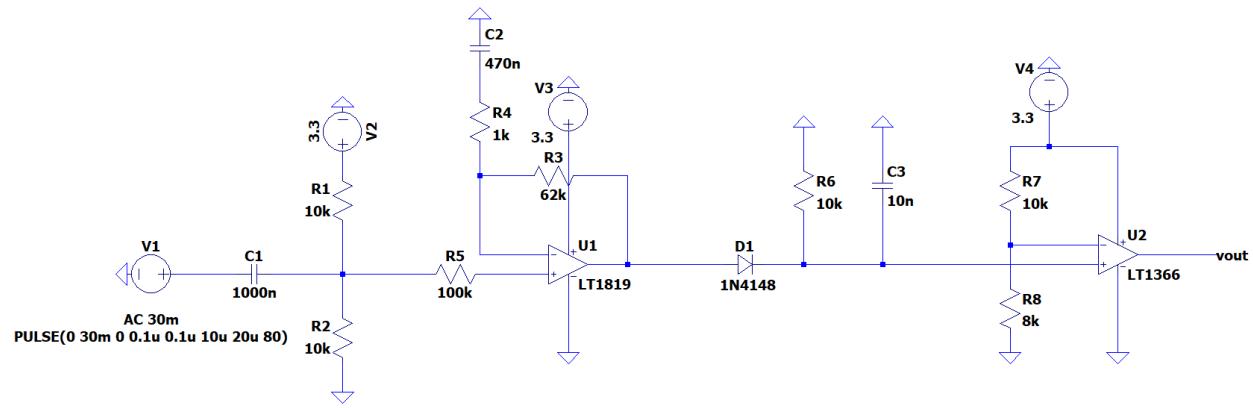


Figure 10: Initial design of circuit

Although the basic structure of the hardware implementation was kept the same, small changes were made to each stage while testing each individually, and also after testing the entire circuit along with the software stage.

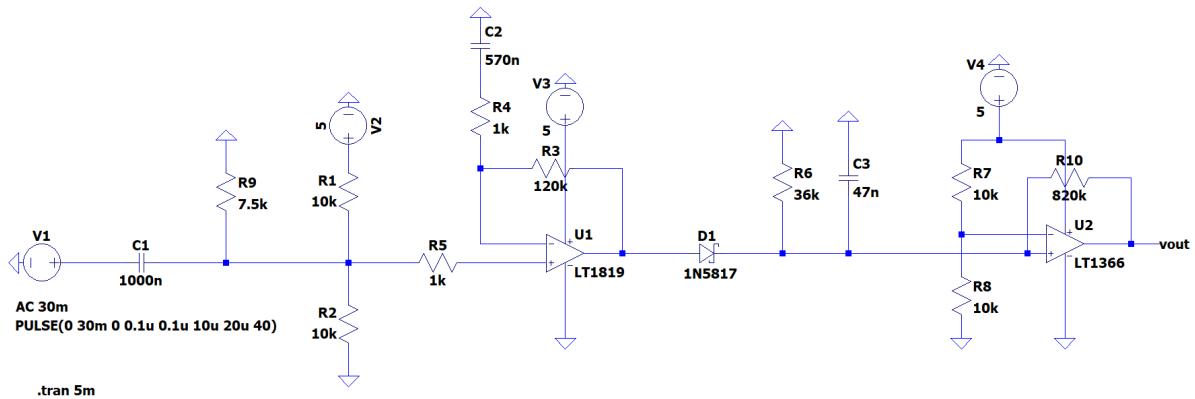


Figure 11: Final design of circuit

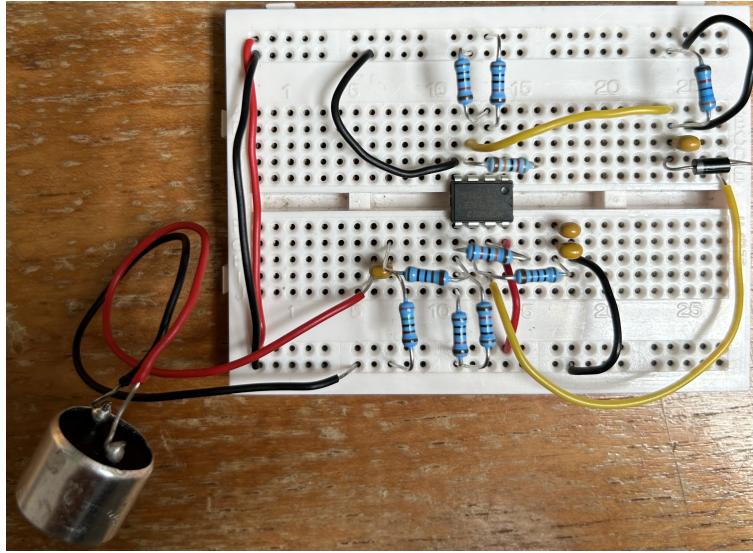


Figure 12: Final circuit design on breadboard

It was also noted that the amplitude of the duck varies significantly based on its rotation relative to the ultrasonic sensor, making it difficult to detect the name from certain angles. During testing, we noticed that there are some positions from which the signal amplitude is very high, and some where it is very low and the name cannot be detected reliably. Hence, the decision was made to use two ultrasonic circuits, with two sensors positioned at different angles on the rover. Only one sensor circuit is needed to detect the name, so using two makes the system more redundant.

2.3.2 Choosing a Sensor

In order to detect the signal, a suitable ultrasonic transducer had to be chosen. The sensor we used is the ProWave 400SR160 [2].

- It has a beam angle of 55° which allows for detection even when the duck and sensor are not aligned perfectly.
- The sensor has a centre frequency of $40 \pm 1\text{kHz}$ which should reduce the effect of noise (mainly from the 50Hz power supply), as frequencies outside this range should be attenuated.

2.3.3 Amplification

The amplitude of the detected signal is quite small and varies based on the rotation of the duck. Hence, amplification is needed to be able to use the envelope detector on the signal, as the forward voltage drop of the diode in the envelope detector is greater than the amplitude of the duck input (around 30mV). A non-inverting op-amp amplifier was used to do this. The input is biased at $0.5V_{CC}$ and the 570nF capacitor is used to preserve the biasing at the output.

For better performance from greater distances and different angles and to improve the quality of the envelope signal, the gain of the amplifier must be quite high. Increasing the value of the feedback resistor to $120\text{k}\Omega$, and reducing the value of the input resistor to $1\text{k}\Omega$ to reduce attenuation of the duck input gives a new gain of around 83 (38.38dB). A supply voltage of 5V was chosen to allow for a larger output amplitude.

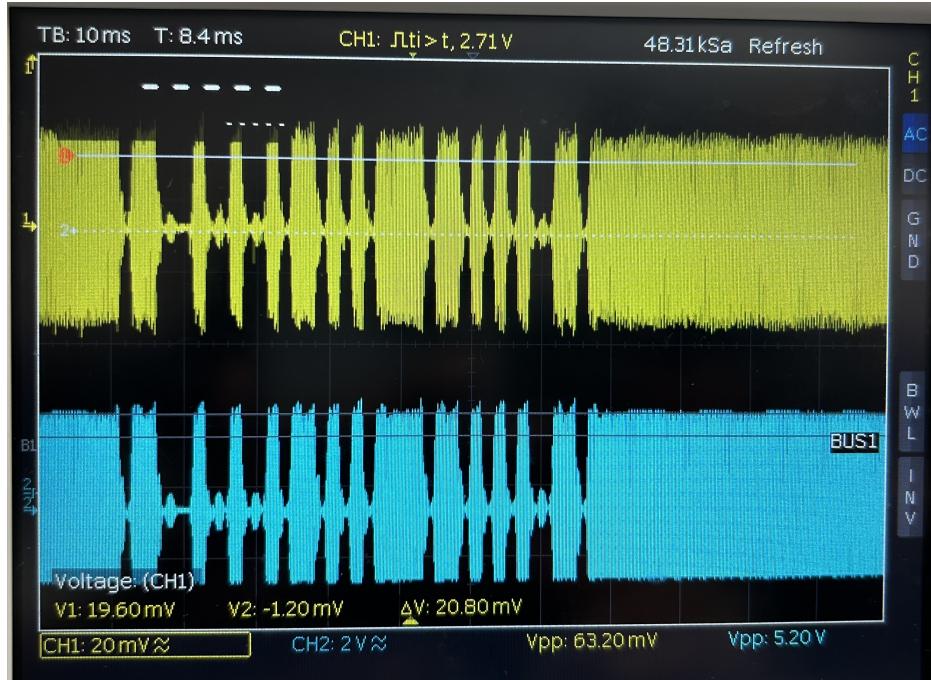


Figure 13: Amplification of input signal

To further reduce low frequency noise, a high pass filter was added to the amplifier input as well, by adding a $7.5\text{k}\Omega$ resistor to ground after the 1nF capacitor. The figure below shows the frequency response of this setup in LTspice; the corner frequency is around 100Hz.

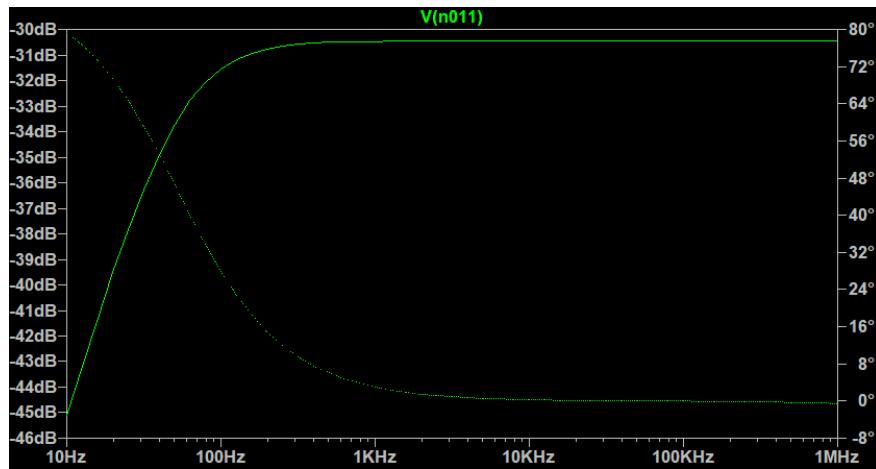


Figure 14: Frequency response of high pass filter

2.3.4 Envelope Detector & Comparator

To convert the amplified and filtered ultrasonic signal received from the duck into a binary signal, it is first demodulated using an envelope detector. Then a comparator is used to turn the demodulated signal into a binary signal that either outputs V_{CC} when the envelope is above a set threshold or 0V when it is below the threshold.

In practice, a Schottky diode was used in the envelope detector because its forward voltage drop is 0.45-0.75V

[5], which is less than a standard silicon diode (0.7V); as a result, it is able to respond faster to changes in the signal and distort the demodulated signal less because of its superior linearity. The time constant τ of the envelope detector is $10k\Omega \times 10nF = 0.1ms$; a smaller time constant means that the envelope detector responds faster to changes in the input.

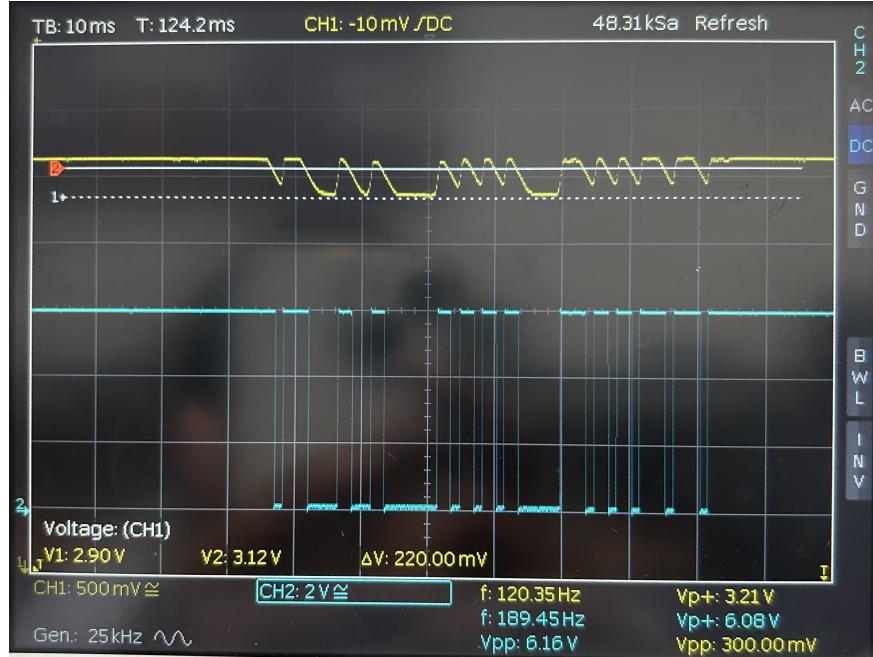


Figure 15: Envelope signal before optimisation

As seen in the figure above, the initial amplitude of the envelope signal was relatively small (200mV), making it difficult to set a suitable bias voltage to trigger the comparator properly; this was rectified by increasing the amplifier gain. After testing different reference voltages, the value determined is $0.5V_{CC}$.

During further testing, we observed that high-frequency noise present in the envelope (after fixing the amplification) resulted in glitching of the rising and falling edges of the comparator output, causing the ESP32 to be unable to process the signal reliably.



Figure 16: Envelope signal with noise

Increasing the values of the resistor and capacitors in the envelope detector filtered out some of this noise and smoothed out the output because it lowers the cutoff frequency of the RC filter, allowing it to reject high-frequency ripple while retaining low-frequency envelope signals. The new τ is $36\text{k}\Omega \times 47\text{nF} = 1.692\text{ms}$. The larger time constant makes the envelope detector less responsive to high-frequency noise, hence smoothing the comparator output.

An $820\text{k}\Omega$ resistor was also added between $V+$ and the output to introduce positive feedback; it adds hysteresis to the output, which eliminates switching due to small changes in the envelope.

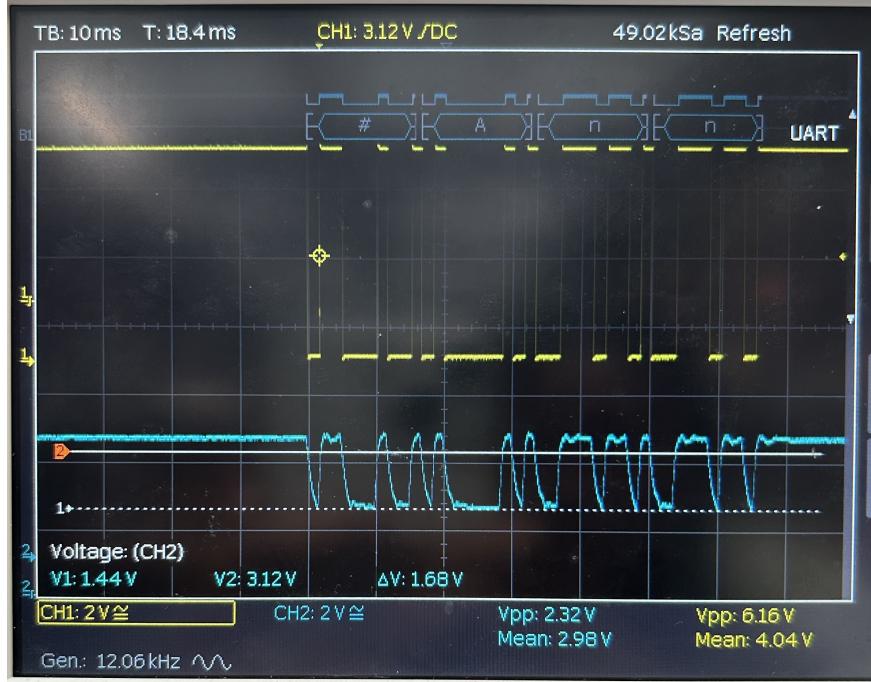


Figure 17: Envelope signal and comparator output from final circuit

2.3.5 Obtaining the Name of the Duck

The circuit produces a binary signal that can be decoded to produce the name of the duck, as described in the section **"Decoding ASCII in UART package for Ultrasound"**.

2.4 Radio

The duck emits an AM radio signal with message frequency $f_m = 100$ Hz or 150 Hz, carrier frequency $f_c = 89$ kHz with modulation index $\mu \leq 1$. The goal is to find f_m to identify the species of the duck. The supply voltage V_{CC} for the circuit is 3.3V.

2.4.1 Antenna

The detection of radio signals is done using a tuned coil antenna, which is an air-core copper wire coil with 118 μ H inductance. A capacitor is chosen to create a tuned circuit with resonant frequency equal to the carrier frequency of 89 kHz.

The antenna is approximately 6cm in diameter which was determined to be optimal for receiving the low frequency radio.

2.4.2 Radio Front End

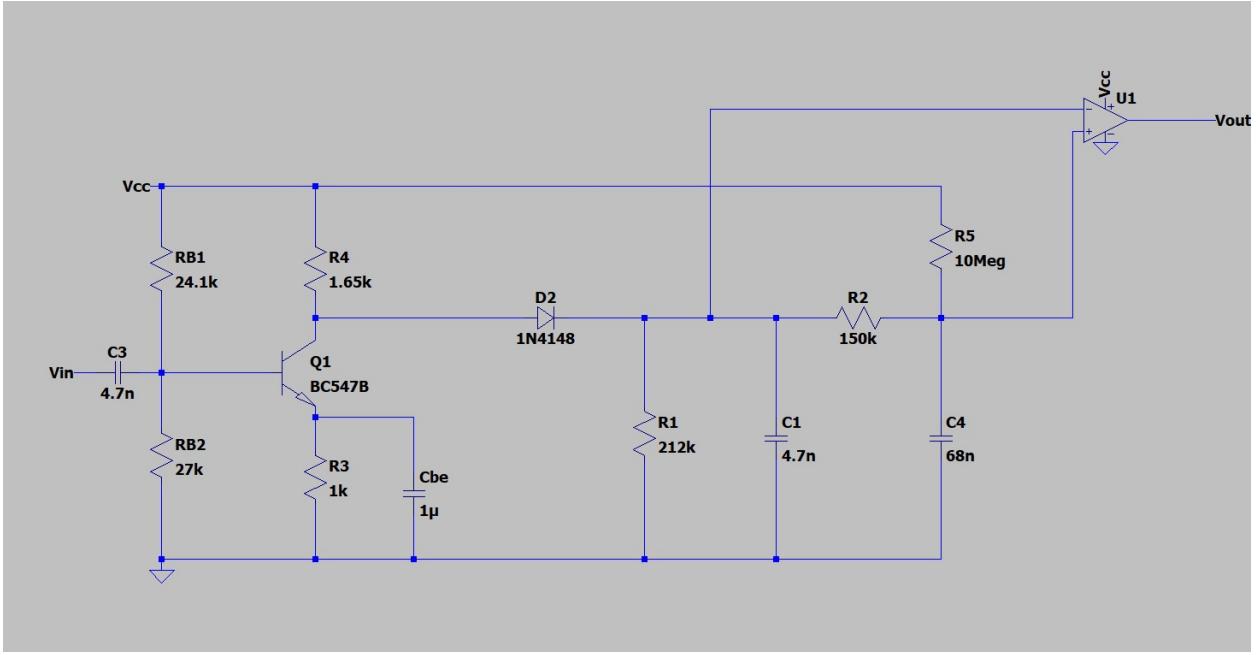


Figure 18: Radio front-end design

2.4.3 Process

The signal produced by the tuned coil is too small to extract, so a common-emitter (CE) amplifier was built to amplify the signal. This method of amplification is used because it is simple to set up and eliminates one op-amp. The output of the CE amplifier is biased at $\frac{V_{CC}}{2}$ with a quiescent collector current of 1 mA, and the gain of the circuit is approximately -70 when tested using an oscilloscope. Emitter degeneration is used to provide bias stability with respect to temperature, and the emitter capacitor restores amplifier gain for high frequencies.

After the amplifier, the AM signal is demodulated in a similar manner to the Ultrasound signal. The envelope detector configured with an RC corner frequency $f_m < f < f_c$ charges and discharge around the edges of the AM waveform, producing the envelope of the signal.

The op-amp functions as a comparator to turn the AM envelope from a sinusoid waveform to pulse wave that be used for the ESP32's digital input. The non-inverting input is connected to an RC low-pass filter with corner frequency $f < f_m$ from the envelope, giving its mean value. The comparison between the envelope and mean value allows RF detection for a range of signal amplitudes. The resistor R_{23} provides a voltage bias to the low-pass filter output to stop the comparator from switching when $V_{in} = 0$.

2.4.4 Software

The frequency measurement code is almost identical for both infrared and radio detection. Hence, it is found in the section “**Sensor Reading**”.

3 Control & Web Design

3.1 Architecture

Two architectures were experimented with: one that leveraged the multicore nature of the ESP32-WROOM-32, and another that used the underlying FreeRTOS system in the ESP32 family for task delegation. We

chose the latter for the final design, as it offered greater extensibility, allowed the ESP32's software to handle core allocation more effectively, and enabled compatibility with a smaller development board that features only a single core for the PCB.

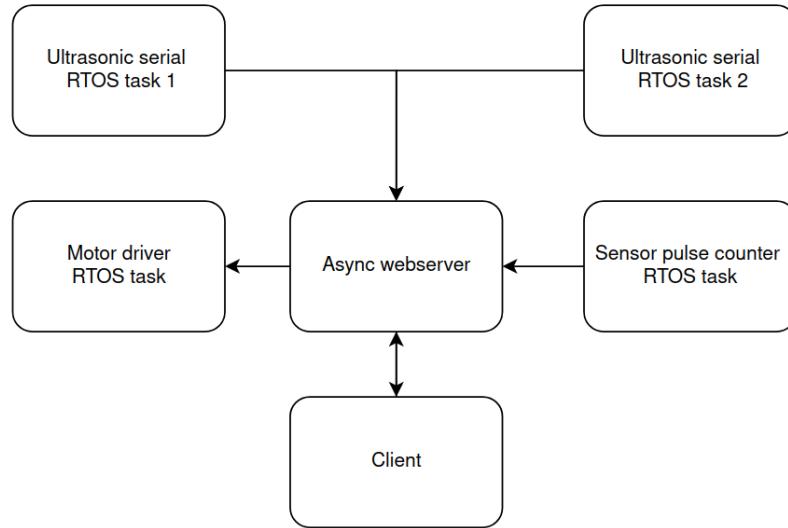


Figure 19: Outline of code structure and handling for RTOS task based implementation

The ESP32 is configured as an access point, allowing users to connect directly via a WiFi capable device. Once connected to this network, the server is accessible at the IP address <http://192.168.4.1/>. The ESP32 serves HTML content over HTTP when users access the root (/) endpoint.

The major advantage of using ESP32 as an access point is the ability to have direct communication between devices and ESP32, without needing a dedicated route, which minimizes latency in signal processing. Another reason that supports our decision for us to transition to the ESP32 platform for our microcontroller.

ESP-32 code

```

1 void setup(){
2   // Other setup code comes before this
3   SPIFFS.begin(true);
4
5   // WIFI
6   WiFi.softAP(ssid, password);
7   Serial.println(WiFi.softAPIP());
8
9   // Async webserver
10  server.on("/", HTTP_GET, handleRoot);
11  server.serveStatic("/assets", SPIFFS, "/assets/");
12  server.on("/joystick", HTTP_GET, handleJoystick);
13  server.on("/read_data", HTTP_GET, handleReadData);
14  server.begin();
15 }
16
  
```

Listing 1: Setup of the Access Point

All our code is loaded on to the ESP code is in our GitHub <https://github.com/h3430/DuckEEE>.

3.2 Sensor reading

The web app sends GET requests displayed on the web app in a table class. As a result of this we have efficient monitoring and debugging without the need for an onboard display; this reduces hardware complexity and power consumption. We use a typical easily parsed format of ASCII through HTML the messages are in this format:

```
"radio=&infrared=&magnetic=&species=&name=bob"
```

with the values coming after the '='.

We gather the values from their respective functions. We chose to have the values calculated and decoded on the ESP as opposed to client-side as it allows us to fully utilise the processing of the esp and reduces the complexity of sending data between the rover and the controller.

3.2.1 Digital Frequency Measurement for Infrared and RF

The frequencies of the infrared signal and demodulated RF signal are determined through software by counting the number of pulses that occur in a fixed gate interval. To implement pulse counting, the hardware pulse counter (PCNT) of the ESP32 was used, as this does not require any overhead from the main processor. The edge counter was configured to increment the pulse count on a rising edge of the input signal. The Infrared and RF frequency measurements are performed simultaneously, hence halving the measurement time compared to a sequential approach.

```
1 // Pulsecounter Setup was created by using https://github.com/pycom/pycom-esp-idf/blob/
2   master/examples/peripherals/pcnt/main/pcnt_example_main.c
3 // to prompt Deepseek R1 Qwen distilled for examples.
4 // Model running on: https://t3.chat/.
5 // Prompt and response are in the appendix.
6 void setup_pcnt()
7 {
8     // IR pulsecounter
9     pcnt_config_t pcnt_config_ir = {
10         .pulse_gpio_num = IR_INPUT_PIN,
11         .ctrl_gpio_num = PCNT_PIN_NOT_USED,
12         .lctrl_mode = PCNT_MODE_KEEP,
13         .hctrl_mode = PCNT_MODE_KEEP,
14         .pos_mode = PCNT_COUNT_INC,
15         .neg_mode = PCNT_COUNT_DIS,
16         .unit = IR_PULSECOUNTER,
17         .channel = PCNT_CHANNEL_0,
18     };
19     pcnt_unit_config(&pcnt_config_ir);
20     pcnt_set_filter_value(IR_PULSECOUNTER, 45);
21     pcnt_filter_enable(IR_PULSECOUNTER);
22
23     // RF pulsecounter
24     pcnt_config_t pcnt_config_rf = {
25         .pulse_gpio_num = RF_INPUT_PIN,
26         .ctrl_gpio_num = PCNT_PIN_NOT_USED,
27         .lctrl_mode = PCNT_MODE_KEEP,
28         .hctrl_mode = PCNT_MODE_KEEP,
29         .pos_mode = PCNT_COUNT_INC,
30         .neg_mode = PCNT_COUNT_DIS,
31         .unit = RF_PULSECOUNTER,
32         .channel = PCNT_CHANNEL_0,
33     };
34     pcnt_unit_config(&pcnt_config_rf);
35     pcnt_set_filter_value(RF_PULSECOUNTER, 85);
36 }
```

Listing 2: Pulsecounter setup code

```
1 // Same as pulse counter initialization, Deepseek R1 Qwen distilled was used to explain
2   and give examples for which functions needed to be called for the pulsecounter.
```

```

2 // Prompt and response is in the appendix
3 vTaskDelay(SENSOR_POLL_TIME / portTICK_PERIOD_MS);
4
5 // Measure frequency
6 pcnt_counter_clear(IR_PULSECOUNTER);
7 pcnt_counter_clear(RF_PULSECOUNTER);
8 pcnt_counter_resume(IR_PULSECOUNTER);
9 pcnt_counter_resume(RF_PULSECOUNTER);
10
11 vTaskDelay(FREQ_MEASURE_TIME / portTICK_PERIOD_MS);
12
13 pcnt_counter_pause(IR_PULSECOUNTER);
14 pcnt_counter_pause(RF_PULSECOUNTER);

```

Listing 3: Code for a single measurement channel

The use of a hardware pulse counter with a gating approach was chosen, instead of an interrupt approach, because this approach reduced CPU overhead and latency. Furthermore, the IR and RF signals for identifying the ducks had relatively large deltas (50Hz for RF, 164Hz for IR). This allowed a lower accuracy approach (gating) to be used for identification.

There is a trade-off between the gate interval and frequency resolution. A shorter gate interval results in a lower delay but also lower frequency resolution due to the reduced pulse count. The current implementation of a 200 millisecond gate interval gives a 5Hz frequency resolution, which is perfectly adequate considering the large difference in signal frequency for each duck species.

To suppress spurious transitions in the comparator output, especially near the duck detection threshold, frequency values are rejected if they deviate from the nominal frequency limits by more than 20%. The table below shows the accepted frequency range for both Radio and Infrared signals. The rejection of false signals is critical in the detection of species since only two of the four species emit infrared, and each species can only emit RF or Infrared. The presence of both Radio and Infrared signals will lead to a conflict.

Signal Type	Nominal Frequency Range (Hz)	Accepted Frequency Range (Hz)
Radio	100 - 150	83 - 180
Infrared	293 - 457	235 - 548

Table 1: Nominal and accepted frequency range for Radio and Infrared signals

3.2.2 Digital Hall Effect Measurement for Magnetic field

The analogue voltage output from the Hall effect sensor is connected to an ADC pin on the ESP32. The ESP32 12 bit Analogue-to-Digital Converter scales the input voltage range (0–3.3 V) to discrete values between 0 and 4095, where 0 corresponds to 0V and 4095 represents 3.3V. This digital conversion allows us to quantify the sensor's response to magnetic fields precisely.

Real-time ADC values are classified into three states using the following thresholds:

- **Up:** ADC value > 3100
- **Down:** ADC value < 2730
- **N/A:** $2730 \leq \text{ADC value} \leq 3100$ (undetermined state)

```

1 String read_hall_value()
2 {
3     int hall_raw = analogRead(HALL_SENSOR_PIN);
4     String hall_state;
5     if (hall_raw >= UPPER_THRESHOLD)
6     {
7         hall_state = "up";
8     }
9     else if (hall_raw <= LOWER_THRESHOLD)
10    {
11        hall_state = "down";
12    }
13    else
14    {
15        hall_state = "N/A";
16    }
17    return hall_state;
18 }
```

Listing 4: Reading hall sensor value and setting state

3.2.3 Decoding ASCII in UART package for Ultrasound

A key component of our system involved decoding UART packets transmitted via ultrasound, which carried each duck's unique identifier in the form of a 4-character name. These packets were transmitted using the UART protocol and received by the ESP32 for processing. We chose to use the built-in UART receiver hardware on the ESP32, due to its efficiency and negligible overhead.

To extract usable data, the incoming bytes were first read in hexadecimal format and then translated into ASCII characters to recover the original name. Each duck name followed a consistent structure: It always began with the character # and terminated with a null character (0x00). This structure enabled reliable parsing, since the system could begin decoding only after detecting the # character, and stop once 4 characters have been recognised as we know by definition within the task, the maximum length for a duck's name is 4 including the initial # character)

By scanning the UART stream for the starting # symbol and reading the characters sequentially until reaching the fourth character, we were able to reconstruct the duck's name in real time.

The decoded ASCII can then be sent to the app through HTTP.

```

1 // Heavily referenced example by UKHeliBob: https://forum.arduino.cc/t/receive-string-from-
   serial-monitor/643266/2
2 void uart_reading_task1(void *parameter)
3 {
4     String name = "";
5
6     while (1)
7     {
8         if (Serial.available())
9         {
10            char c = Serial.read();
11            // wait for # and start reading next 3 chars
12            if (c == '#')
13            {
14                name = "#";
15            }
16            else if (name.length() > 0)
17            {
18                name += c;
19                if (name.length() == 4)
20                {
21                    ultrasonic_name1 = name;
22                    name = "";
23                }
24            }
25        }
26    }
27 }
```

```

25     }
26     // delay to not oversaturate task
27     vTaskDelay(UART_READ_DELAY_MS / portTICK_PERIOD_MS);
28 }
29 }
```

Listing 5: Decode ASCII in UART packets code

3.3 Motor control

For motor control, two methods were considered. First, a virtual joystick and second, using the W, A, S, D keys on the keyboard.

The joystick would allow for extremely precise movement over the rover, which allows us to manoeuvre the duck into a position where we receive the greatest signal strength from the ducks.

Keyboard control would sacrifice some precision, but it would allow team members to move the robot around the arena in a more intuitive manner, as they are familiar with these control mechanics in video games. Furthermore, this control mechanics allows for quicker changes in direction compared to the joystick. We mitigated the accuracy cons with keyboard control by adding a low- and high- speed mode to the controls.

During testing in the arena, we found that using the joystick was cumbersome for very rapid movements. Therefore, we decided to implement both. The keyboard controls will be much more effective for traversing the arena, getting between ducks. The virtual joystick will allow us to do extremely fine movements and position the rover into the exact orientation we need.

The joystick and keyboard controllers stream their x- and y-positions through HTTP requests to the ESP. There is also a table included where the read data from the sensors is stored for us to see and assess the relevant factors of the ducks. The data is received from responses to GET requests.

```

1 void handleMotors(AsyncWebServerRequest *req)
2 {
3     motor_x_val = req->getParam("x")->value().toInt();
4     motor_y_val = req->getParam("y")->value().toInt();
5     req->send(200, "text/plain", "command received");
6 }
7
8 void motorTask(void *parameter)
9 {
10    while (1)
11    {
12        if (motor_x_val == 0 && motor_y_val == 0)
13        {
14            // Stopped
15            ledcWrite(0, 0);
16            ledcWrite(1, 0);
17            ledcWrite(2, 0);
18            ledcWrite(3, 0);
19        }
20        else
21        {
22            int left_val = motor_y_val + motor_x_val;
23            int right_val = motor_y_val - motor_x_val;
24            int left_speed = min(abs(left_val), 255);
25            int right_speed = min(abs(right_val), 255);
26
27            if (right_val >= 0)
28            {
29                ledcWrite(1, right_speed);
30                ledcWrite(0, 0);
31            }
32            else
33            {
34                ledcWrite(0, right_speed);
35                ledcWrite(1, 0);
36            }
37        }
38    }
39 }
```

```

36 }
37     if (left_val >= 0)
38     {
39         ledcWrite(2, left_speed);
40         ledcWrite(3, 0);
41     }
42     else
43     {
44         ledcWrite(3, left_speed);
45         ledcWrite(2, 0);
46     }
47 }
48 // Small delay to not oversaturate task
49 vTaskDelay(MOTOR_DELAY / portTICK_PERIOD_MS);
50 }
51 }
```

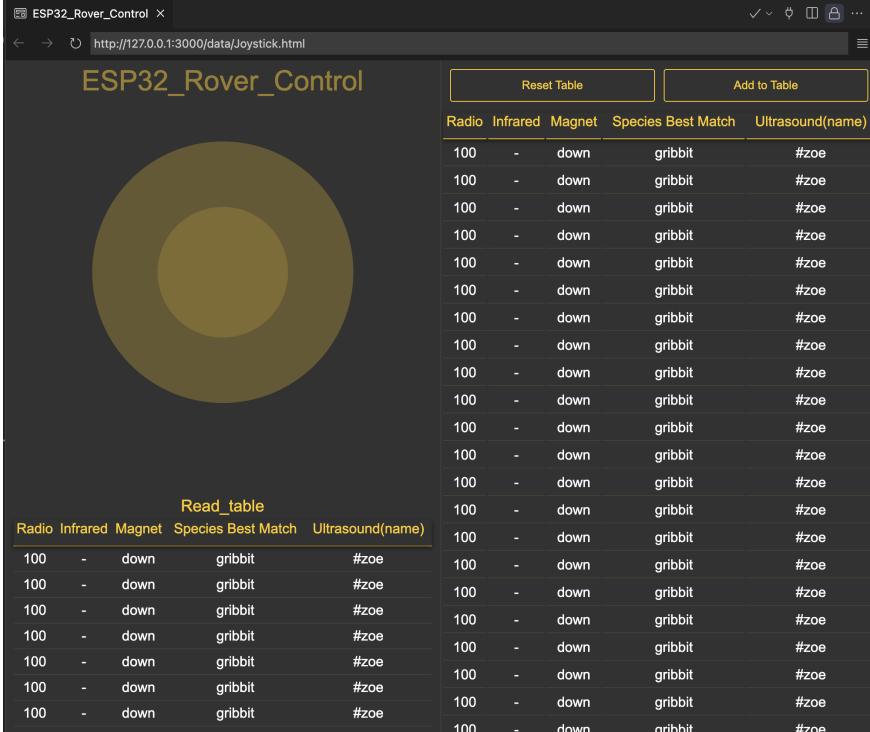
Listing 6: Control of the Motors

This is the algorithm that decodes the information from the web-server and then sends the relevant logic to drive the motors.

One drawback from using the joystick is the large amount of HTTP requests required, this uses a large bandwidth of both the WiFi and the ESP32 processing. This should not be an issue if we ensure we do not progress to a stage where requests are ignored; this will not be an issue. The keyboard-based approach does not have this problem to such a high extent.

3.4 Web page

Web page is an HTML app with a dedicated styling file to ensure that control and sensor handling is intuitive and readable.



The screenshot shows a web browser window titled "ESP32_Rover_Control". The URL bar indicates the page is at <http://127.0.0.1:3000/data/Joystick.html>. The main content area features a large circular graphic with concentric rings. Below it is a table with two sections: "Read_table" and "Radio Infrared Magnet Species Best Match Ultrasound(name)". The "Read_table" section contains 20 rows of data. The "Radio Infrared Magnet Species Best Match Ultrasound(name)" section contains 10 rows of data. Both tables have columns for Radio, Infrared, Magnet, Species, Best Match, and Ultrasound(name). The data in the tables is mostly identical, showing values like 100, -, down, gribbit, #zoe, and #zoe.

Radio	Infrared	Magnet	Species	Best Match	Ultrasound(name)
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe

Radio	Infrared	Magnet	Species	Best Match	Ultrasound(name)
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe
100	-	down	gribbit	#zoe	#zoe

Figure 20: Web page interface

The image above of the web interface shows the joystick on the left and a table that adds rows containing data from sensors. Sensor data is read in intervals of 300ms; this is stored on the table on the right, allowing us to go back and review the data we received even after driving away from the duck. Data included in the table is just placeholder to show how it is stored and displayed.

The "Add to Table" button adds the current data into the table on the left, which allows us to split actual reads, for example, when we are near the duck and the data from when we are driving between the ducks and expect no good data.

We chose this approach of displaying the data so that all information gathered by the sensors will be stored; this allows any bad reads of one sensor to have little effect on our ability to decode the species and name of the ducks. For example, if on one read we get a good infrared reading but a bad radio, and vice versa on the next reading, we will still have gathered and stored all the information we need to make a fair conclusion of the species of duck.

```

1 function fetchLiveData() {
2     fetch('/read_data')
3         .then(response => response.text())
4         .then(queryString => {
5             const params = new URLSearchParams(queryString);
6             latestData = {
7                 radio: params.get('radio'),
8                 infared: params.get('infared'),
9                 magnetic: params.get('magnetic'),
10                species: params.get('species'),
11                name: params.get('name')
12            };
13            updateLiveRow(latestData);
14        })
15        .catch(error => {
16            console.error('Error fetching live data:', error);
17        });
18    }

```

Listing 7: Collecting Data from HTTP

Through our effort to minimise the time taken for the sensors to collect the readings, this can update many times a second, giving a very convincing live response from the sensor. We found this to be a crucial goal as it ensures that we are gathering plenty of information about a duck. We therefore do not allow any chance for inability to gather data from a duck.

4 PCB

4.1 Considerations

- Weight
- Reliability
- System integration

The decision to create a dedicated PCB implementation that would integrate all of the sub-teams' sensors was made due to the requirement of weight reduction and the need for a reliable and rugged system. Breadboards were used for prototypes, but the team quickly found that these implementations are unreliable, and are prone to intermittent and unreliable connections when under mechanical forces, which is a more significant concern as the designs increase in complexity. The team also considered implementing the system on veroboard, but a custom PCB implementation was chosen as it would be more robust and compact.

4.2 Schematic

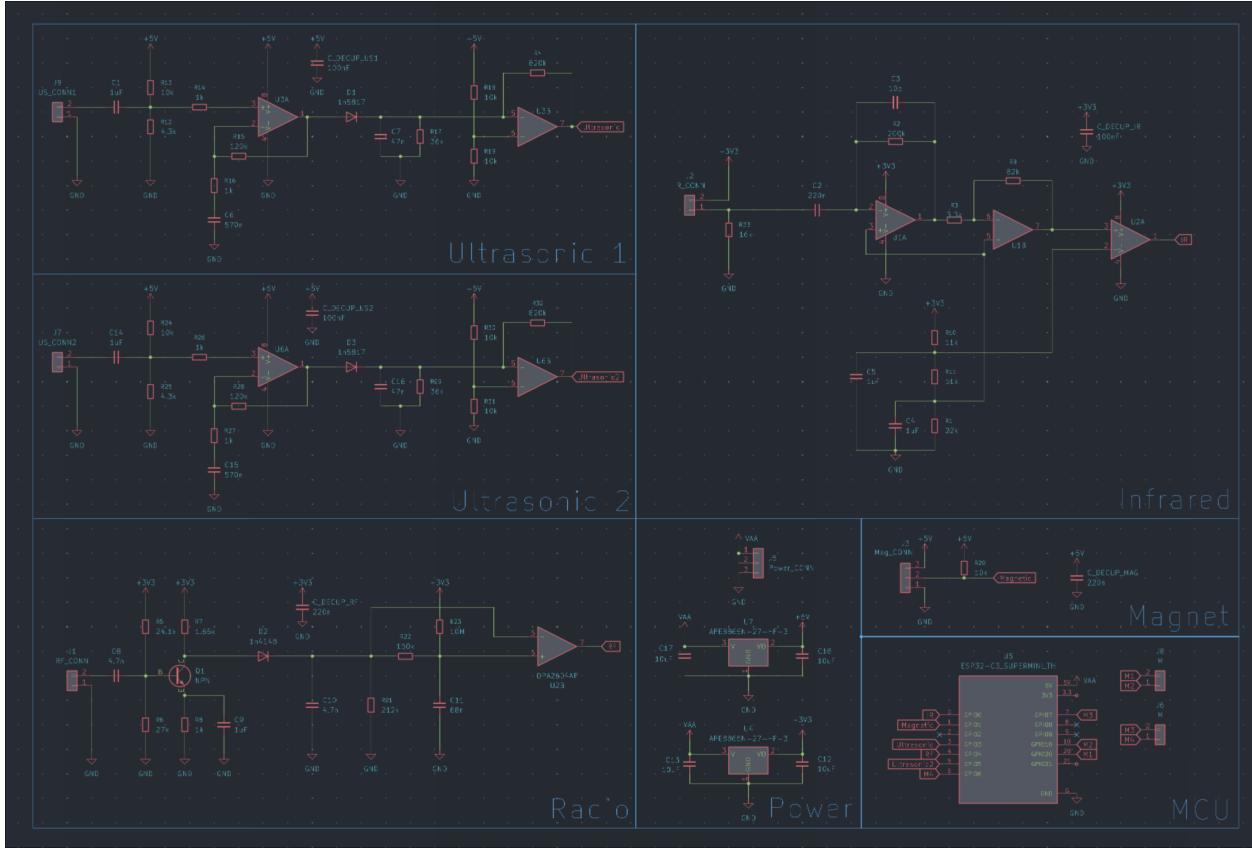


Figure 21: Schematic

The schematic integrates the LT spice schematics of each subteams' working sensor boards into one.

4.3 Implementation

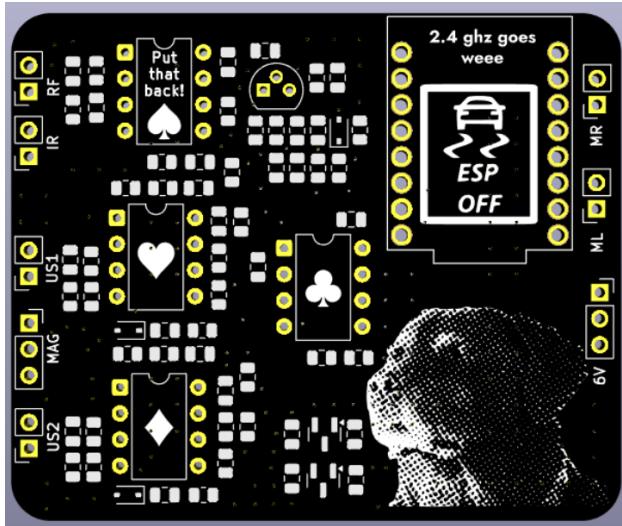


Figure 22: Front side of PCB

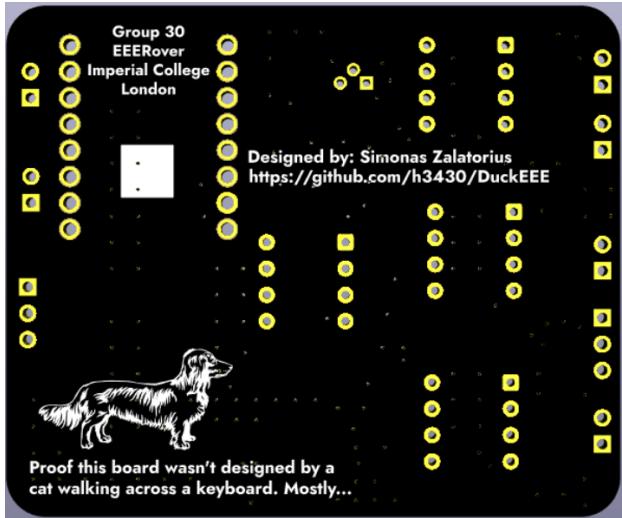


Figure 23: Back side of PCB

Intrinsically, a PCB can facilitate complex designs more reliably than a veroboard or breadboard, as it allows us to make connections arbitrarily and for those connections to be reliable.

For the PCB itself, two main goals were set: First, it had to be as close to 6cm x 6cm to fit on the rover chassis and to allow for much more flexibility; such as adding a veroboard circuit if we found any part of the PCB to be faulty beyond repair or if we find a more optimal solution to a sensor circuit. Second, the PCB must be reliable under reasonable mechanical stressors (touching the board, laying it on a flat surface, etc.). This was accomplished by using surface mount components, as these have a far lower chance of being disturbed by mechanical forces, as they lie very close to the PCB body itself, and the existing headers are taller than them. Furthermore, we made the decision to use sockets for our op-amps, as this would allow us to quickly swap out a failed component. The size goal was met by using surface mount 0805 components and an ESP32 microcontroller (ESP32-C3 SuperMini). Reliability for sensor connections is solved by soldering the sensor wires directly to the PCB to form strong connections.

The main problem that was faced during the design of the PCB was the size of the development boards that could be used for it. We experimented with the idea of a fully custom ESP32 implementation but decided against this, as it would introduce a point of failure for the design. We settled on the ESP32-C3 SuperMini, as it was the smallest commercially available development board that we could find with a low lead time.

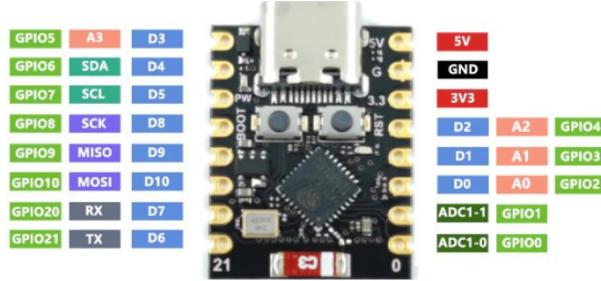


Figure 24: Devboard

We also faced issues with PCB manufacturers. Initially, JLCPCB was chosen as our manufacturer, due to their quick lead times, low prices, and familiarity. Unfortunately, the department was having issues with JLC, therefore we went with Euro Circuits after consulting with lab staff. At the time of submitting the report, we have not yet received our board from Eurocircuits, but we have communicated with their representative responsible for Imperial College London, and he has said that they were making every effort possible to ship out the boards on Wednesday, and that we are highly likely to have the boards in our possession before the end of Friday (Before the demo).

Due to the mentioned delay, we have created a working prototype of our rover, with the sensors on breadboards. We did this to test the code in a fully built and integrated system and, more importantly, to have a contingency plan in case the PCB does not arrive before our demonstration time slot.

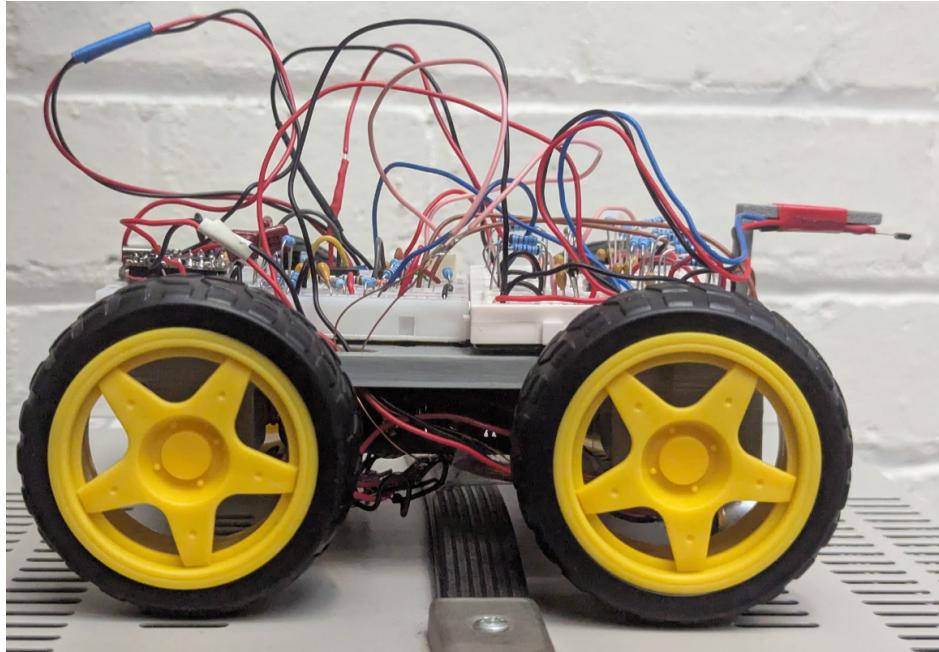


Figure 25: Final Rover prototype

5 Body

5.1 Drivetrain

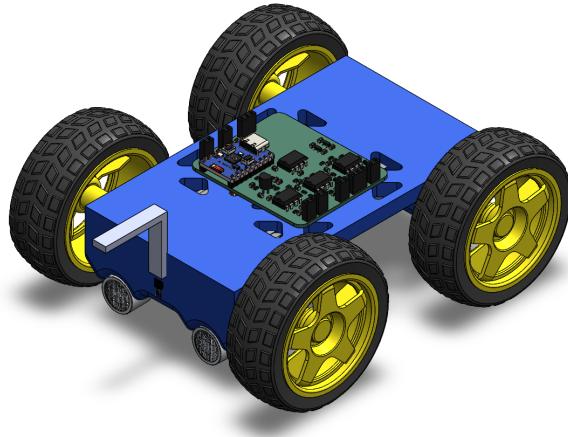


Figure 26: CAD model of rover

Our goals with the drivetrain were:

- rapid rotation (360 degree rotation in 1 second)
- rapid acceleration
- high top speed (at least 50% faster than the EEEbug)
- ability to rotate and drive at a low velocity to precise positioning next to a duck.

These goals were chosen because our team hypothesised that most of our time would be spent traversing the arena in search of ducks, rather than performing the detection itself. Additionally, we set the goal of slow and precise movement because our magnetic sensor's range was very short; therefore, we would need to drive very close to the duck and properly position the sensor.

To meet these objectives, we chose a nearly square drivetrain using four N20 motors. Using four motors instead of two allowed the rover's axis of rotation to be closer to its centre, resulting in more intuitive control. Moreover, the increased torque of the four motors allowed slower and more controlled movements, translating to greater precision and accurate movement.

In determining the drivetrain dimensions, we needed to balance turning agility and straight-line stability. After multiple iterations, we found that a 1.05:1 wheelbase offers the ideal compromise for us. Our initial design used a 1.2:1 wheelbase, but this resulted in too much drag while turning. Reducing the length to a 0.8:1 wheelbase improved turning but significantly compromised straight-line stability, causing the rover to drive at a random initial angle when starting to move forward. The 1.05:1 configuration provided the best balance. Furthermore, drivetrain dimensions were influenced by antenna size requirements because the baseband signal operated at low frequency, the antenna needed a sufficient diameter to maintain adequate communication range. This constraint helped determine the width of the wheelbase, which in turn defined its vertical length.

The shape and material were also carefully considered. We initially used 3D-printed PLA wheels with flexible filament tyres, but they lacked sufficient grip and caused the rover to skid during acceleration or deceleration. We then explored the use of mecanum wheels, which would have allowed omnidirectional movement. However, this proved impractical: commercial wheels exceeded our budget, and 3D printing them posed significant challenges due to the tight tolerances required and the poor traction offered by PLA plastic. As a compromise, we adopted 65mm Adafruit wheels with rubber tyres. These offered a practical balance between availability, size and grip.

Ultimately, we chose a drivetrain design using four N20 motors with 65mm wheels, supported by a 100mm by 95mm base. This setup delivered the optimal combination of straight-line stability and responsive turning. Additionally, the use of four motors allowed the implementation of advanced turning techniques such as torque vectoring, further enhancing control and performance.

5.2 Body Modules

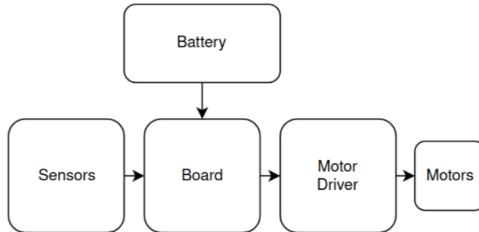


Figure 27: Body module flowchart

5.2.1 Motors



Figure 28: Motor

The motor choice was guided by our drivetrain performance objectives. After searching through the list of approved suppliers, we selected brushed N20 motors, as they have a well-balanced combination of torque, velocity, weight, and ease of control. Specifically, we chose the 500 RPM, 6V variant of the N20 motor, as one of our team members already had several available and they aligned with our objectives. Additionally, we found that we could drive these motors at 7.5 volts, which would further increase the straight-line velocity. Although this voltage exceeds the motor's nominal rating and may reduce the lifetime of the motor, the trade-off was acceptable for our use case since we are only running the motor for short periods (tens of minutes) and not under significant load. We also verified that the current consumption of each motor does not exceed 200mA, which is well within the rated amount for the batteries. Furthermore, during our testing, we did not find any thermal problems with either the controller or the motors.

5.2.2 Motor Driver

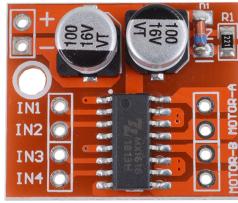


Figure 29: TC1508A

The motor driver used: TC1508a

Motor specifications[4]:

Input Voltage: 5V to 35V.

Output Current: Up to 1.5A per channel.

Logic Voltage: 1.8-5V.

Peak current: 3.5A per channel.

Low standby current: less than 0.1uA

Drive Mode: Dual H-bridge driver.

Protection: Built-in flyback diodes for motor protection against reverse voltage spikes.

Our main objectives with the motor driver were that it needs to be lightweight, at least 800mA per channel, up to 6V input voltage, and a logical voltage range that would include 3.3V. Conveniently, one of our team members had this motor driver on hand and its specifications are perfect for this application, as we do not exceed either the current or input voltage ratings. The ESP outputs a 3v3 PWM signal at 5000Hz and 8 bit resolution, which worked great with this driver. Furthermore, it was able to handle the stall and startup currents of our N20 motors without issue.



Figure 30: Motor driver on the rover

Initially, we used four modified servo motor drivers, where the included potentiometer was replaced with two resistors. Although this setup functioned reasonably well at first, we soon discovered that these drivers were not rated for the current our motors required, causing one motor driver to fail. Furthermore, due to the tolerances of the resistors, many modifications had to be made in the code to synchronise the RPM of each motor.

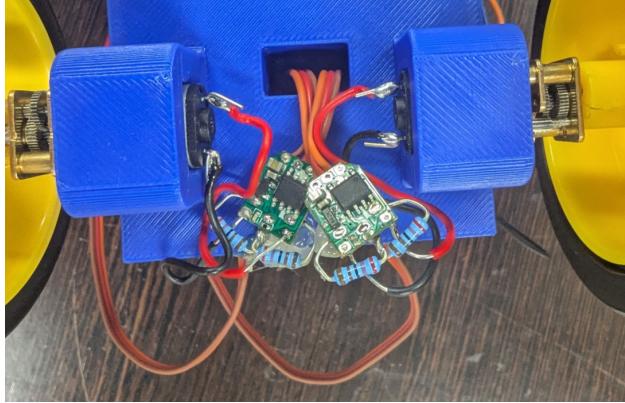


Figure 31: Modified servo driver

5.2.3 Power

Power is managed as shown in the flowchart:

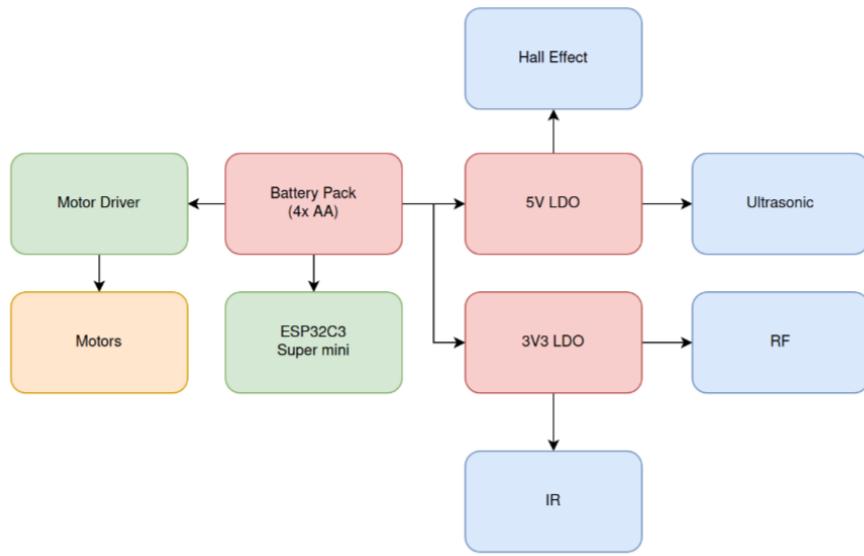


Figure 32: Power flowchart

Our main objective with the battery was to be relatively lightweight and have adequate voltage and current (at least 6V and 1.3A). Our initial idea was to use readily available 18650 lithium ion cells, but these are banned in the EEE department. Therefore, the main supply is 4x AA batteries. AA battery cells were chosen because they are capable of supplying the motors with their rated voltage and amperage. The voltage output from these cells is also fed to the ESP32-C3 SuperMini, which has an internal LDO that then feeds the module itself. The sensors are fed by 2x LDO's. One for 5V and another for 3V3. This was done to provide the sensors and their circuitry with noise-free and stable voltage rails. Furthermore, each sensor and op-amp which uses the power rails has its own decoupling capacitor to provide a power source in the case of transients.

5.3 Chassis

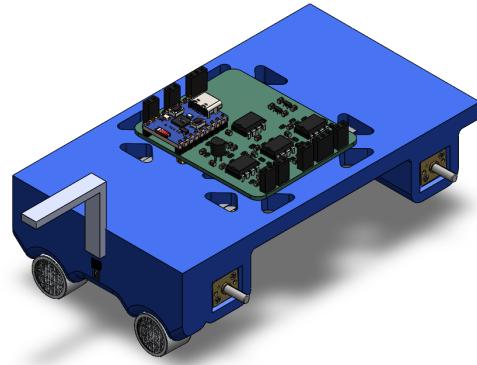


Figure 33: CAD model of rover body

The main objectives of the chassis were weight and adherence to the maximum dimensions for the challenge, with a secondary goal of robust sensor and component mounting. To achieve these goals, we chose to 3D print the chassis using PLA filament. We found that after multiple mass optimisations, we were able to get the chassis down to 62 grams, which fulfils our goal. Furthermore, we created cutouts in the middle section of the chassis to further reduce total mass and make routing wires easier. Through strategic planning of the chassis size and wheel selection, we met the dimension goal, as the rover is 87mm tall, 123mm long and 108mm wide, which are well below the maximum dimension requirements. Furthermore, we were also able to meet and exceed the total rover weight goal, with an estimated weight of around 450 grams from the sensor near the pond.

5.3.1 Sensor placement

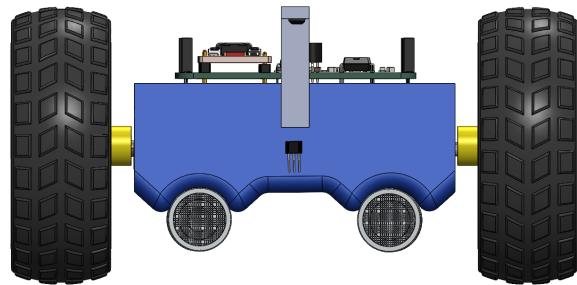


Figure 34: CAD model of sensor placement

All sensors are mounted on the front of the body, as we will aim this at the duck when detecting it. The magnetic sensor is mounted on a boom arm that will be above the duck's head. There are 2 ultrasound sensors, both converging to get a more accurate reading. Furthermore, the coil is mounted on the bottom of the robot, as the duck has a planar PCB antenna. Theoretically and practically, this configuration offers the best input power at the antenna, therefore, it was chosen. We were afraid of motor noise coupling into the antenna, but testing showed that this was not an issue.

Before deciding on the optimal sensor locations, each team experimented with rotating the duck and changing sensor heights to find the ideal configuration for sensor placement.

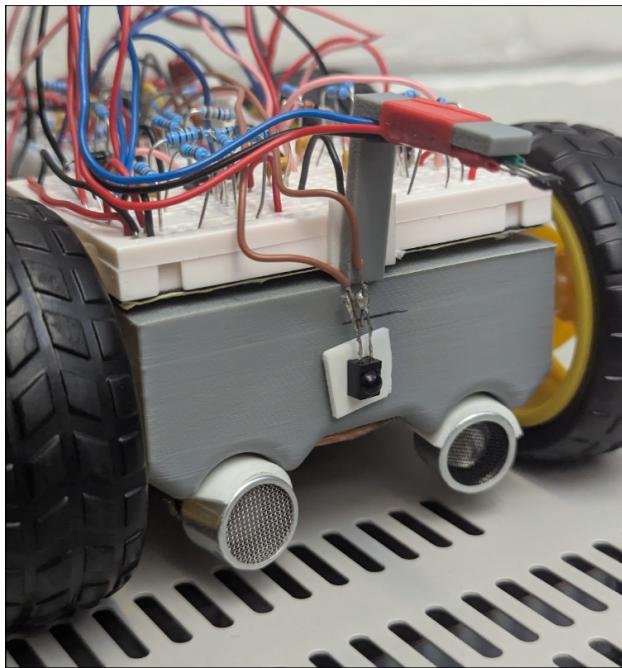


Figure 35: Sensor placement on the front

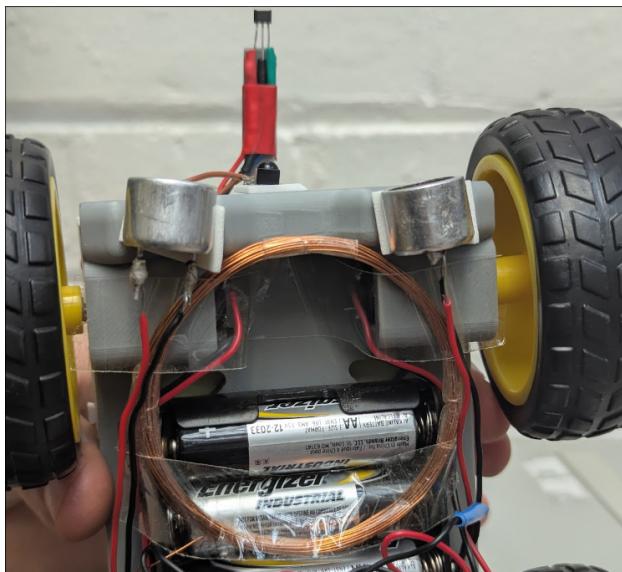


Figure 36: Sensor placement on the bottom

IR: The IR subteam found that the optimal location for the IR sensor would be head on to the duck. As we cannot predict how we will approach each duck, we decided that the best location to integrate the sensor would be in the very middle of the frontal area of the robot, as this location would be the most intuitive for the driver to aim towards the duck.

RF: The RF subteam found that the signal was much less dependent on the rotation and height around the duck than the other teams, therefore the most convenient position was chosen, which still provided strong signal reception. This was below the main chassis and behind the ultrasonic sensors.

Ultrasonic: The ultrasonic team found that the ultrasonic sensors were the most sensitive to the orientation of the duck and to sensor height. The reception of the signal is very poor if the sensor is greater than 1cm off of the base of the duck, so the sensor is placed relatively low on the rover. Each ultrasonic sensor is angled slightly inward for the highest probability of at least one of them being orientated in a favourable direction when approaching the duck.

Magnetic: The magnetic team found that this sensor was the most sensitive to distance. Therefore, after consulting with the body sub-team, the decision was made to mount it on a boom arm and drive it right above the duck's head.

6 Further Development

6.1 Ultrasound

Determining the ideal reference voltage for the comparator stage was quite challenging. The reference had to be carefully tuned to trigger correctly and consistently. One potential solution is to derive the average value of the signal and use it as the comparator reference. This implementation would be more robust, especially in cases where the envelope amplitude varies significantly, a behaviour that we observed while developing the circuit. Moreover, using the average as a reference would help avoid false triggers caused by noise spikes in the signal.

An alternative approach is to implement the comparator in software. The ESP32 reads the envelope signal using `analogRead()` and applies a trigger based either on a fixed threshold or the computed average value of the signal. Hysteresis could also be added to the software to further improve the quality and stability of the binary signal. This software-based approach would reduce the number of hardware stages required, as only one op-amp stage would be required per ultrasonic circuit, decreasing PCB size and overall cost. As part of our future work, we could test both hardware and software comparator implementations to determine which performs the best in the system.

Additionally, to ensure accurate name recognition, a list of all possible names could be obtained, which the detected name could be cross-checked against before sending to the web app.

6.2 Web app

With further development, it would be extremely beneficial to have a live update controlled by updates from the sensors to the ESP. This development would include further reducing the required time to read from the sensors and having a higher rate of HTTP from the ESP.

Another feature that would be advantageous to explore would be a physical joystick control. This would further improve our ability to apply precise control to the rover and would make the controls even more intuitive as we would have the physical feedback.

6.3 PCB

Miniaturising the PCB proved to be a challenging task, which requires careful compromises between component size, cost, and availability. The development board and operational amplifiers were the biggest hurdles when it came to size optimisations. By replacing the current development board with a custom implementation of an ESP32-C3, the operational amplifiers could be replaced with their SMT variants to further reduce size. The passive components could also be replaced with 0603's or 0402's instead of the existing 0805's to further reduce size. These changes would also make the PCB fully compatible with standard pick-and-place assembly services offered by many PCB manufacturers, enabling faster and rapid deployment of further production variants.

6.4 Chassis

Rapid and precise manoeuvring was the largest challenge encountered when creating the chassis. While our current design strikes a balance between movement speed, manoeuvrability and weight, all three could be vastly improved. For manoeuvrability, mecanum wheels and a holonomic drive algorithm could be used to arbitrarily position the rover in 2D space, which would allow the driver to position the rover around a duck with extreme precision. Furthermore, lower torque motors with a higher peak RPM could be used to further increase straight line velocity. To allow for these motors to be used, further mass optimisations should be made; more specifically, replacing the AA battery pack with a lithium ion or lithium polymer cell. These cells are around twice as energy dense, therefore a battery that is significantly lighter could be used to power the rover.

7 Conclusion

Our rover project successfully met its core objectives: designing a compact, lightweight, and manoeuvrable “amphibious” rover capable of reliably identifying the names and species of ducks using infrared, magnetic field, ultrasound, and radio sensing techniques. Through a combination of careful system integration, iterative circuit and chassis design, and efficient task management, the final prototype was able to navigate the test environment and be able to extract information from the ducks.

Each sensor circuit was optimised for accurate name and species detection, and for reliability in the demo environment. The hardware designs were continually improved to perform well, even when the rover may not be in the optimal position for signal detection.

The integration of both analogue and digital signal processing, such as frequency counting and UART decoding, allowed the detection of signals despite the limitations of ambient interference and hardware constraints. Sensor placement was optimised through extensive testing, and all major components were consolidated onto the PCB, further improving reliability and reducing overall system complexity.

From a software perspective, using FreeRTOS on ESP32 platform allowed multitasking between motor control, sensor data processing, and user interface updates on the web page. The intuitive control system featured both keyboard and joystick functions, which enabled precise handling of the rover and simplified the process of manoeuvring and identification of ducks. The live table response allowed us to gather large amounts of data, reducing the possibility of failing to properly identify a duck.

This project demonstrated the value of cross-functional collaboration and rapid prototyping to meet both the technical and environmental constraints set by the project brief. While future developments may further improve sensing accuracy and interface responsiveness, the current design establishes reliable hardware and an intuitive web interface that performs well in the test environment and satisfies all the project requirements.

References

- [1] *Circuit Description of the IR Receiver Modules*. Tech. rep. 80069. Retrieved online from <https://www.vishay.com/docs/80069/circuit.pdf>. Vishay Intertechnology, Feb. 2013.
- [2] *Datasheet - 400SR160*. Retrieved online from <https://docs.rs-online.com/18d6/A700000012634220.pdf>. Midas Displays Ltd.
- [3] *IR Receiver for Data Communication [U2538B]*. Retrieved online from <https://www.mouser.com/datasheet/2/36/doc4717-355647.pdf>. Atmel Corporation. Sept. 2005.
- [4] *Motor driver TC1508*. Retrieved online from https://synacorp.my/v3/en/index.php?controller=attachment&id_attachment=777. Synacorp Technologies.
- [5] *Schottky Barrier Plastic Rectifier*. 88525. Retrieved online from <https://www.vishay.com/docs/88525/1n5817.pdf>. Vishay Intertechnology. July 2020.
- [6] *Si Photodiodes*. Tech. rep. KSPD9001E03. Retrieved online from https://www.hamamatsu.com/content/dam/hamamatsu-photonics/sites/documents/99_SALES_LIBRARY/ssd/si_pd_kspd9001e.pdf. Hamamatsu Photonics, Oct. 2023.
- [7] *Silicon PIN Photodiode [BPV22F]*. 81508. Retrieved online from <https://www.vishay.com/docs/81508/bpv22f.pdf>. Vishay Intertechnology. Aug. 2011.
- [8] *Through Hole Hall Effect Sensor [A1324LUA-T]*. Retrieved online from <https://docs.rs-online.com/e64f/0900766b8106a158.pdf>. Allegro Microsystems. June 2022.

8 Appendix

8.1 GitHub Repository

<https://github.com/h3430/DuckEEE>

8.2 Full Size PCB Layout Pictures

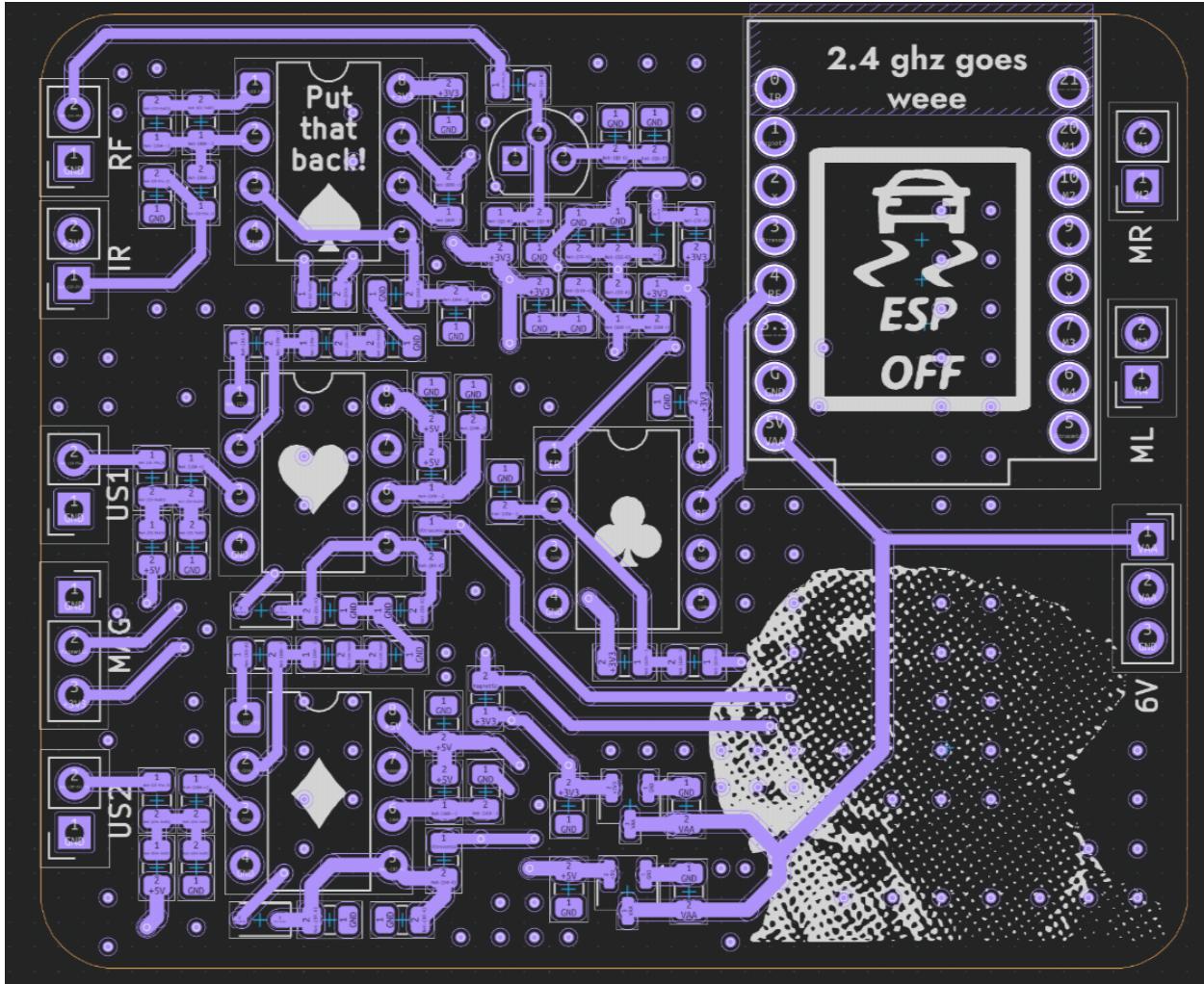


Figure 37: PCB front side

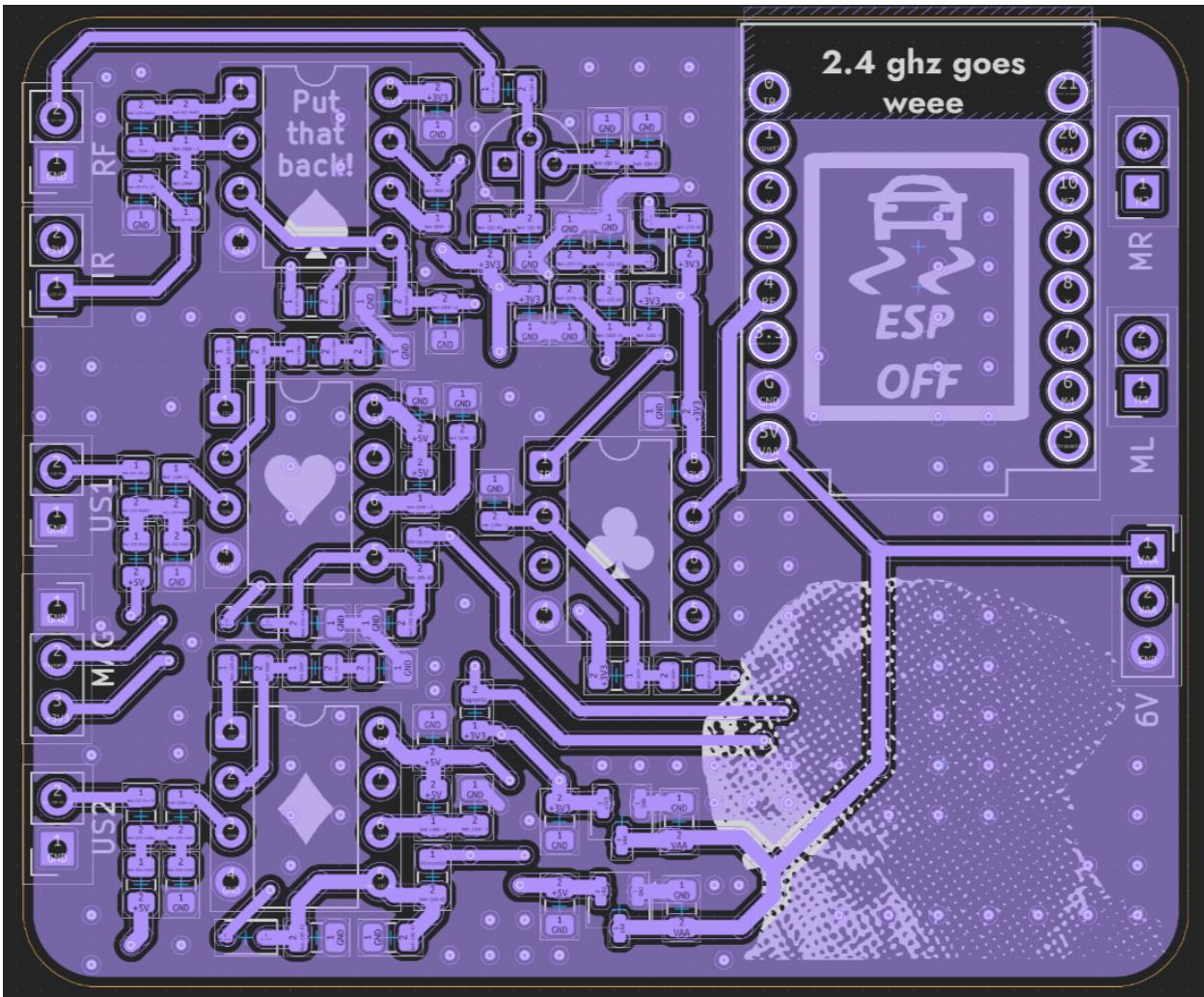


Figure 38: PCB front side with pours

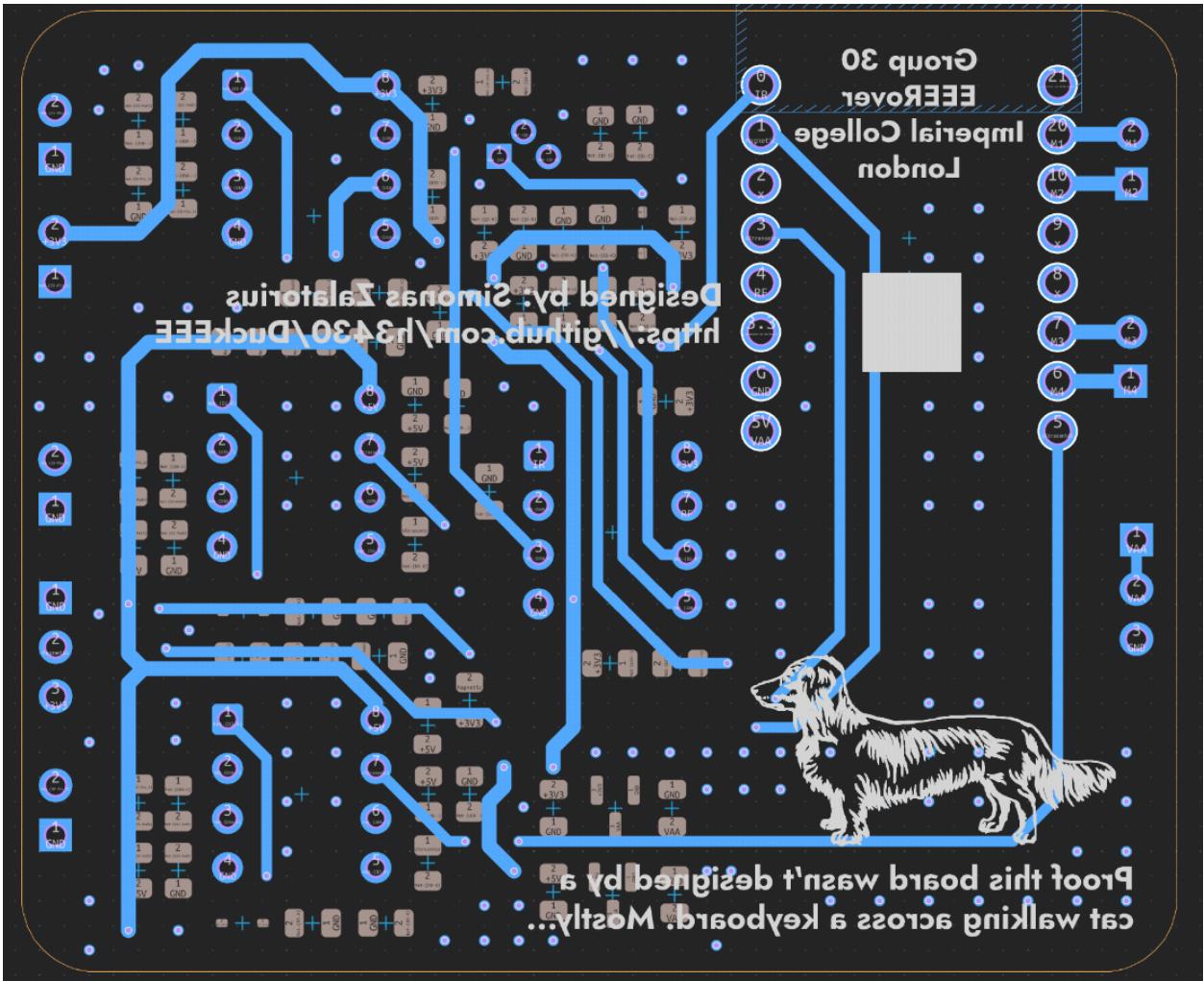


Figure 39: PCB back side

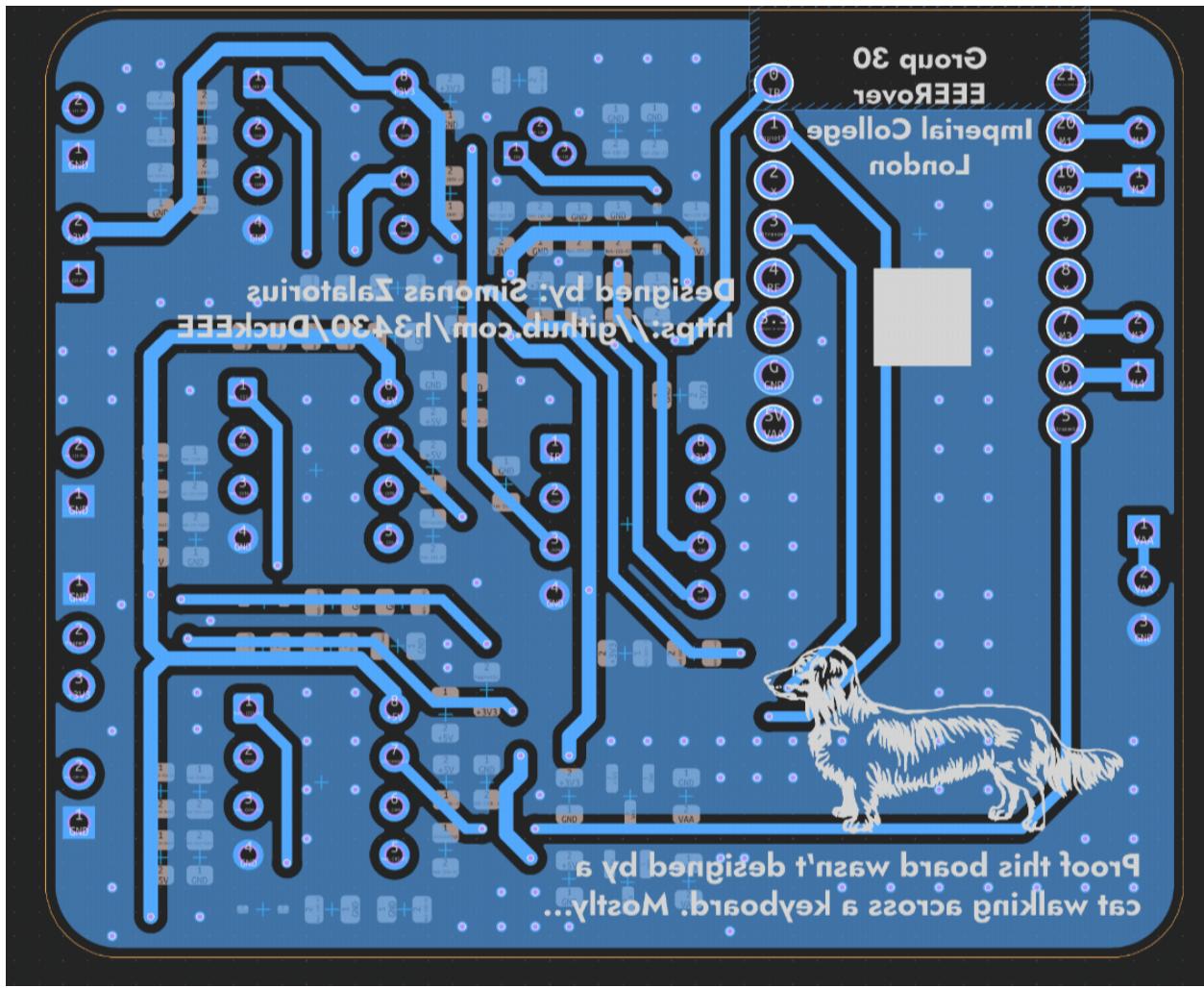


Figure 40: PCB back side pour, ground plane void intentional

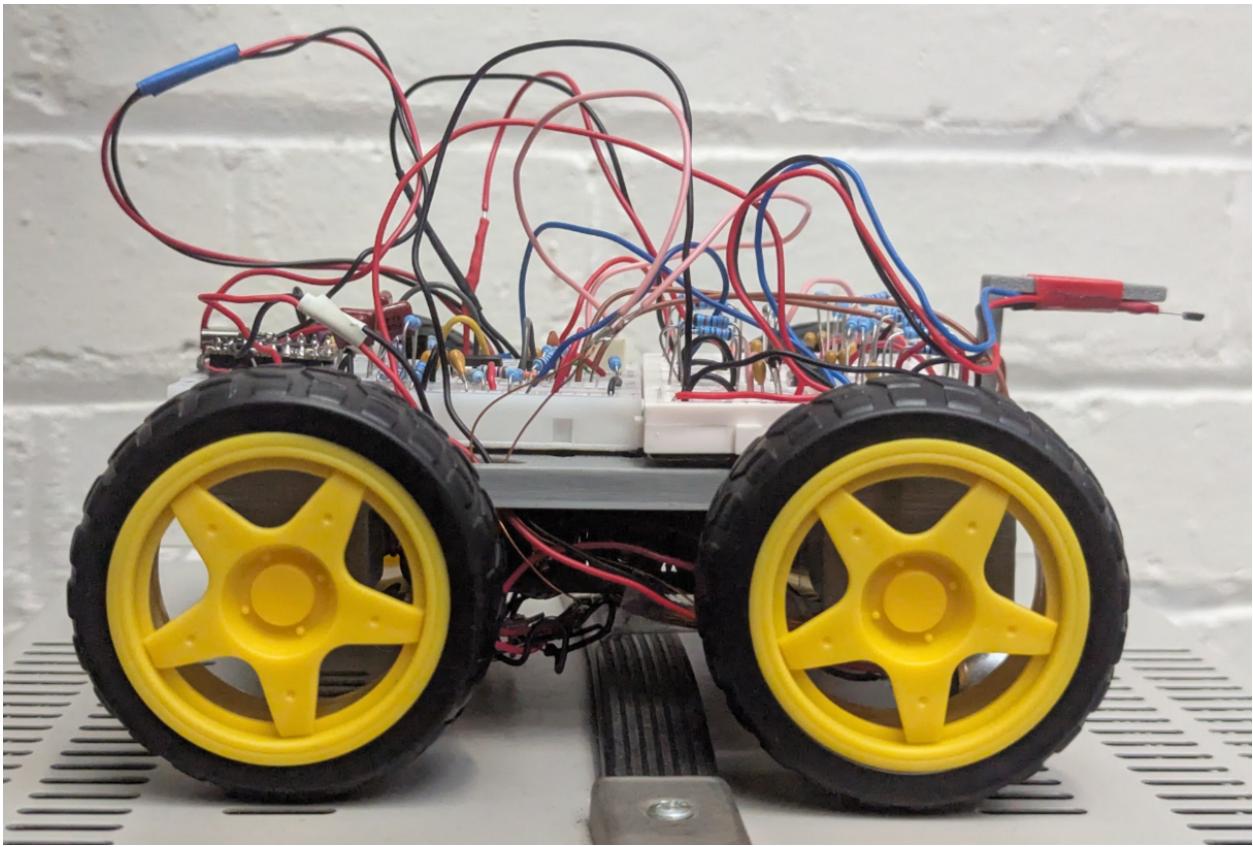


Figure 42: main rover prototype

8.3 Chassis Prototype Pictures

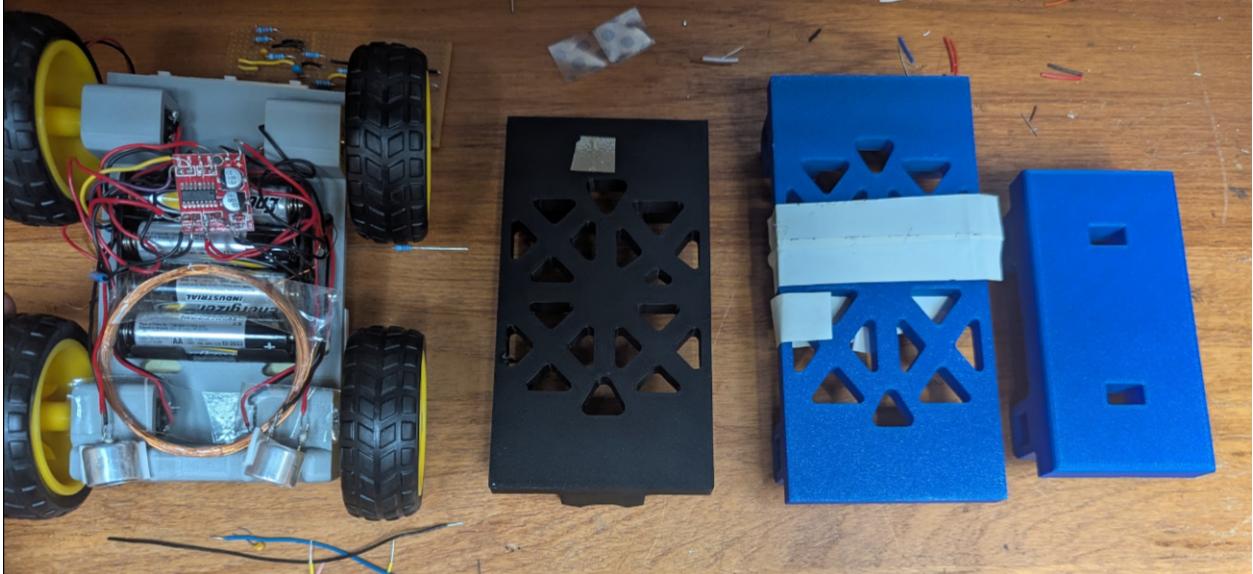


Figure 41: Chassis prototypes

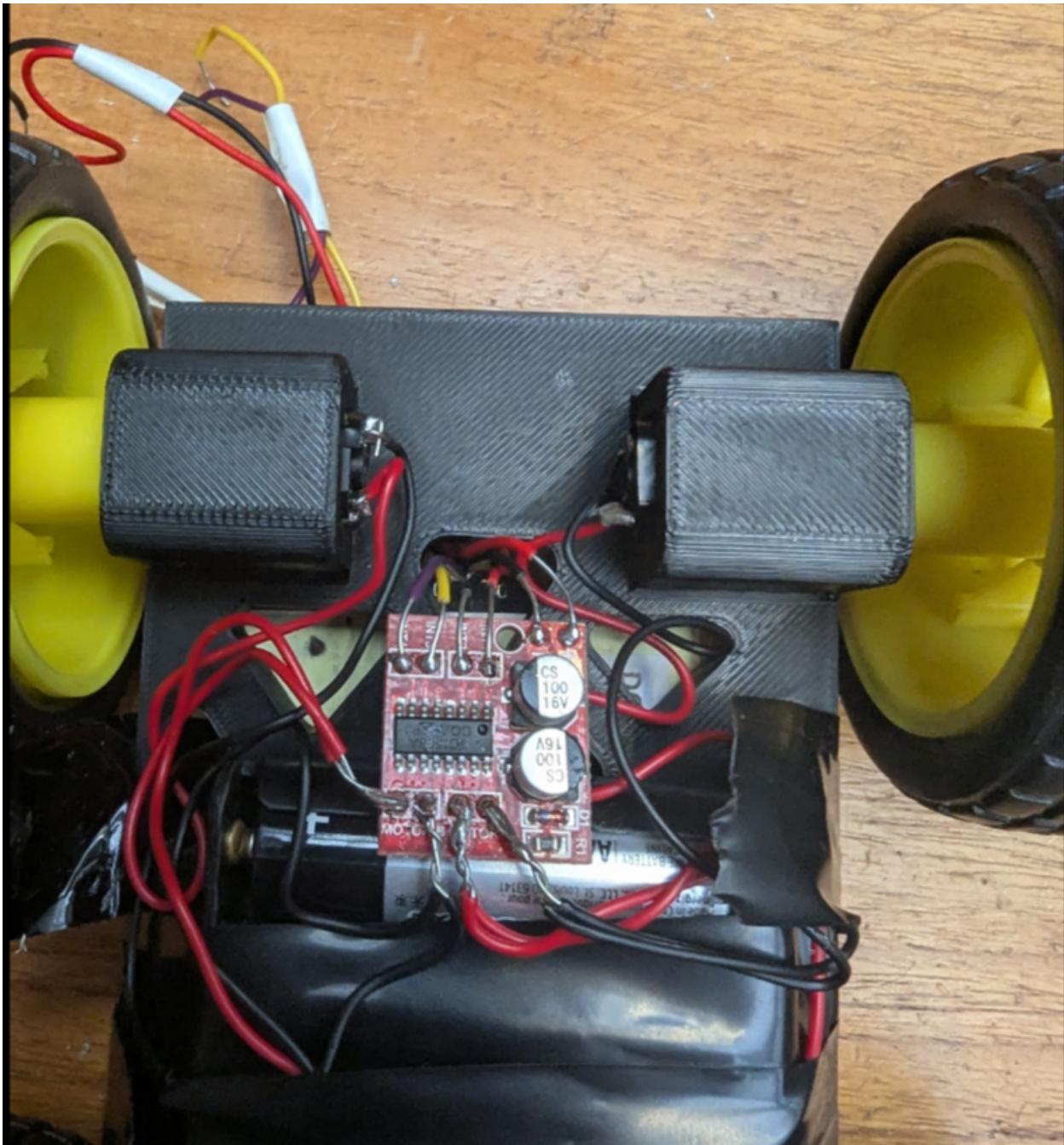


Figure 43: Older chassis prototype

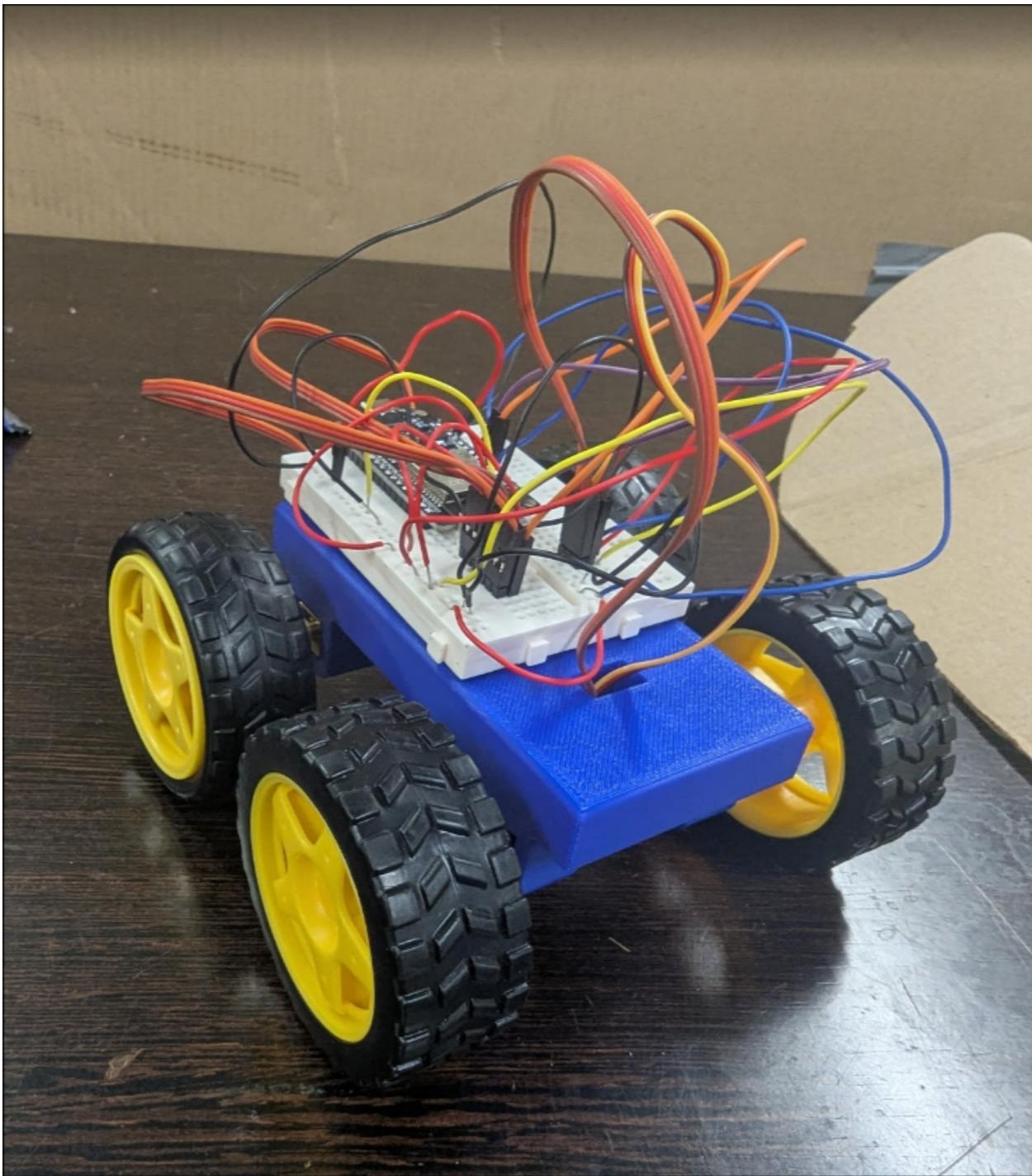


Figure 44: First chassis prototype

8.4 LLM Prompt and Response, Model: Deepseek R1 Qwen Distilled Running on T3chat

Prompt:

Env: esp32, platformio, arduino hal.

Problem: get pulsecounter setup from example, trigger on posedge, enable fliter, keep everything as simple as possible. I need just the setup code for a pulsecounter, no interrupt stuff or anything like that. Furthermore, what functions do I use to start, stop, clear, and to get values?

Example code:

```
/* Pulse counter module - Example
```

For other examples please check:

<https://github.com/espressif/esp-idf/tree/master/examples>

This example code is in the Public Domain (or CC0 licensed, at your option.)

Unless required by applicable law or agreed to in writing, this software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

```
/*
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/portmacro.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "driver/periph_ctrl.h"
#include "driver/ledc.h"
#include "driver/gpio.h"
#include "driver/pcnt.h"
#include "esp_attr.h"
#include "esp_log.h"
#include "soc/gpio_sig_map.h"

/**
 * TEST CODE BRIEF
 *
 * Use PCNT module to count rising edges generated by LEDC module.
 *
 * Functionality of GPIOs used in this example:
 *   - GPIO18 - output pin of a sample 1 Hz pulse generator,
 *   - GPIO4 - pulse input pin,
 *   - GPIO5 - control input pin.
 *
 * Load example, open a serial port to view the message printed on your screen.
 *
 * To do this test, you should connect GPIO18 with GPIO4.
 * GPIO5 is the control signal, you can leave it floating with internal pull up,
 * or connect it to ground. If left floating, the count value will be increasing.
 * If you connect GPIO5 to GND, the count value will be decreasing.
 *
 * An interrupt will be triggered when the counter value:
 *   - reaches 'thresh1' or 'thresh0' value,
```

```

*      - reaches 'l_lim' value or 'h_lim' value,
*      - will be reset to zero.
*/
#define PCNT_TEST_UNIT      PCNT_UNIT_0
#define PCNT_H_LIM_VAL      10
#define PCNT_L_LIM_VAL      -10
#define PCNT_THRESH1_VAL    5
#define PCNT_THRESH0_VAL   -5
#define PCNT_INPUT_SIG_IO   4 // Pulse Input GPIO
#define PCNT_INPUT_CTRL_IO  5 // Control GPIO HIGH=count up, LOW=count down
#define LEDC_OUTPUT_IO       18 // Output GPIO of a sample 1 Hz pulse generator

xQueueHandle pcnt_evt_queue; // A queue to handle pulse counter events

/* A sample structure to pass events from the PCNT
 * interrupt handler to the main program.
 */
typedef struct {
    int unit; // the PCNT unit that originated an interrupt
    uint32_t status; // information on the event type that caused the interrupt
} pcnt_evt_t;

/* Decode what PCNT's unit originated an interrupt
 * and pass this information together with the event type
 * the main program using a queue.
 */
static void IRAM_ATTR pcnt_example_intr_handler(void *arg)
{
    uint32_t intr_status = PCNT.int_st.val;
    int i;
    pcnt_evt_t evt;
    portBASE_TYPE HPTaskAwoken = pdFALSE;

    for (i = 0; i < PCNT_UNIT_MAX; i++) {
        if (intr_status & (BIT(i))) {
            evt.unit = i;
            /* Save the PCNT event type that caused an interrupt
             * to pass it to the main program */
            evt.status = PCNT.status_unit[i].val;
            PCNT.int_clr.val = BIT(i);
            xQueueSendFromISR(pcnt_evt_queue, &evt, &HPTaskAwoken);
            if (HPTaskAwoken == pdTRUE) {
                portYIELD_FROM_ISR();
            }
        }
    }
}

/* Configure LED PWM Controller
 * to output sample pulses at 1 Hz with duty of about 10%
 */
static void ledc_init(void)
{
    // Prepare and then apply the LEDC PWM timer configuration

```

```

ledc_timer_config_t ledc_timer;
ledc_timer.speed_mode      = LEDC_HIGH_SPEED_MODE;
ledc_timer.timer_num       = LEDC_TIMER_1;
ledc_timer.duty_resolution = LEDC_TIMER_10_BIT;
ledc_timer.freq_hz         = 1; // set output frequency at 1 Hz
ledc_timer_config(&ledc_timer);

// Prepare and then apply the LEDC PWM channel configuration
ledc_channel_config_t ledc_channel;
ledc_channel.speed_mode = LEDC_HIGH_SPEED_MODE;
ledc_channel.channel   = LEDC_CHANNEL_1;
ledc_channel.timer_sel = LEDC_TIMER_1;
ledc_channel.intr_type = LEDC_INTR_DISABLE;
ledc_channel.gpio_num   = LEDC_OUTPUT_IO;
ledc_channel.duty      = 100; // set duty at about 10%
ledc_channel_config(&ledc_channel);
}

/* Initialize PCNT functions:
 * - configure and initialize PCNT
 * - set up the input filter
 * - set up the counter events to watch
 */
static void pcnt_example_init(void)
{
    /* Prepare configuration for the PCNT unit */
    pcnt_config_t pcnt_config = {
        // Set PCNT input signal and control GPIOs
        .pulse_gpio_num = PCNT_INPUT_SIG_IO,
        .ctrl_gpio_num = PCNT_INPUT_CTRL_IO,
        .channel = PCNT_CHANNEL_0,
        .unit = PCNT_TEST_UNIT,
        // What to do on the positive / negative edge of pulse input?
        .pos_mode = PCNT_COUNT_INC, // Count up on the positive edge
        .neg_mode = PCNT_COUNT_DIS, // Keep the counter value on the negative edge
        // What to do when control input is low or high?
        .lctrl_mode = PCNT_MODE_REVERSE, // Reverse counting direction if low
        .hctrl_mode = PCNT_MODE_KEEP, // Keep the primary counter mode if high
        // Set the maximum and minimum limit values to watch
        .counter_h_lim = PCNT_H_LIM_VAL,
        .counter_l_lim = PCNT_L_LIM_VAL,
    };
    /* Initialize PCNT unit */
    pcnt_unit_config(&pcnt_config);

    /* Configure and enable the input filter */
    pcnt_set_filter_value(PCNT_TEST_UNIT, 100);
    pcnt_filter_enable(PCNT_TEST_UNIT);

    /* Set threshold 0 and 1 values and enable events to watch */
    pcnt_set_event_value(PCNT_TEST_UNIT, PCNT_EVT_THRES_1, PCNT_THRESH1_VAL);
    pcnt_event_enable(PCNT_TEST_UNIT, PCNT_EVT_THRES_1);
    pcnt_set_event_value(PCNT_TEST_UNIT, PCNT_EVT_THRES_0, PCNT_THRESH0_VAL);
    pcnt_event_enable(PCNT_TEST_UNIT, PCNT_EVT_THRES_0);
}

```

```

/* Enable events on zero, maximum and minimum limit values */
pcnt_event_enable(PCNT_TEST_UNIT, PCNT_EVT_ZERO);
pcnt_event_enable(PCNT_TEST_UNIT, PCNT_EVT_H_LIM);
pcnt_event_enable(PCNT_TEST_UNIT, PCNT_EVT_L_LIM);

/* Initialize PCNT's counter */
pcnt_counter_pause(PCNT_TEST_UNIT);
pcnt_counter_clear(PCNT_TEST_UNIT);

/* Register ISR handler and enable interrupts for PCNT unit */
pcnt_isr_register(pcnt_example_intr_handler, NULL, 0, NULL);
pcnt_intr_enable(PCNT_TEST_UNIT);

/* Everything is set up, now go to counting */
pcnt_counter_resume(PCNT_TEST_UNIT);
}

void app_main()
{
    /* Initialize LEDC to generate sample pulse signal */
    ledc_init();

    /* Initialize PCNT event queue and PCNT functions */
    pcnt_evt_queue = xQueueCreate(10, sizeof(pcnt_evt_t));
    pcnt_example_init();

    int16_t count = 0;
    pcnt_evt_t evt;
    portBASE_TYPE res;
    while (1) {
        /* Wait for the event information passed from PCNT's interrupt handler.
         * Once received, decode the event type and print it on the serial monitor.
         */
        res = xQueueReceive(pcnt_evt_queue, &evt, 1000 / portTICK_PERIOD_MS);
        if (res == pdTRUE) {
            pcnt_get_counter_value(PCNT_TEST_UNIT, &count);
            printf("Event PCNT unit[%d]; cnt: %d\n", evt.unit, count);
            if (evt.status & PCNT_STATUS_THRES1_M) {
                printf("THRES1 EVT\n");
            }
            if (evt.status & PCNT_STATUS_THRES0_M) {
                printf("THRES0 EVT\n");
            }
            if (evt.status & PCNT_STATUS_L_LIM_M) {
                printf("L_LIM EVT\n");
            }
            if (evt.status & PCNT_STATUS_H_LIM_M) {
                printf("H_LIM EVT\n");
            }
            if (evt.status & PCNT_STATUS_ZERO_M) {
                printf("ZERO EVT\n");
            }
        } else {
            pcnt_get_counter_value(PCNT_TEST_UNIT, &count);
        }
    }
}

```

```

        printf("Current counter value :%d\n", count);
    }
}
}

```

Response:

Here's a simplified version of the pulse counter setup code for ESP32:

```

/***
 *include "driver/pcnt.h"
 *include "esp_log.h"

#define PCNT_UNIT PCNT_UNIT_0
#define PCNT_INPUT_GPIO 4 // GPIO pin for pulse input

void setupPulseCounter() {
    pcnt_config_t config = {
        .pulse_gpio_num = PCNT_INPUT_GPIO,
        .channel = PCNT_CHANNEL_0,
        .unit = PCNT_UNIT,
        .pos_mode = PCNT_COUNT_INC, // Increment on positive edge
        .neg_mode = PCNT_COUNT_DIS // Disable counting on negative edge
        .lctrl_mode = PCNT_MODE_KEEP, // Keep counting direction when control is low
        .hctrl_mode = PCNT_MODE_KEEP // Keep counting direction when control is high
    };

    // Initialize PCNT unit
    pcnt_unit_config(&config);

    // Enable input filter with 100 clock cycles filter value
    pcnt_set_filter_value(PCNT_UNIT, 100);
    pcnt_filter_enable(PCNT_UNIT);

    // Clear and start counter
    pcnt_counter_clear(PCNT_UNIT);
    pcnt_counter_resume(PCNT_UNIT);
}

// Functions to control the counter
void startCounter() {
    pcnt_counter_resume(PCNT_UNIT);
}

void stopCounter() {
    pcnt_counter_pause(PCNT_UNIT);
}

void clearCounter() {
    pcnt_counter_clear(PCNT_UNIT);
}

int16_t getCounterValue() {

```

```

    int16_t value;
    pcnt_get_counter_value(PCNT_UNIT, &value);
    return value;
}

void app_main() {
    setupPulseCounter();

    while(true) {
        // Example usage: print counter value every second
        int16_t count = getCounterValue();
        ESP_LOGI("PulseCounter", "Current count: %d", count);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}
```

```

Key functions used:

- ‘pcnt\_unit\_config()’: Configures the pulse counter unit
- ‘pcnt\_filter\_enable()’: Enables input filtering
- ‘pcnt\_counter\_resume()’: Starts the counter
- ‘pcnt\_counter\_pause()’: Stops the counter
- ‘pcnt\_counter\_clear()’: Resets counter to zero
- ‘pcnt\_get\_counter\_value()’: Reads current counter value

The code sets up a pulse counter on GPIO4, triggers on positive edges, and includes input filtering to debounce the signal. The counter increments each time a positive edge is detected on the input pin.