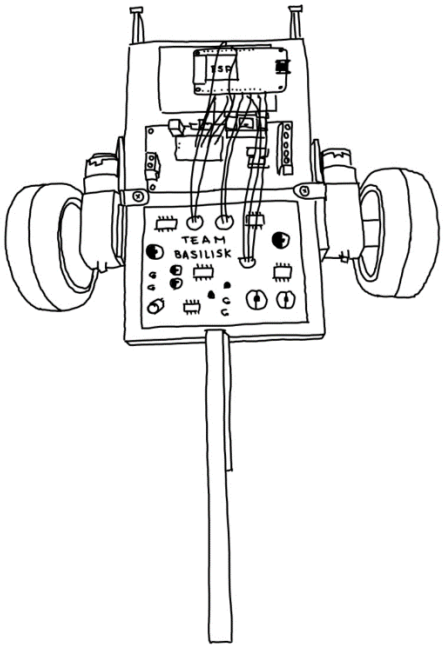
IMPERIAL

Imperial College London

Department of Electrical and Electronic Engineering

Team Basilisk Report
EEERover2024



Module: ELEC40006 Engineering Group Design Project

Authors (1st Year EEE/EIE):
Eva Littlewood (02391728)
Jeremy Tan (02372939)
Olly Cassidy (02380001)
Sam Mueller-Menrad (02489933)
Selime Ozyurt (02493530)
Tara Venkatesam (02500340)

Submission Date: 13th June 2024

GitHub: https://github.com/ollycassidy13/EEERover2024

Submitted to: Dr. Zohaib Akhtar

IMPERIAL

IMPERIAL

Contents

Abstract

This report describes the final design for the Team Basilisk Rover which collects and decodes data to determine the name and species of different lizards as specified in the design brief. It details our design process, the choices we made as a group, briefly outlines the problems we encountered when developing our solution and includes our final design and verification processes such as circuit diagrams, our PCB design, code snippets and relevant testing.

Definitions

To shorten common terms throughout the report for readability purposes, we have used the following shorthand:

- Op-amp refers to an operational amplifier
- ESP refers to the Espressif ESP32-C3-32S WiFi and Bluetooth enabled microcontroller module

Problem Statement

We had to design a remotely controlled rover that could explore a given environment and detect, decode and display signals from lizards on a webpage to identify the name and species of each lizard. The lizards emitted ultrasound waves to communicate their name and used radio and infrared waves alongside had a magnetic polarity to communicate their species.

The rover will be tested in a given deserted area and will be assessed against the following criteria as specified in the design brief[1]:

- Is it possible to find the characteristics of all the lizards?
- Is the design cost and weight effective?
- Is the rover manoeuvrable enough to negotiate the environment?
- Is the construction robust and reliable?
- Is the remote-control interface logical and easy to use?

Specification

After analysing the design specification, we decided to break it down into subsections and create our own specification to reference our solution against.

Cost

The overall cost of the rover must be under £60. We were allowed to use different equipment readily available in the first-year labs and robotics labs and we could order components from selected Imperial

[1] 10/06/2024 16:06: 'GitHub: EEERover2024' - https://github.com/hakanmerdan/EEERover2024/blob/main/doc/brief.md

suppliers to EEStores. We allocated a cost of £25 towards components for the rover, allowing £35 to be remaining for a PCB should our design lead us to require one.

Weight

The rover had a weight limit of 750g as the lizards would be scared off if the rover is too heavy. To ensure the rover was able to navigate the arena quickly and efficiently, we decided out rover would have weight limit of 700g.

Manoeuvrability

The environment is mostly flat with a few sticks, debris, elevated platforms, rocks and trees as obstacles. The rover will either have to move around these or move over them.

Sensors

The sensors should all be able to detect the signals from a minimum of 6cm away. This is due to the size of the lizard, and the placement of the sensors within the lizard. There should be 4 sensors, magnetic, infrared, radio and ultrasound, to allow all the signals to be detected and decoded.

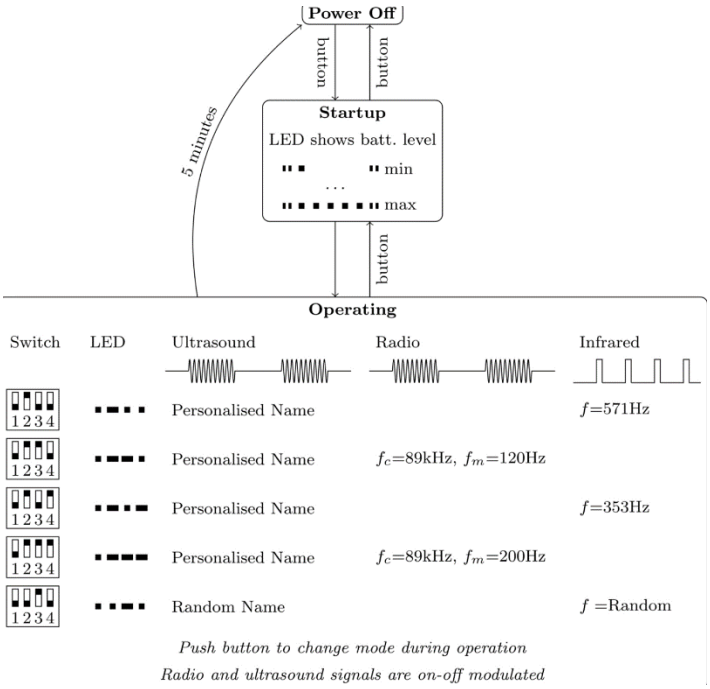


Figure 1: Table showing the signals emitted by the lizards

Useability

We aimed to have a user-friendly user interface (UI) which should be simple to use and require no instruction. We also aim to make the rover easy to control which is especially important due to the intricate nature of positioning the sensors around the lizard.

Task Distribution

Having decided on how the tasks would be split, we allocated each task to distinct members:

- **Olly Cassidy** was the Project Manager, developed the radio signal detection, amplification and demodulation and designed the PCB and chassis
- **Sam Müller-Menrad** worked on the magnetic field detection and interpretation, the mechanical design and the motor driver circuit
- **Selime Ozyurt** developed the ultrasound detection, amplification and demodulation
- **Tara Venkatesam** designed the infrared pulse detection and interpretation
- **Jeremy Tan** programmed the webpage's front and back-end and code to allow the microcontroller to interpret all the sensor circuit outputs
- **Eva Littlewood** was the Editor and programmed the webpage's front and back-end and code to allow the microcontroller to interpret all the sensor circuit outputs

We then further broke down these larger subsections into smaller tasks which we then used Notion for to assign them to specific members and set deadlines for each task.

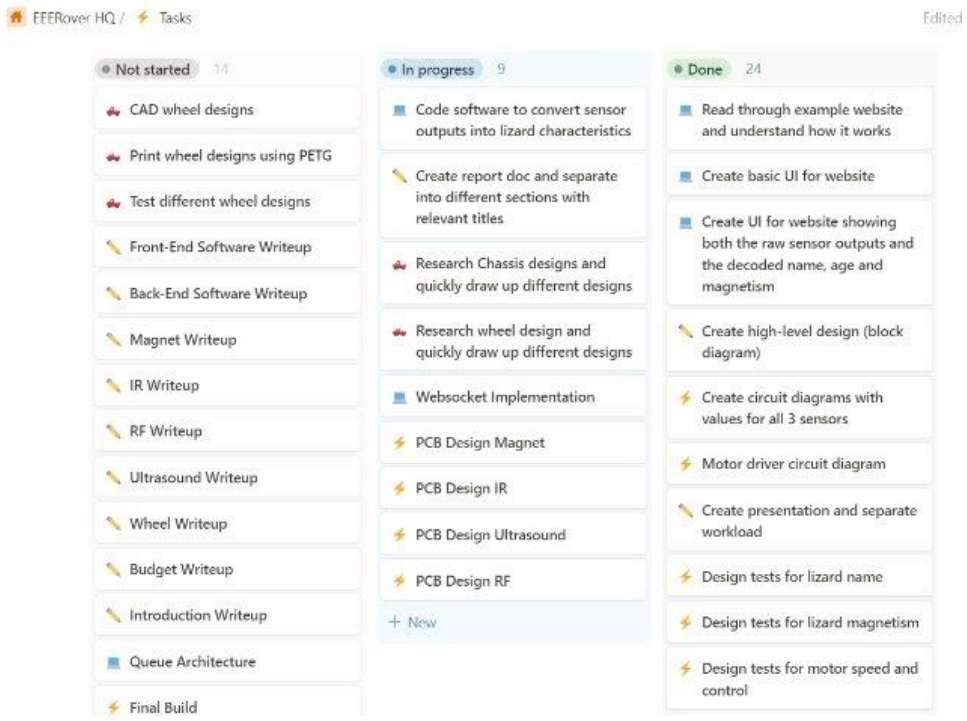


Figure 2: Notion to keep track of tasks

Initial Solution Description

To help briefly illustrate our solution, we created a high-level block diagram to help break down the problem into smaller sections. From left to right, there are 4 sensor blocks; infrared, radio and magnetic were used to determine the species of the lizard and ultrasound was used to receive UART packets of data for the name. The microcontroller then decoded these inputs and hosted the server upon which the website could be accessed, sending the decoded data to be displayed on the website for the user. The server also provided inputs to the microcontroller, which were passed to the motors to control the movement of the buggy, and determined when the microcontroller should update its output from the sensors.

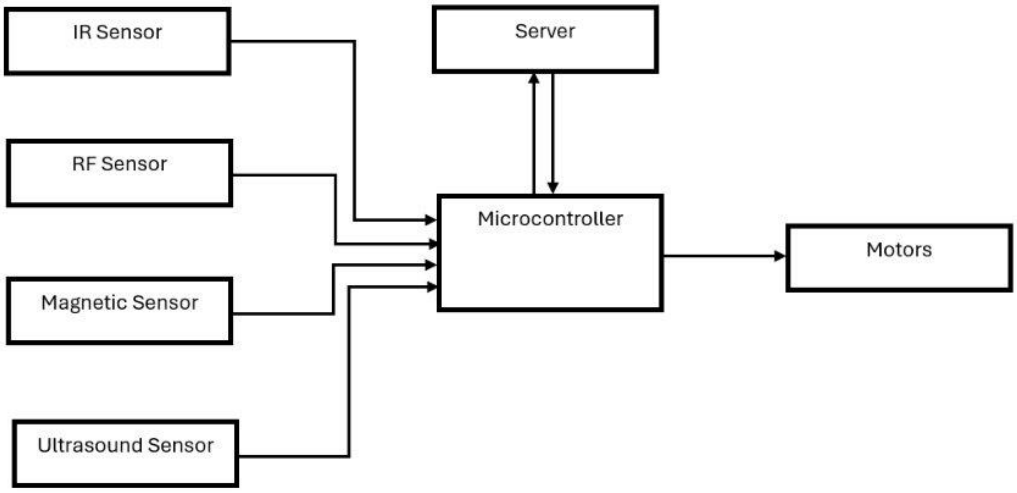


Figure 3: Basic block diagram of our solution

Sensor Design

Radio Receiver

Overview

The radio receiver subsystem provides a digital input to the microcontroller to allow the radio input to be decoded to determine the species of the lizard. The subsystem demodulates an amplitude shift key (ASK) modulated sine wave input where it is then fed into software which measures the frequency of the signal and determines if it is close to 120Hz or 200Hz therefore helping determine the species of the lizard.

Inputs	Outputs
ASK Modulated Radio Wave (89kHz carrier) from a distance of 6cm or greater	Digital Output (0V low / 3.3V high)

Figure 4: Specification table showing the inputs and outputs of the radio receiver circuit

Initial Ideas

First, we had to translate the radio pulses emitted by the lizards into electrical signals so that we could process them with our circuit. We achieved this initially using an inductor acting as a coiled antenna. This works by Faraday's law of Induction: the electromagnetic (EM) radio waves induce a changing magnetic field which causes a changing electric field producing a potential across the inductor. As Faraday's Law of Induction states that the induced electromotive force (EMF) is proportional to the number of turns in the

coil, which is proportional to the inductance of the inductor, we chose to use an inductor with a large inductance to increase the induced EMF.

We used a band pass filter with a peak at 89kHz to filter for the carrier wave we are aiming to receive and reduce interference from noise. We then applied transfer functions to the circuit to derive the capacitance and inductance of the necessary circuit components. To verify our design, we simulated our circuit in LTSpice and confirmed the peak.

$$f_c = \frac{1}{2\pi\sqrt{LC}}$$

$$f_c = \frac{1}{2\pi\sqrt{100 \times 10^{-3} \times 33 \times 10^{-12}}}$$

$$f_c = 87.6\text{kHz}$$

Figure 5: Calculations verifying the 89kHz peak of the band pass filter

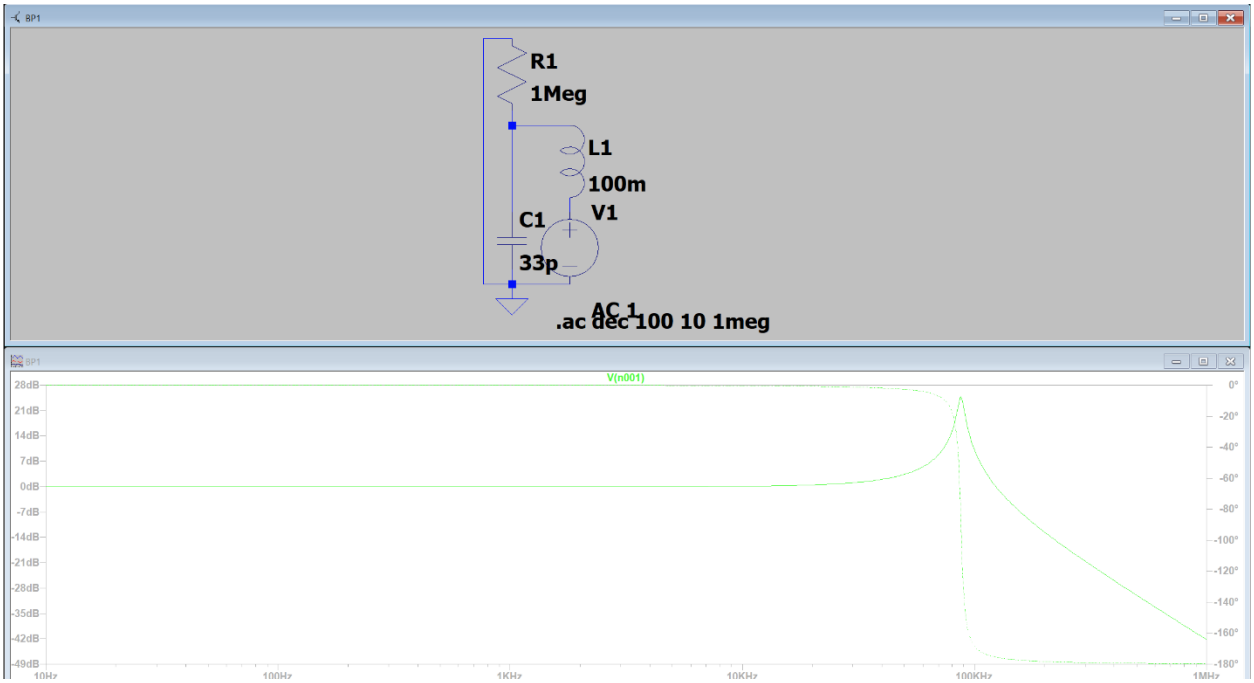


Figure 6: LTSpice AC Analysis Simulation of the radio receiver

To measure the frequency of the pulses, we needed to isolate the positive peaks of the signal, so we achieved this by using a precision halfwave rectifier and inverting amplifier with a gain of -1 to rectify the circuit and so that the 0.7V lost with a standard halfwave rectifier is recovered. This is important as the

signal amplitude is very small (<0.7V), so without this the entire signal would be lost. An RC network is then used as an analogue 'sample and hold'. This allows the sine wave peaks to be smoothed with a small ripple. A comparator is then used to create a digital signal to be read by the microcontroller.

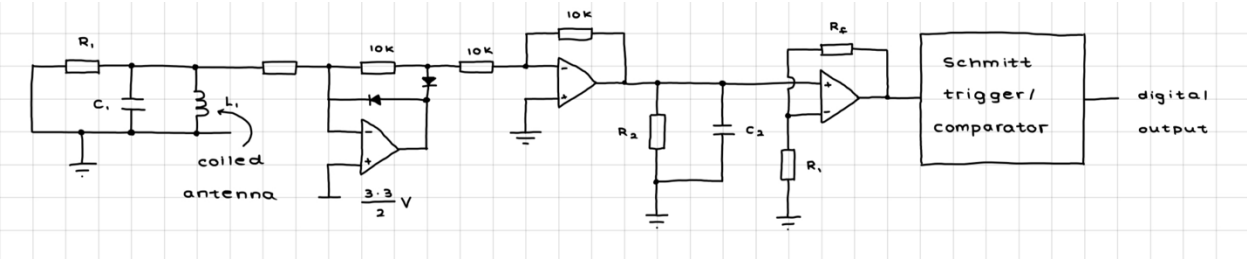


Figure 7: full circuit diagram of the initial design for the radio receiver

Final Design

Next, we aimed to reduce the number of components in our circuit and make it more efficient to reduce the overall space the circuit takes and the number of op-amps and therefore cost. We chose to use a LM358 op-amp due to its low cost, high gain bandwidth product, high slew rate and its small size for a chip containing two op-amps. The bandpass filter is used again with a peak at 89kHz and values of resistance, inductance and capacitance to filter for the carrier wave we are aiming to receive and reduce interference from noise such as the 50Hz noise from electrical mains. We also decided to use a 100mH inductor instead of the coiled antenna in the band pass filter due to its far greater inductance than any coil we were able to create ourselves (~500x greater). A non-inverting amplifier configured on a single supply with a gain of ~500 is used to rectify and amplify the signal as shown in *Figure 9*; the half bridge rectifier in the previous design can be eliminated as the op-amp allows the negative parts of the signal saturate at 0 and positive parts of the signal are amplified. A capacitor is then used with the output impedance of the op-amp to create an RC network which acts as an analogue 'sample and hold'. This allows the sine wave peaks to be smoothed with a small ripple. A comparator is used to create a digital signal and a BJT configured as a switch gives a rail-to-rail inverted output. As we are only measuring the frequency of the output to determine the species of the lizard, the output being inverted does not affect the wider purpose of the circuit within the project.

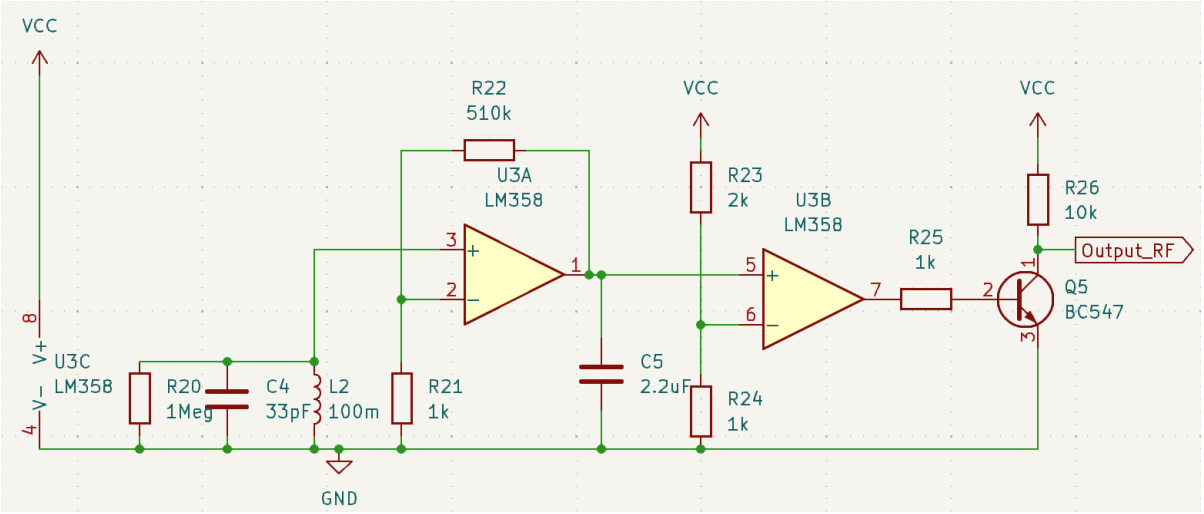


Figure 8: Circuit diagram of the final radio receiver design

$$G = 1 + \frac{R_f}{R_1}$$

$$G = 1 + \frac{510}{1}$$

$$G = 511$$

Figure 9: Calculations to find the gain of the non-inverting amplifier

$$V_{\text{out}} = V_{\text{in}} \frac{R_2}{R_1 + R_2}$$

$$V_{\text{out}} = 3.3 \frac{1}{1 + 2}$$

$$V_{\text{out}} = 1.1V$$

Figure 10: Calculations for the potential divider of the comparator in the radio receiver

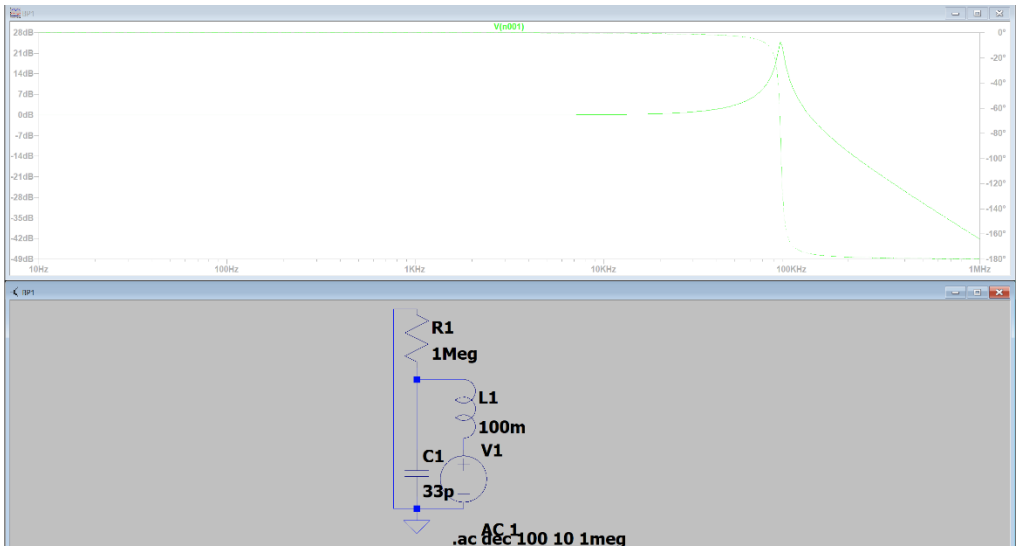


Figure 11: LTSpice simulation of the bandpass filter of the radio receiver. A peak of ~89kHz is shown as required

Testing

We initially considered measuring the output of the bandpass filter with a picoscope to ensure it worked correctly however upon further research we realised the output of the bandpass filter should not be measured with the picoscope due to the picoscope's input impedance of 1MΩ in parallel with a 12pF capacitance. This is the same resistance as the resistor in the band pass filter and therefore causes the two resistances to act in parallel behaving together as a 500kΩ resistor. This reduces the peak and shifts the peak frequency to 74.9kHz as the capacitances act in parallel, therefore giving a different transfer function to how it will run in practice giving a false test result.

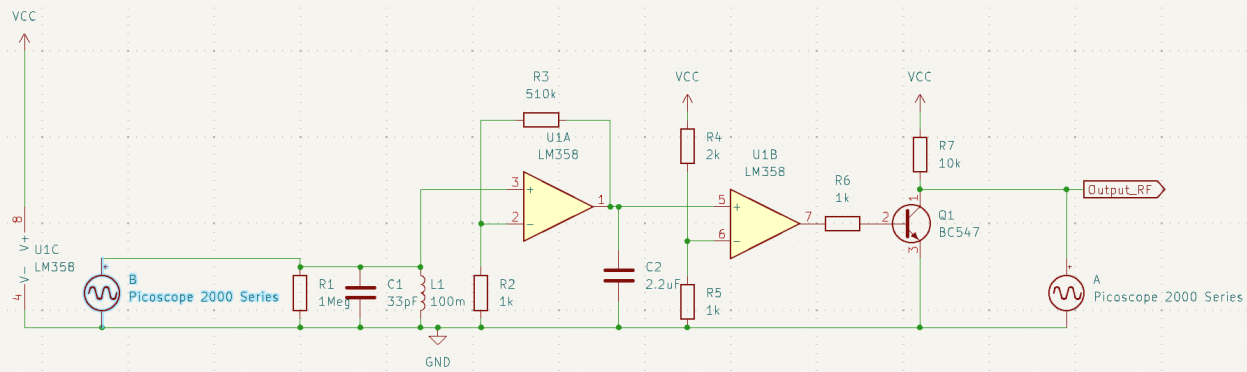


Figure 12: circuit diagram showing the problem with the original way of measuring the bandpass filter using a picoscope due to the picoscope's input impedance

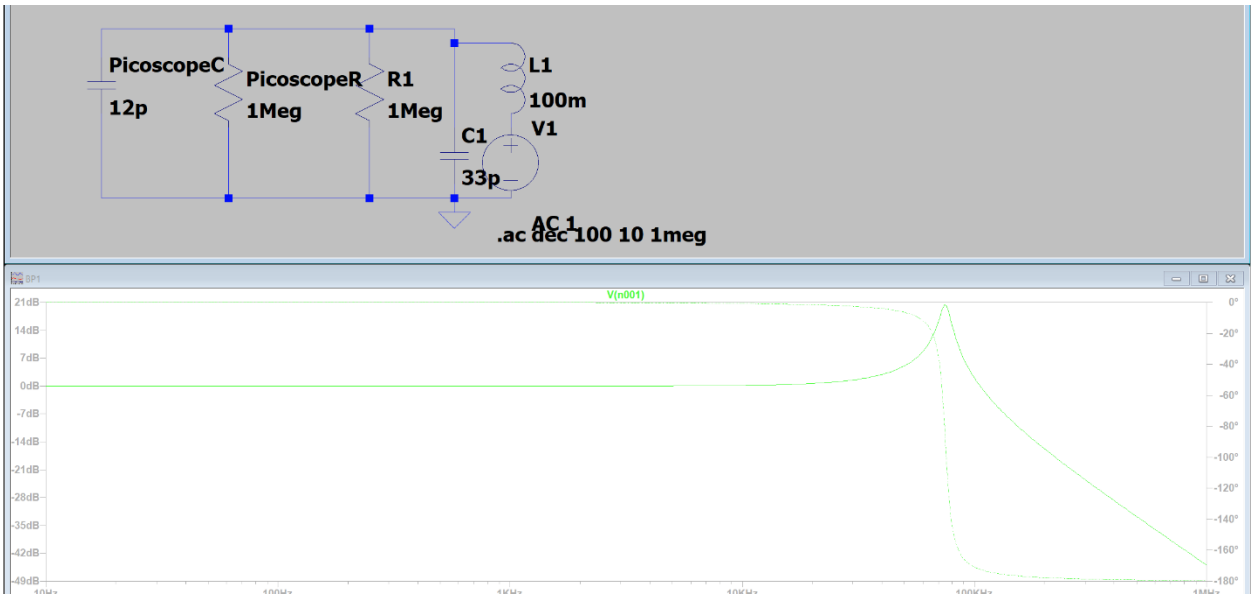


Figure 13: LTSpice AC Analysis Simulation of picoscope in bandpass filter

With the test lizard transmitting the ASK modulated wave from 6cm away initially and slowly moving away until the signal was no longer received correctly, the picoscope was set up to measure the output of the non-inverting amplifier and the output of the subsystem as shown in *Figure 14.*

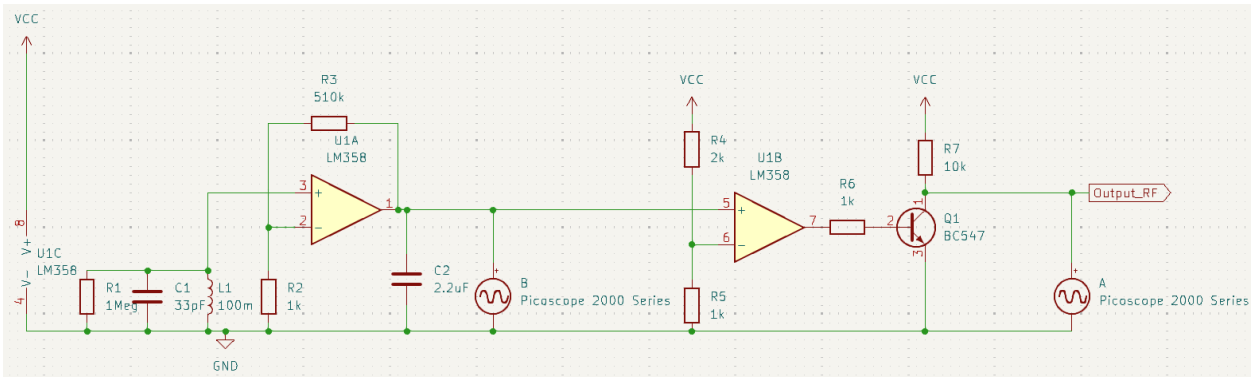


Figure 14: circuit diagram showing our solution to this problem

Results

The picoscope trace shows a result of 198.5Hz which is close to the specified 200Hz and within the ±10Hz tolerance that will be built into the software due to imperfections in the lizard's construction. The high voltage is 3.3V and the low voltage is 0V, which meets the high and low voltage ranges required for the next subsystem's components. We were able to make reliable accurate results from ~8cm away.

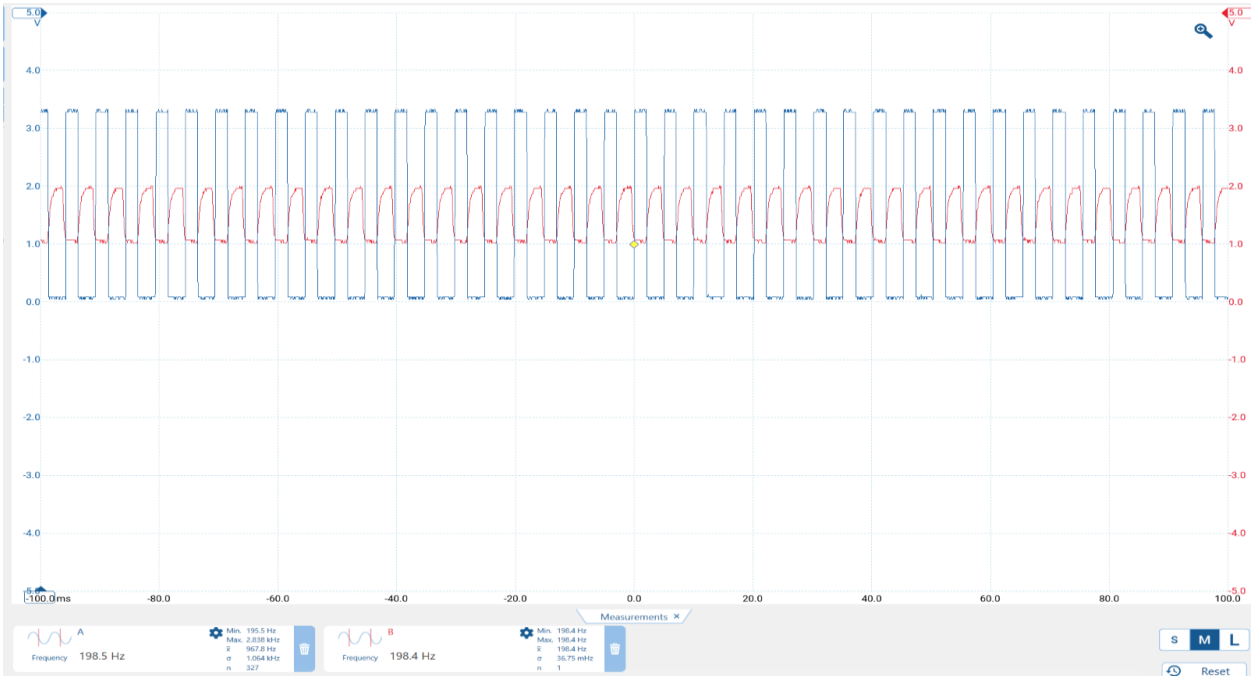


Figure 15: Picoscope trace showing a result of 198.5Hz. The output of the non-inverting amplifier is shown in red, and the output of the subsystem is shown in blue.

Evaluation

The subsystem meets the specification; It works from a distance greater than the minimum of the other sensors with the distance of the magnetic sensor being the limiting factor. It withstands interference from other frequencies well, such as 50Hz electrical noise which is not present in the output signal. It could have been improved by using a rail-to-rail op amp instead which would have removed the need for the BJT and therefore the signal would not have been inverted, however this would have significantly increased the cost, or space and therefore weight of the subsystem for something which doesn't change the functionality of the output due to the frequency being the key element. Therefore, the design was kept this way as overall this was the most efficient solution.

Ultrasound Sensor

Overview

This subsystem will provide a digital input to the microcontroller to allow the ultrasound input to be decoded. The subsystem demodulates a two-level ASK modulated input representing the lizard's name encoded using ASCII character codes in UART packets with 1 start bit and 1 stop bit. The lizard emits the input at 600 bits per second and each lizard's name is four characters long including an initial # symbol. The output of the subsystem is then fed into software which determines the name of the lizard.

Inputs	Outputs
Two-Level ASK Modulated Ultrasound Wave (40kHz Carrier) from a distance of 6cm or greater	Digital Output (0V low/3.3V high)

Figure 16: Specification table showing the inputs and outputs of the ultrasound receiver circuit

Initial Ideas

We chose to use the PROWAVE 400SR160 Ultrasound Sensor[2] in this circuit due to its peak carrier frequency being 40kHz and its low cost. This sensor is appropriate for our application as its peak receiving frequency matches the frequency we are aiming to receive. This is shown in *Figure 17* where the solid red line shows the impedance of the sensor and the solid blue line shows the phase of the signal as the sensor receives different frequencies. Originally, we designed a circuit inspired by our radio receiver design shown below in *Figure 19*. A band pass filter combined with an ultrasound sensor with a peak at 40kHz is used to filter for the carrier wave we are aiming to receive and reduce interference from noise. A precision halfwave rectifier and inverting amplifier with a gain of –1 is then used to rectify the signal so only the positive peaks are left, and so that the 0.7V lost with a standard halfwave rectifier is recovered. An RC network is then used as an analogue 'sample and hold'. This allows the sine wave peaks to be smoothed with a small ripple. A non-inverting amplifier followed by a comparator is then used to create a digital signal.

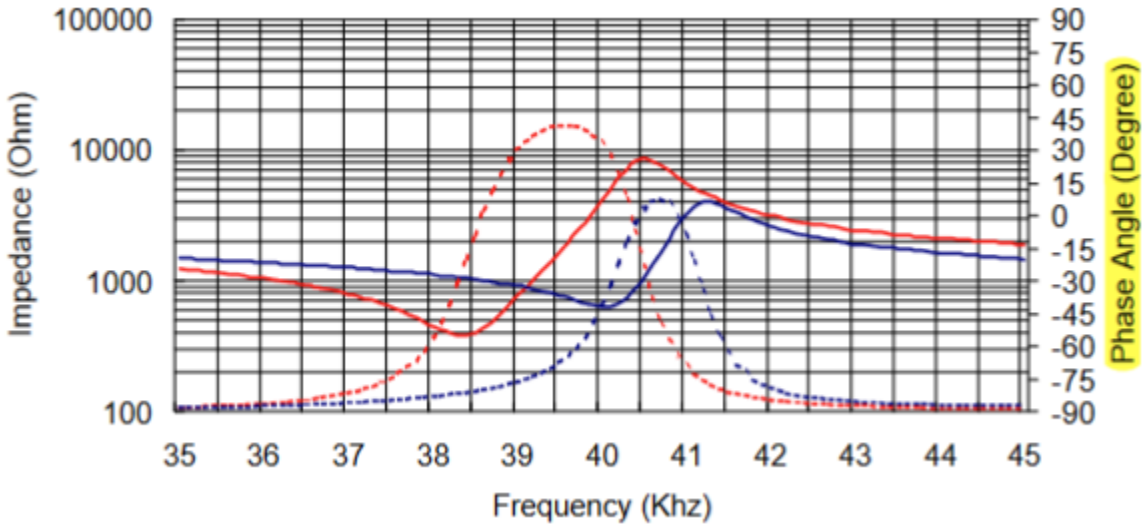


Figure 17: Impedance/phase characteristic of the PROWAVE 400SR160

[2] 18:34 09/06/2024: 'PROWAVE 400SR160 Datasheet' – https://www.farnell.com/datasheets/3109335.pdf

400SR160	Receiver
Center Frequency	40.0±1.0Khz

Figure 18: PROWAVE 400SR160 datasheet values showing the centre frequency of the receiver.

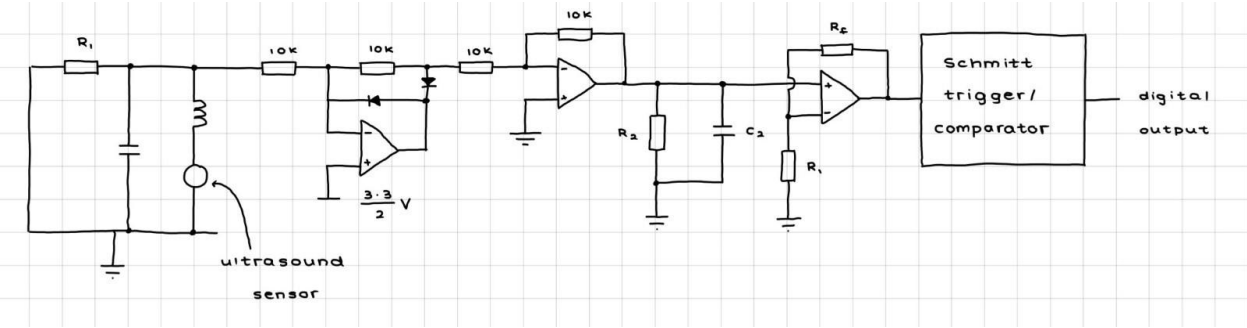


Figure 19: Initial ultrasound circuit design

Final Design

To improve our design, we looked to reduce the number of components required to implement the same function. We aimed to do this by exploiting the imperfections with real components, such as the output impedance of the op amp being used as the resistance in the RC network. A single supply was also used so that a non-inverting amplifier could be used to rectify and amplify the signal in one stage, building on the idea of using the real-world properties of the components available to us.

Our bandpass filter in our final design is identical to our previous design; it already had a peak at 40kHz to filter for the carrier wave we were aiming to receive while reducing interference. After further investigation, we found we could use an op-amp to both rectify and amplify the signal therefore eliminating the need for the extra components such as in the half-bridge rectifier in our initial design. We used an op-amp in a non-inverting amplifier configuration on a single supply with a gain of 11 as this caused the negative parts of the signal to saturate at 0 and the positive parts of the signal to be amplified therefore both rectifying and amplifying the positive sections of signal to give a voltage large enough for the ripple in the sample and hold to not be an issue. We took ideas from our previous ultrasound receiver design to create an analogue ‘sample and hold’ using a capacitor with the output impedance of the op-amp to create an RC network which allowed the sine wave peaks to be smoothed with a small ripple. Finally, to translate our output into a signal that our microcontroller could easily process in software, we used a comparator and a BJT configured as a switch giving a rail-to-rail inverted output to create a digital signal with 0V LOW / 3.3V HIGH logic. Our calculations to determine the correct component values for the bandpass filter with a peak at 40kHz, gain of the non-inverting amplifier and potential divider in the comparator are shown below in *Figures 21, 22, 23 and 24*. We decided not to further invert the signal so that the output would not be inverted as this would require a greater hardware cost and therefore increase cost, space and weight, which were all limited according to our specification. Instead, we decided

to correct this in software which was much easier to implement and allowed us to create the same accurate result with fewer components.

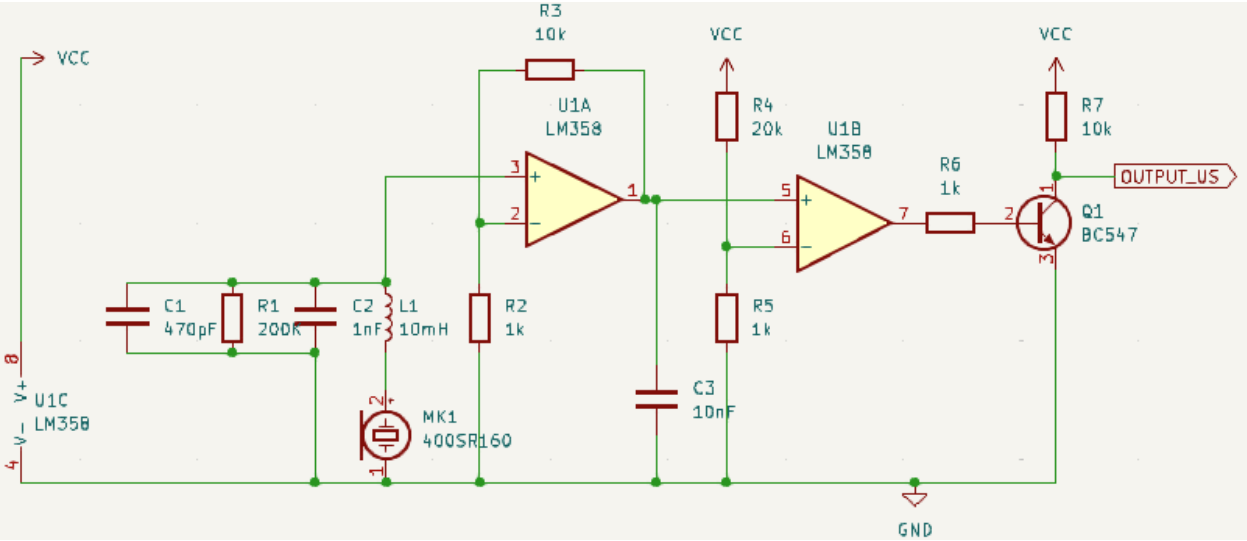


Figure 20: Final ultrasound design

$$f_c = \frac{1}{2\pi\sqrt{LC}}$$

$$f_c = \frac{1}{2\pi\sqrt{10 \times 10^{-3} \times 1.5 \times 10^{-9}}}$$

$$f_c = 41 \cdot 1\text{kHz}$$

Figure 21: Calculations showing the peak of ultrasound receiver's bandpass filter

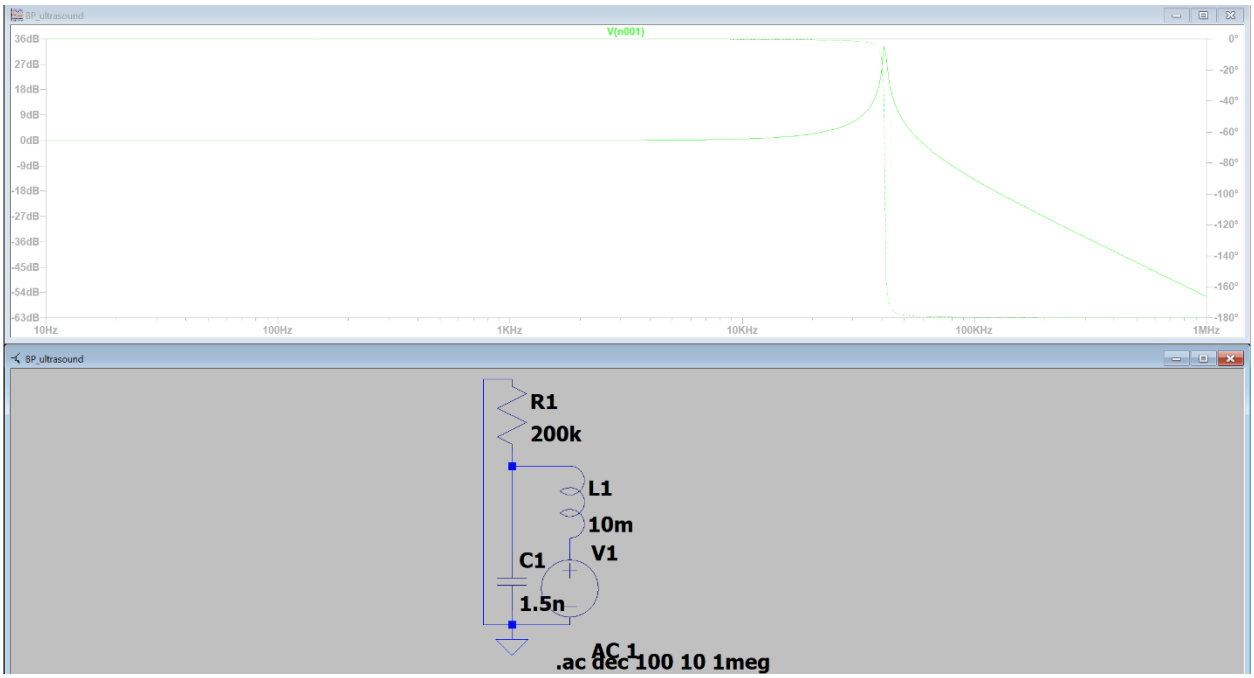


Figure 22: LTSpice simulation of the bandpass filter of the ultrasound receiver

$$G = 1 + \frac{R_f}{R_1}$$

$$G = 1 + \frac{10}{1}$$

$$G = 11$$

Figure 23: Calculations showing the gain of the non-inverting amplifier in the ultrasound receiver

$$V_{\text{out}} = V_{\text{in}} \frac{R_2}{R_1 + R_2}$$

$$V_{\text{out}} = 3.3 \frac{1}{20 + 1}$$

$$V_{\text{out}} = 157\text{mV}$$

Figure 24: Calculations showing the potential divider for the ultrasound receiver's comparator

Testing

With the test lizard transmitting the two- level ASK modulated wave from a distance of 6cm, the picoscope should be set up to measure the output of the band pass filter and the output of the subsystem. The lizard was then slowly moved away until the signal was no longer received correctly, and the maximum distance where our desired output could still be received was recorded.

Results

The picoscope trace shows the correct signal with the UART packets of data, but it is inverted as specified in our final design. The high voltage is 3.3V and the low voltage is 0V, which meets the high and low voltage ranges required for the microcontroller. Upon further testing, we found it worked from a maximum distance of 16cm.

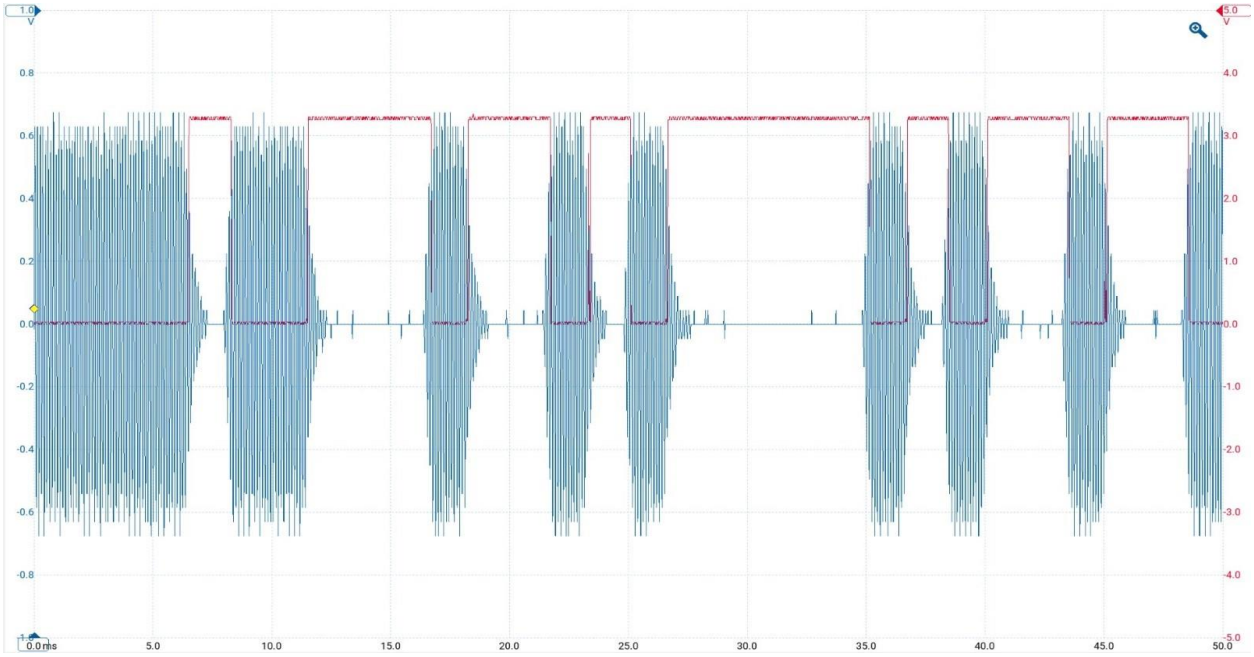


Figure 25: Picoscope trace showing the output signal of the ultrasound circuit

Evaluation

The subsystem meets our specification. It works from a distance far greater than the minimum of the other sensors with magnetic working from 6cm away. It also withstands interference from other frequencies well, such as 50Hz electrical noise is not present in the output signal.

It could have been improved by using a rail-to-rail op amp instead, removing the need for the BJT, so the signal would not have been inverted. We decided, however, that as this would have significantly increased the cost, space and therefore the weight of the subsystem to simply correct this in software which was easy to do and required no extra components, space or weight. Therefore, the design was kept this way as we felt overall that this was the most efficient solution.

Infrared (IR) Receiver

Overview

The IR receiver subsystem should provide a digital input to the microcontroller to let the infrared input be decoded to determine the lizard's species. The subsystem translates infrared light of wavelength 950nm into a digital output which is passed to software which measures the frequency of the signal and determines whether it is close to 571Hz or 353Hz, helping determine the species of the lizard. The subsystem should be able to measure the system from 6cm away.

Inputs	Outputs
Infrared light (wavelength 950nm) from a distance of 6cm	Digital Output (0V low/3.3V high)

Figure 26: Specification table showing the inputs and outputs of the infrared receiver circuit

Initial Ideas

First, we had to choose which receiver sensor to use to convert the light energy from the IR pulses into electrical energy to be interpreted by the software. Upon further research, we found these generalised into two categories: photodiodes or phototransistors. Photodiodes normally have a quicker response time than phototransistors, a lower capacitance and do not require an external power source to operate but as phototransistors have a higher gain than photodiodes due to their beta value they are more suited to our application where we need to detect specific wavelengths of light in conditions where there is lots of background light.

We chose to use OP593B phototransistor due to its low cost of 60p and its high receptiveness to 950nm wavelength light as shown in *Figure 27*.

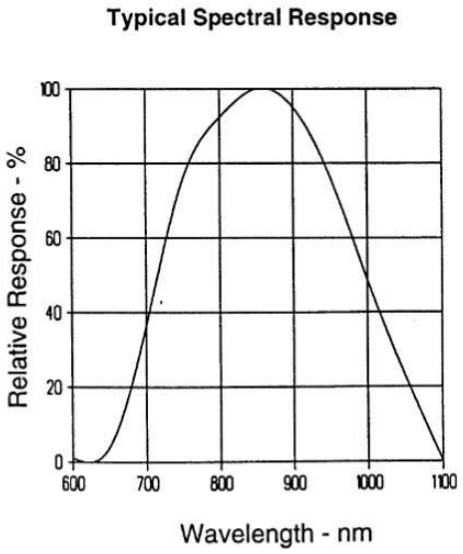


Figure 27: Graph showing the typical spectral response of the OP593B phototransistor[3]

Our initial design is shown in *Figure 28* below; a single phototransistor is used to detect the signal. We then used a non-inverting amplifier with a gain of 5 to amplify the signal making the spikes clearer for the comparator which then creates a digital output to be interpreted by the microcontroller.

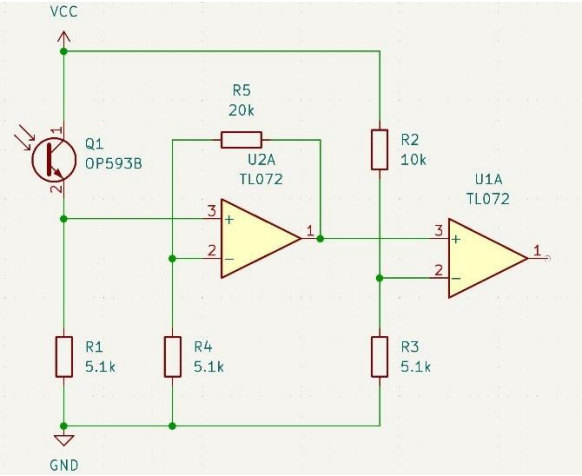


Figure 28: Initial design of infrared receiver

This design has a significant flaw; the gain of the non-inverting amplifier must be set such that the op amp does not saturate due to background light. This would require a different gain to optimise the amplifier for every change in background light.

[3] 20:45 09/06/2024: 'OP593B Datasheet' - https://docs.rs-online.com/1c44/0900766b814b5188.pdf

Final Design

So that the circuit would work consistently in varying light conditions, we chose to use a differential amplifier with two IR sensors configured in a bridge circuit. One phototransistor will receive both the signal and background light, and one will receive only background light. The output of this circuit is then the difference between the voltage from each sensor, which is then amplified by the differential amplifier removing any common mode input and increasing the amplitude of the differential signal. This represents a significant improvement in the useability and real-world applicability of the subsystem, as it will be much more able to cope with variable light conditions, as excess background light could cause the original non-inverting amplifier to saturate giving a false high.

A differential amplifier with a gain of 20 was configured on a single supply to amplify the signal. One phototransistor is to be placed at the front of the buggy to receive the signal and background light, and one in the middle to measure background light. A comparator is then used to create a digital output, before the BJT creates a rail-to-rail digital output. By using the inverting input to detect the signal and background, the output of the differential amplifier is inverted, so the BJT corrects this.

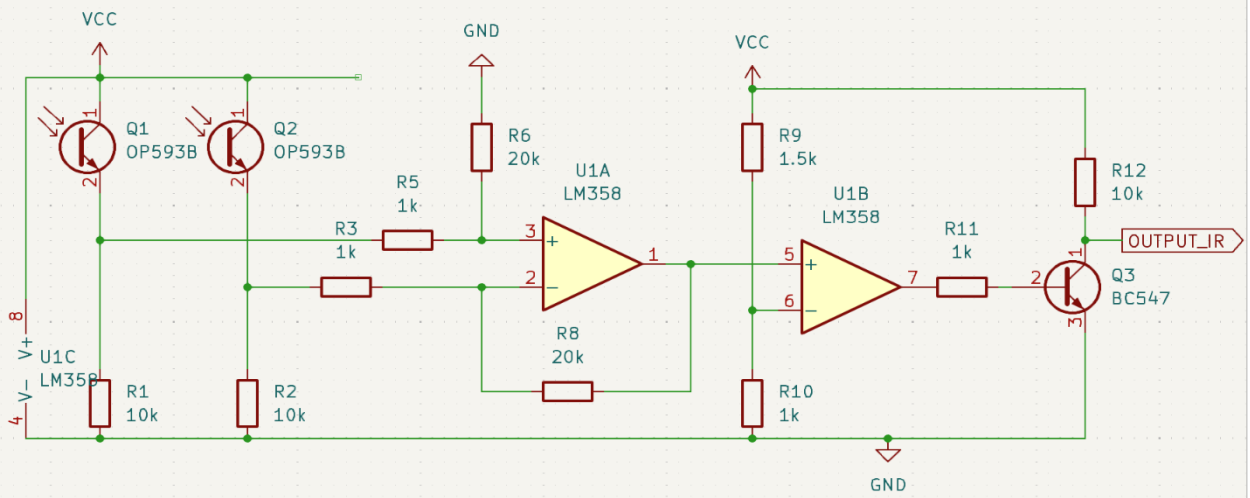


Figure 2129: Final design of the infrared receiver

$$\frac{V_{\text{out}}}{V_{\text{diff}}} = \frac{R_f}{R_1}$$

$$G = \frac{20k}{1k}$$

$$G = 20$$

Figure 3013: Calculations for the differential amplifier gain of the IR receiver

$$V_{\text{out}} = V_{\text{in}} \frac{R_2}{R_1 + R_2}$$
$$V_{\text{out}} = 3 \cdot 3 \frac{1}{1.5 + 1}$$
$$V_{\text{out}} = 1.32V$$

Figure 3114: Calculations for the IR receiver's potential divider

The phototransistors we used were cheap off the shelf components, and as a result they were not well matched. This meant we had to use different resistor values to ensure we matched the output in identical light conditions. This led us to swap R8 to a 100k and R9 to a 1k to achieve this, allowing the bridge circuit to be used with the differential amplifier.

Testing

With the test lizard transmitting the infrared signal from a distance of 6cm and slowly moving away until the signal is no longer received correctly, the picoscope should be set up to measure the output of the differential amplifier (blue) and the output of the subsystem (red).

Results

The picoscope trace shows a result of 569.2Hz which is close to the specified 571Hz (and within ±10Hz tolerance that will be built into the software due to imperfections in the lizard's construction). It was

received up to a distance of 20cm. The high voltage is 3.3V and the low voltage is 0V, which meets the high and low voltage ranges required for the microcontroller.

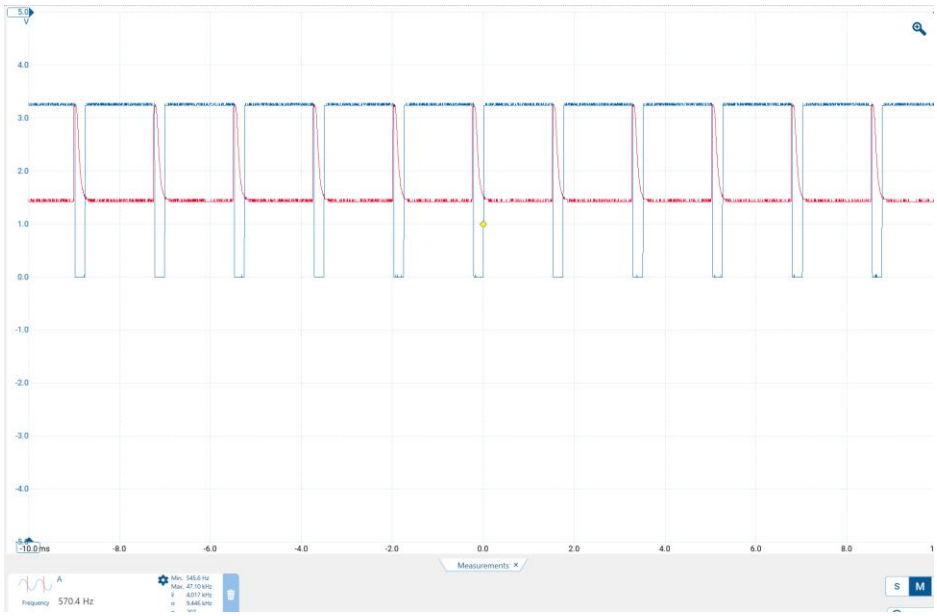


Figure 3215: Picoscope trace showing output of the differential amplifier in red and the subsystem output in blue

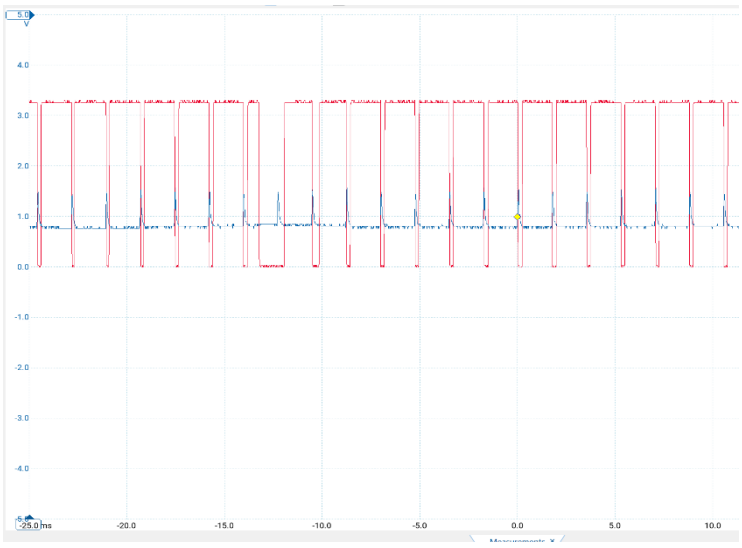


Figure 3316: Picoscope trace showing output of the phototransistor in blue and the subsystem output in red

Evaluation

This subsystem worked well, from a range much greater than the minimum specified. This was essential, as when the IR is transmitted through the skin of the lizard, the signal amplitude and therefore the

detectable distance is significantly reduced. In the future, using a rail-to-rail op amp would help improve the subsystem by reducing the number of components required for the subsystem.

Magnetic Receiver

Overview

The magnetic receiver subsystem should provide an analogue input to the microcontroller to allow the static magnetic pole input to be decoded to help determine the species of the lizard. The voltage output should increase or decrease from a standing point to differentiate between South or North poles. The lizard's static magnetic field comes from a small magnet placed horizontally on the roof of the lizard's mouth so the subsystem should be able to measure the magnetic polarity from at least 6cm away.

Inputs	Outputs
Static magnetic field from a distance of 6cm	Analogue Output which increases or decreases depending on the polarity of the static magnetic field

Figure 3417: Specification table showing the inputs and outputs of the infrared receiver circuit

Figure 3518: Placement of magnet in the lizard's mouth

Choosing a Sensor

The key factors we considered when designing the magnetic sensor subsystem were the low magnet strength and therefore short measuring distance, the magnet orientation and how we could measure its field despite this and any external magnetic fields which might interfere with our signal.

Low Magnet Strength: The lizard's magnets were very small and limited in strength. As a result, the magnetic field we needed to detect was only sizeable in strength at close proximities, meaning the sensor had to be placed close to the lizard. Amplification of the sensor output was necessary to increase the distance at which a sizeable enough change in voltage could be detected and therefore allow the polarity of the magnetic field to be determined.

Magnet Orientation: The magnets were placed horizontally in the roof of the lizard's mouths, meaning the sensor had to measure the magnetic field coming from above or below such that the magnetic flux lines intersect the sensor perpendicularly.

External Magnetic Fields: With several other lizards and rovers from other teams being in the arena simultaneously, the level of magnetic and electrical interference was going to be considerable on testing day, meaning that the sensor circuit had to be designed in such a way to reduce external magnetic fields

Initial findings quickly revealed that two types of sensors would come into question for this application: Hall Effect and Magnetoresistive sensors. Extensive research led to the choice of Hall Effect sensors primarily due to their lower cost and easier implementation for our specific purpose: solely to detect the polarity of the magnetic field.

The Hall Effect occurs when a magnetic field is applied perpendicular to the flow of electric current in a semiconductor, causing the Hall voltage to be generated perpendicular to both the current and the magnetic field. This phenomenon results from the Lorentz force acting on the moving charge carriers, pushing them to one side of the material, creating a potential difference.

In Hall Effect sensors, this principle is utilized to detect the presence and magnitude of a magnetic field. The sensor consists of a thin strip of semiconductor material through which a constant current flows. When a magnetic field is present, the Hall voltage generated across the material is proportional to the strength of the magnetic field.

As only the polarity of the magnetic field is to be detected, research showed that the most effective way of identifying this would be to utilize a linear sensor which has an output voltage proportional to the detected magnetic field strength, and then identifying whether the output voltage is above or below the threshold voltage to determine the magnetic field as either North or South.

Sensor Choice

After comparing several sensors, we decided to opt for the AH49F Linear Hall Effect Sensor as this matched all our specifications, such as an operating voltage of 3.3V matching the logic high level of our microcontroller and a sensitivity appropriate for the low magnetic field strengths at 2.1mV/Gauss.[4]

[4] 11/06/2024, 22:52: 'AH49F Datasheet' - https://www.diodes.com/assets/Datasheets/AH49F.pdf

V_{SEN}	Output Voltage Sensitivity	B = 0 to ±600 (Gauss)	1.7	2.1	2.5	mV/Gauss

Figure 3619: Extract from AH49F Linear Hall Effect Sensor datasheet

Initial Ideas

Initially, we designed a simple circuit based off a single AH49F sensor which was connected to a non-inverting amplifier with a sufficient gain of 21 to amplify the sensor's output signal so the detection range could be increased. This circuit design functioned as expected and allowed us to correctly determine the polarity of the magnetic field at ~6cm.

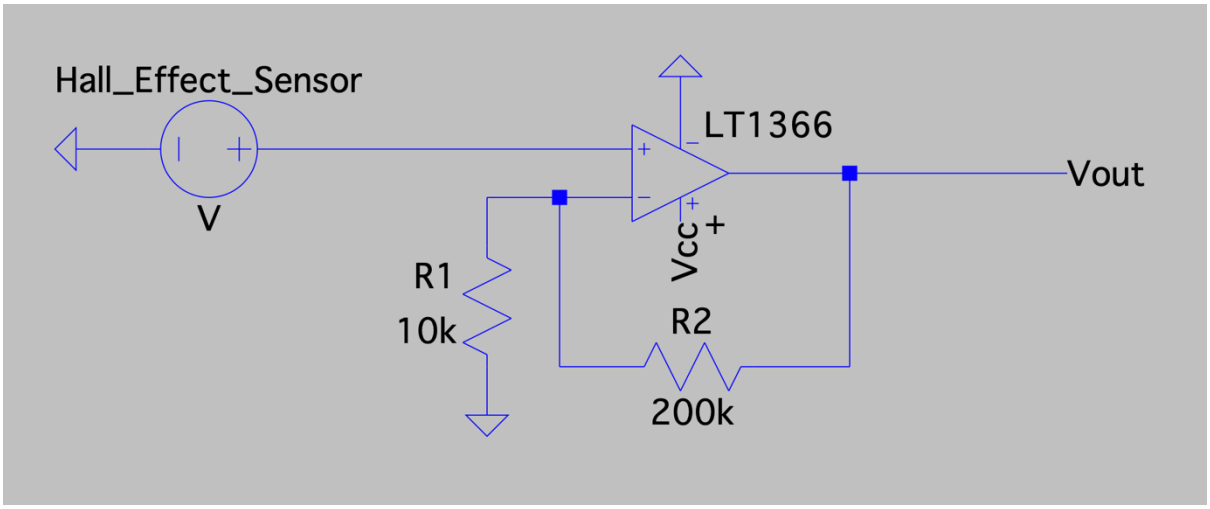


Figure 3720: Initial design of the magnetic receiver

However as per our specification, there could be a considerable amount of external magnetic field influence so to remove the potential of this becoming an issue on testing day, we decided to explore an alternative two-sensor design.

Final Design

Further research showed implementing a second identical AH49F sensor and connecting these to a differential amplifier would isolate the lizard's magnetic field from any external field influence. This can be achieved if one 'secondary' sensor is placed at the base or centre of the rover, whilst the 'primary' sensor is placed close to the lizard. By doing this, the output of the differential amplifier would yield solely the strength of the lizard's magnetic field regardless of external influence.

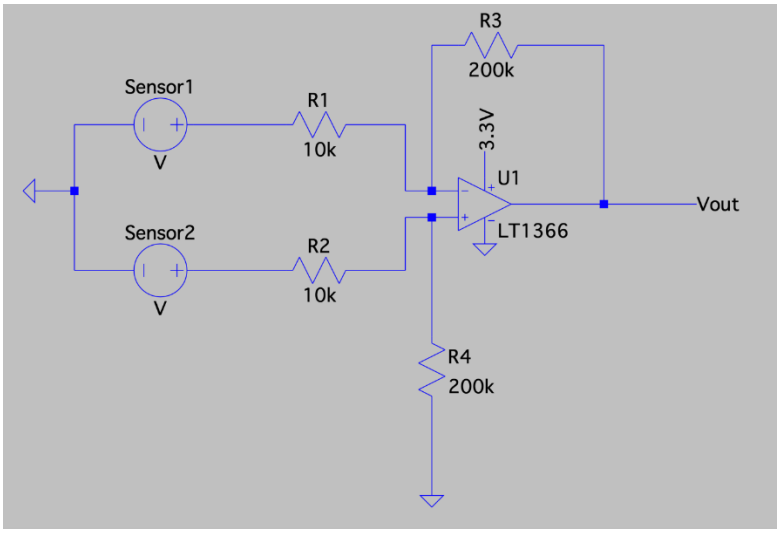


Figure 38: Second magnetic receiver design

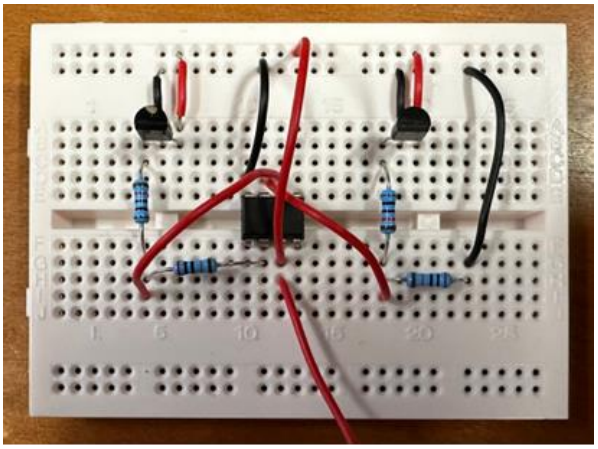


Figure 3921: Breadboard implementation of the second magnetic receiver design

Since the Arduino could not show negative values when a negative magnetic field was detected, we needed to add an offset. To provide equal voltage swing in both directions, the offset was set to half the supply voltage by adding the potential divider seen at the bottom with two 10k resistors (R30 and R33). This yielded the final circuit design, which had now also been optimized for space efficiency:

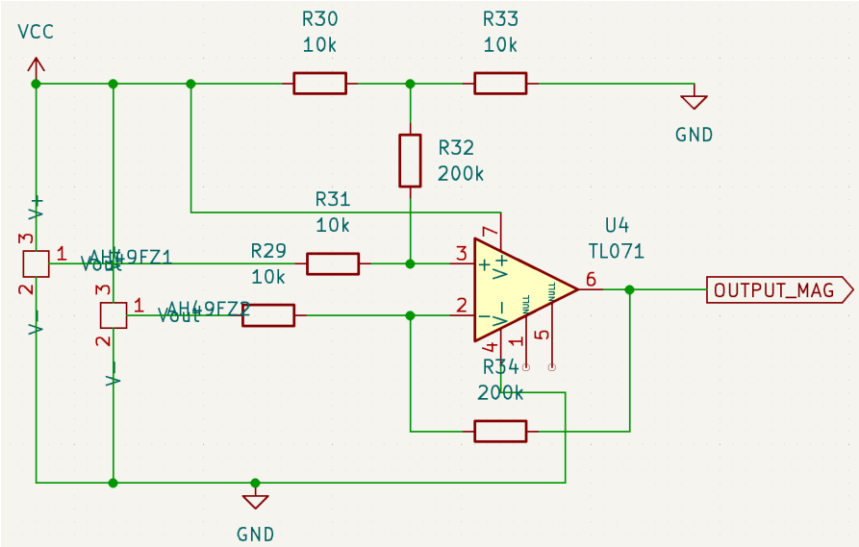


Figure 40: Final magnetic receiver design

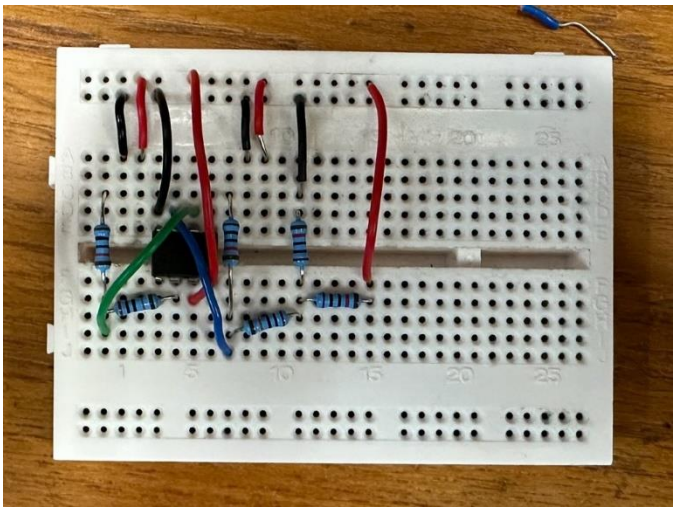


Figure 4122: Breadboard implementation of final magnetic receiver design

Testing

Analogue testing of the magnetic circuit was conducted using a voltmeter as shown below. We tested the sensor circuit by placing the magnet next to the sensor and moving it away until it could no longer be detected. When no magnetic field was present, the circuit operated at its bias voltage of approximately 2V. In the left photo, a South Pole was detected as the output voltage of the circuit increases above the bias voltage. The opposite was true for the right photo, where a North Pole is detected. The testing coincides with the manufacturer's specification of the sensor and reveals that the circuit operates as intended.

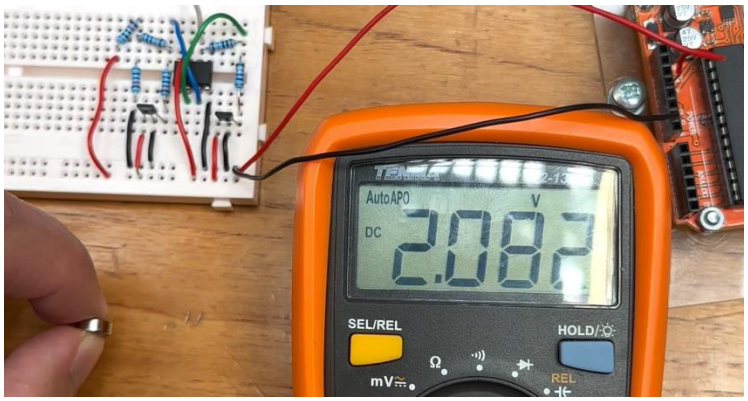


Figure 42: Testing showing no magnetic field is present

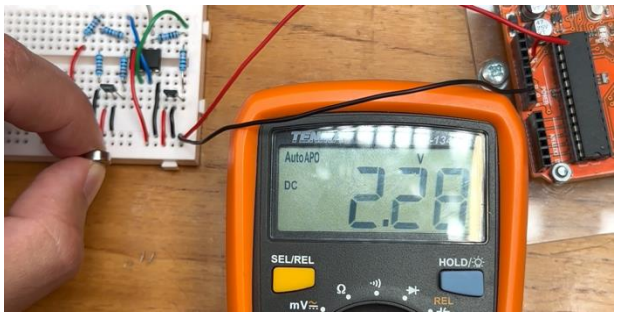


Figure 43: Testing showing a South Pole is detected

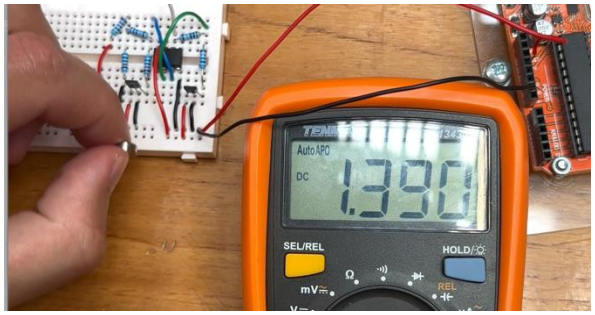


Figure 4423: Testing showing a North Pole is detected

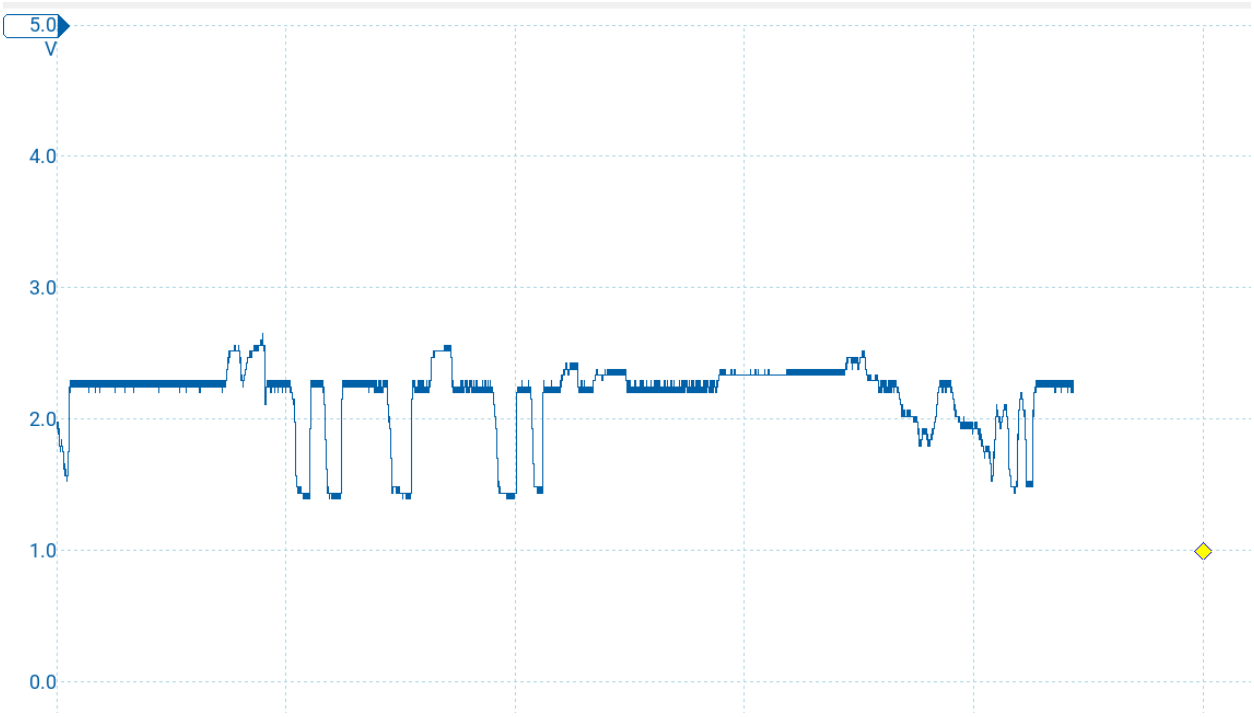


Figure 45: Picoscope trace showing the magnetic pole readings

Evaluation

This subsystem worked well, meeting the specification. In the future, it could be improved by increasing the range by using better sensors however this proved difficult due to the weakness of the magnet in the lizard. Therefore, it was decided this was too expensive for the little reward. This meant sensor placement for the magnetic sensors was crucial, which could potentially be a weakness of our design.

PCB Design

Originally, we optimised our circuits such that they occupied only 2 breadboards, however with the increased space this would require on a chassis we decided that it would be too heavy. Therefore, we chose to use a custom PCB design. To reduce the cost of the PCB, we chose to have a separate chassis. This allowed us to utilise the original EEEBug PCB for the motors and voltage regulation to 5V, and reduced wastage as we did not have to re-print this section into our new PCB. The PCB also allows for a clearer more modulated design by separating the way the circuits are grouped on the board, but with simplified common power connections. It clearly marks the outputs and power supply connections using the connectors helping to improve the usability of the rover and presents a significantly more durable solution than prototyping breadboards.

We designed the PCB to use 2 layers, one with wider 0.4mm tracks for the power connections to allow for easier current flow and another with thinner 0.2mm tracks for the connections between components to

optimise space and allow more intricate connections. This allowed us to reduce the size of the circuitry from 2 breadboards to a PCB which was 82.50x67.00mm (a size smaller than 1 breadboard), significantly reducing the weight of the buggy. This also allowed the design of a new lighter chassis to further reduce weight.

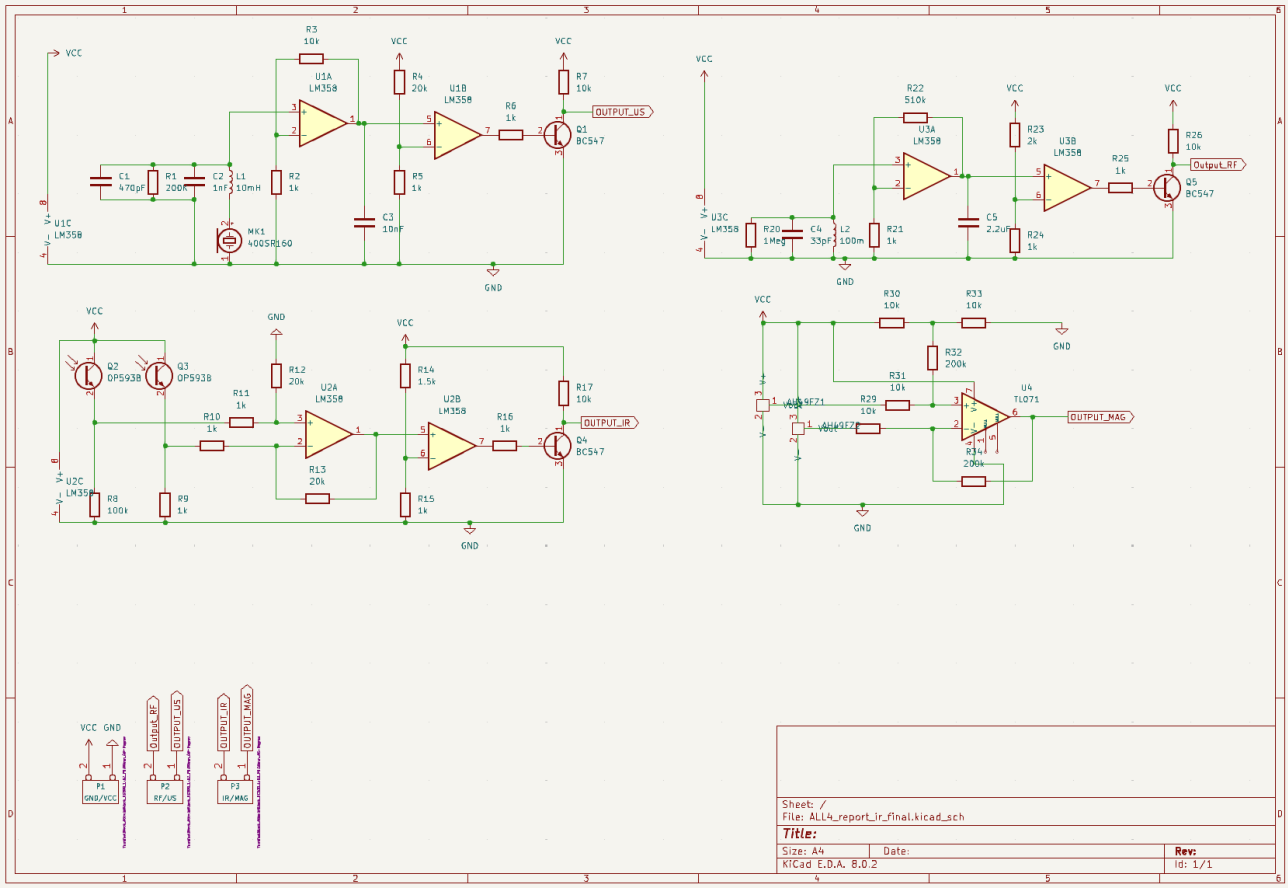


Figure 46: PCB Schematic

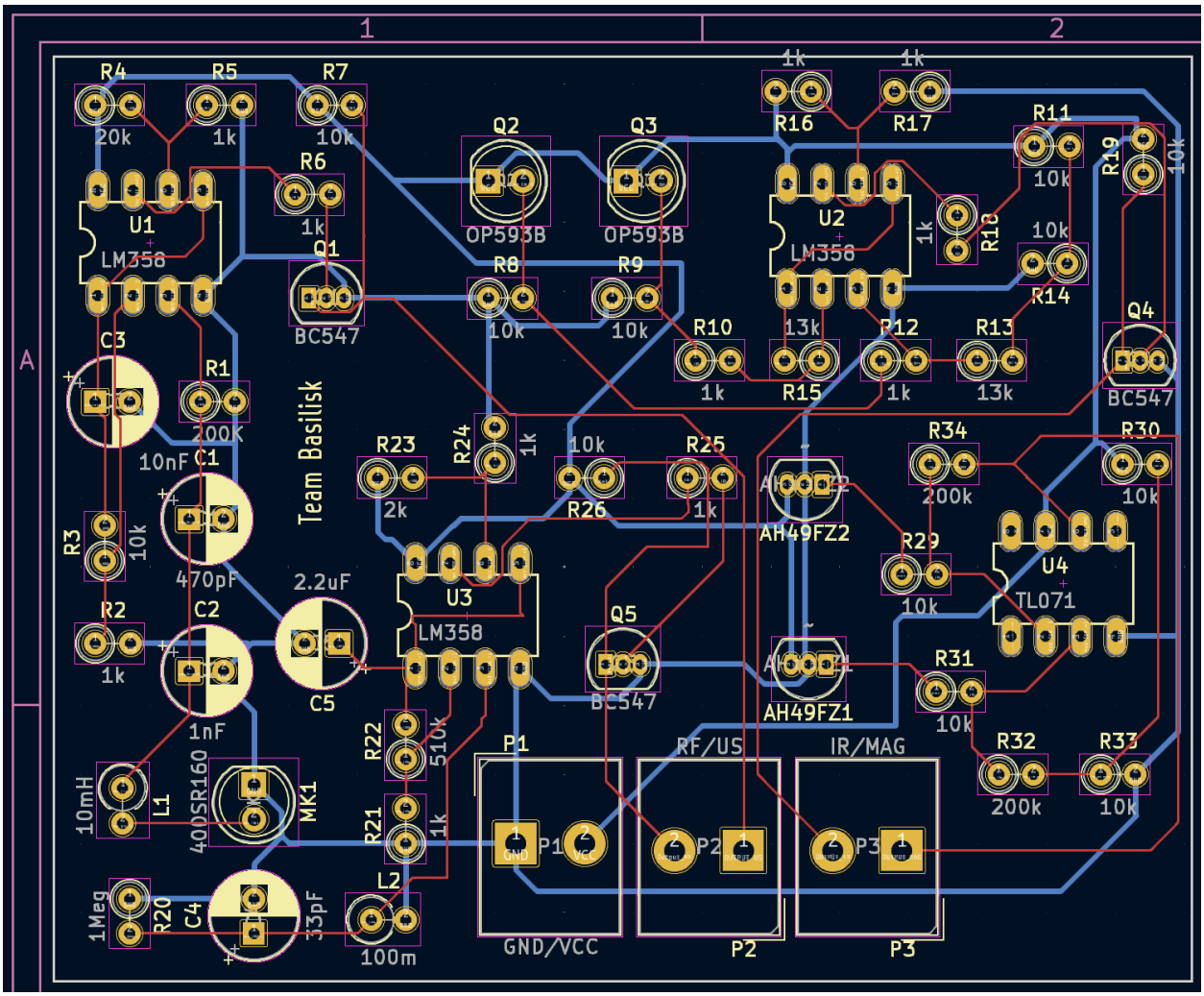


Figure 4724: PCB design

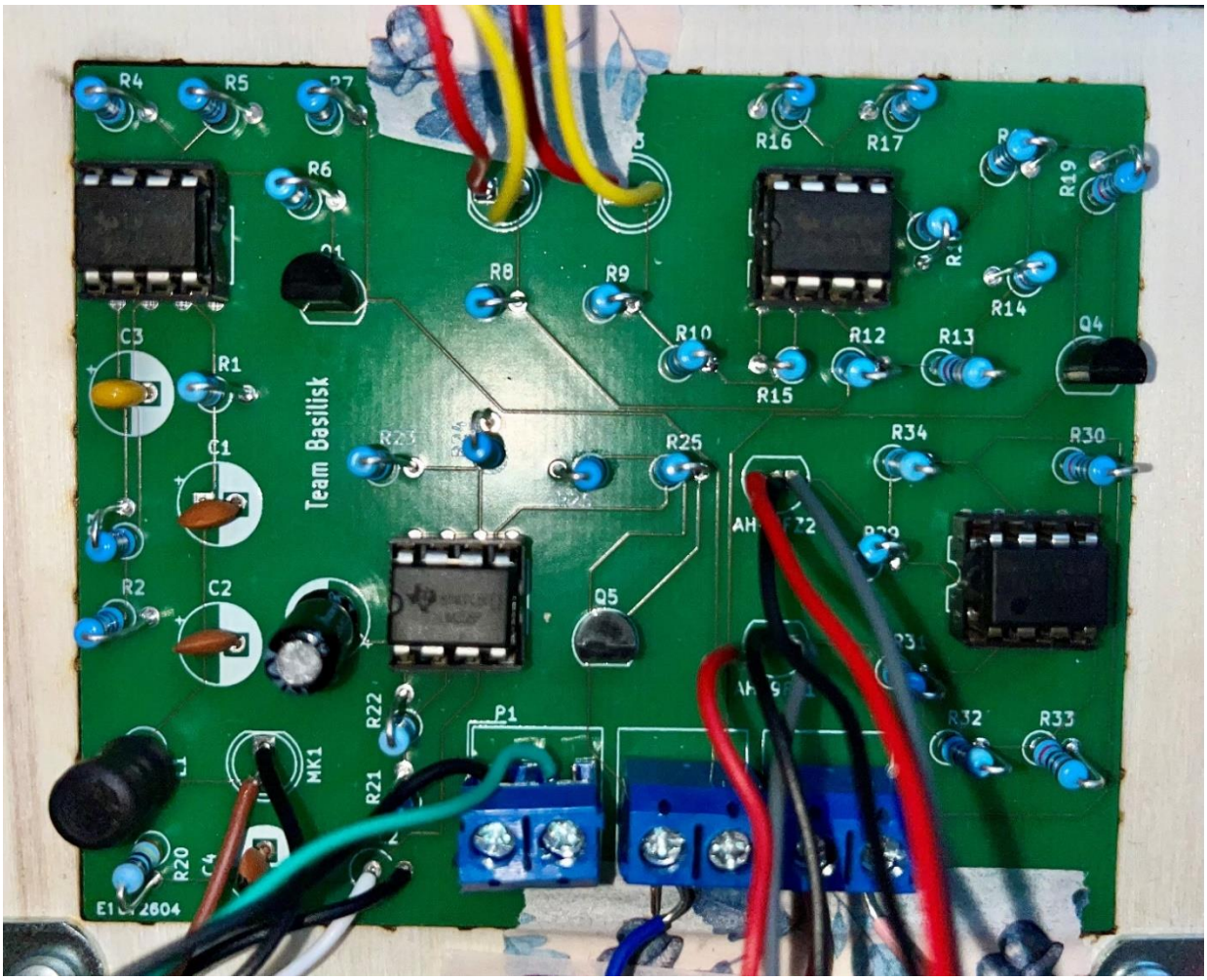


Figure 48: PCB design with the components soldered onto it

Final Construction

Originally, we considered using the EEEBug chassis, however after the PCB was chosen, we realised the EEEBug chassis was incompatible with our new PCB. The placement of the microcontroller was also too big for the ESP32 we chose to use instead of the Metro M0. Therefore, we chose to design our own custom chassis. Firstly, we chose to 3D print a new chassis and designed a CAD model in *Figure 49* for a chassis. However after considering the material it would have to be printed from, we decided a better solution would be to laser cut two sheets of poplar wood. Poplar wood was chosen as it is exceptionally light, but it has considerable strength compared to its weight. We cut two sheets, one with cutouts for the motors and PCBs and one with just cutouts for the motors, such that when the two were arranged on top of each other the holes for the motors went completely through the chassis and the sections for the PCB had cutouts for the solder joints. This enabled us to reduce the weight of the buggy significantly, while maintaining its durability and improving the housing of our components.

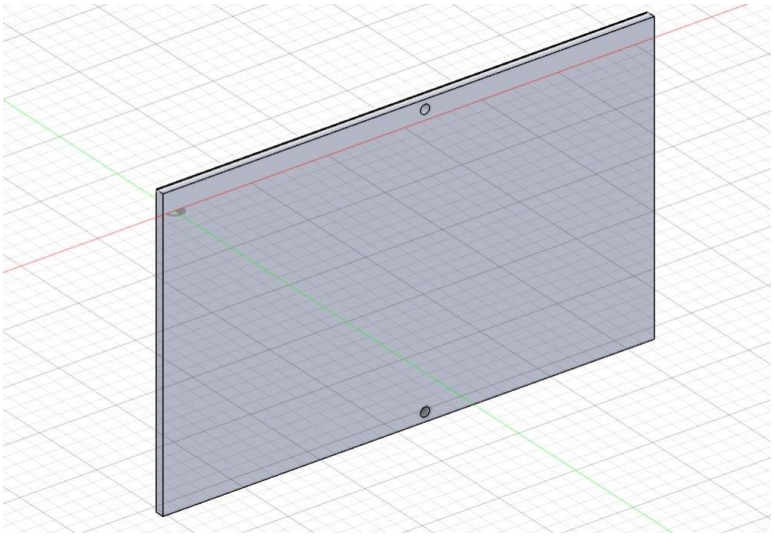


Figure 4925: Initial CAD design of the chassis

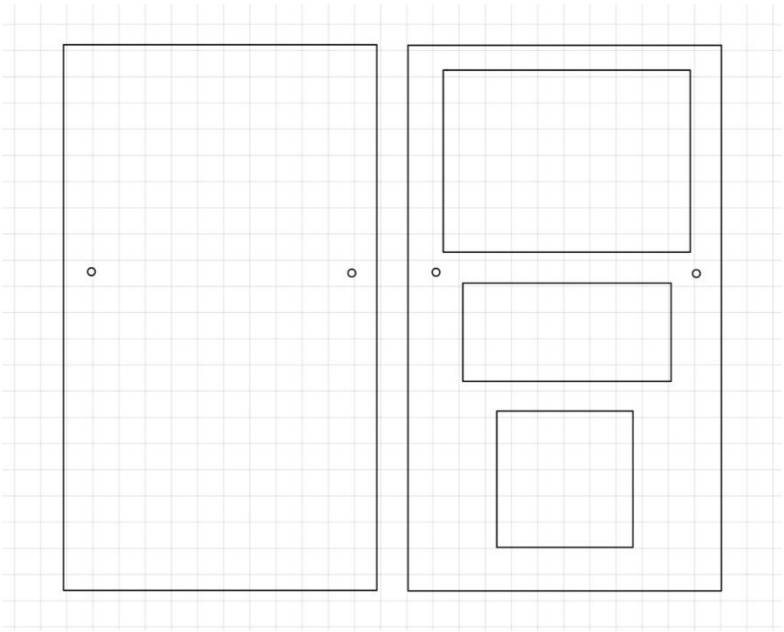


Figure 5026: 2D laser-cut design

Wheels

Overview

To increase the speed and handling precision of our rover, we decided to design new wheels. Due to the time constraints of the final assessment, we needed wheels which were able to help the rover maneuverer the terrain quicker, more precisely, and more reliably so that the user can easily, quickly and accurately control the rover when surveying the lizards. After extensively testing the current wheel designs given in the EEEBug project alongside further research, we determined the new wheels should be:

Lighter: considering that the rover must maintain a weight ≤750g, and that a further reduction in weight would increase its speed.

Larger & Wider: the current wheels from the EEEBug are very small in diameter. This means they could potentially struggle with obstacles such as plants or bumps in the terrain. Increasing the wheel diameter would reduce the risk of the rover getting stuck. Furthermore, the contact patch/tread of the current wheels is extremely narrow, meaning that the wheels easily begin to slip and lose traction. Increasing the width of the wheel (specifically the width of the tread) would remove this issue.

More Secure Rubber Element**:** the rubber ring that is currently stretched across the wheel rim is not secured in any other way, for example it is not glued to the rim. As a result, it easily comes loose from the wheel especially when the rover is moving which further hinders the rover's movement.

Initial Ideas

Our immediate thoughts to reduce the weight of the current wheels was to change the material and design. The current wheels from the EEEBug were made from dense plastic and are completely filled; they had no spoke design which would reduce weight. By utilising 3D-Printed wheels, we would be able to design to our specifications and make use of the light product weight. Several designs were initially tested, such as the one shown below, featuring a larger diameter (70mm) and a much wider tread (30mm), while also being significantly lighter due to the 3D-Printed material:

Figure 5127: CAD model of our designed wheel from the front

Figure 52: CAD model of our designed wheel from the back

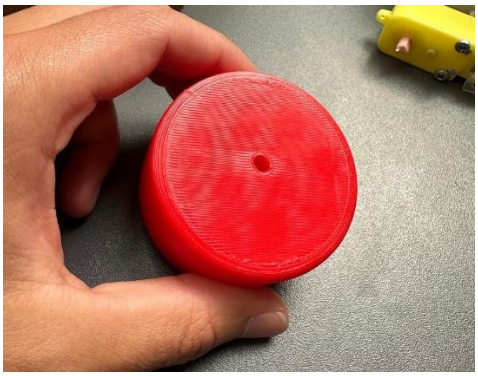


Figure 5328: 3D printed wheel

However, this wheel design required an additional rubber attachment which was to be the tread, as the 3D-Printer filament was very slippery and thus provided very little surface traction. As we were unable to find such a suitable item in the labs or from one of the available order stores, we had to resort to alternative designs.

We also investigated other wheel designs such as the Mecanum wheels shown below in *Figure 54*, which would allow the rover to move omnidirectionally. The slanted design of the rollers allowed the rover to do basic movements such as move forwards or backwards however also allowed the rover to move directly sideways and diagonally (strafe) without having to turn the rover in that direction first therefore allowing the user to move towards the lizards they want to survey more quickly. Upon further testing however, we found most people found it hard to control due to the number of different available controls and it took time to master how to move the rover which detracted away from the useability of the rover. Additionally, to be able to successfully use Mecanum wheels in a way which allowed us to access these advantages we had to use a minimum of four wheels which would significantly add to the weight of the rover which we had to keep under 750g. This became an especially significant problem as the Mecanum wheels themselves were heavier than our original wheels which also added to the weight.

Figure 5429: CAD model of the Mecanum wheel[5]

Final Design

As a result, we decided to order a new set of manufactured wheels which fulfilled our specifications of being larger, lighter and more secure/reliable. We opted for the MC02755 Wheels, as they had an increased diameter of 70mm, as well as a significantly increased tread width of 25mm and a more robust and secure rubber element which would heavily improve traction. Because of their spoke design, the wheels remain similar in weight to the original ones even with their much larger size. Therefore, they fulfil all our design specifications.

Figure 55: Chosen wheel design[6]

[5] "Mecanum wheel" - https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.pinterest.com%2Fpin%2F372039619199299604%2F&psig=AOvVaw3NensBNN1tCw-0y5bxQqeq&ust=1718396809788000&source=images&cd=vfe&opi=89978449&ved=0CBQQjhxqFwoTCIj114612YYDFQAAAAAdAAAAABAE

[6] 'MC02755 Wheel' - https://onecall.farnell.com/unbranded/mc02755/wheeltyre-70-x-25mm/dp/MC02755

Sensor Arm

Overview

As most of the sensors, especially infrared and magnetic, have a small operating range, they must be placed close to the lizard to be able to correctly identify the species. To facilitate this, a 'sensor arm' was designed which would allow the sensors to protrude from the main chassis and get very close to the lizard.

Initial Ideas

Initially, we designed a simple, thin arm construction on which the sensors would be placed. This was thin enough to eliminate as much weight as possible therefore not tipping our rover over with its weight and ensuring our rover stayed within the 750g weight limit.

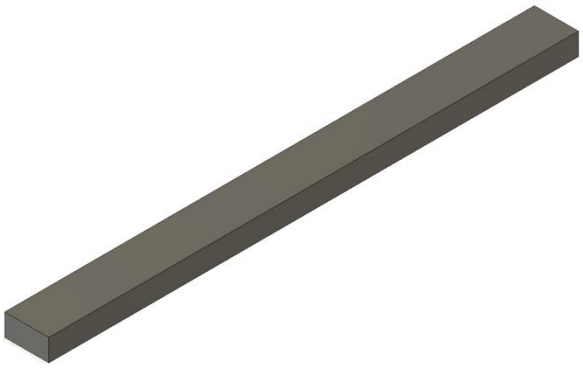


Figure 5630: CAD model of initial design of sensor arm

However, upon considering the sensor's orientation, we realized while ultrasound and magnetic must be level with the lizard's mouth, IR and radio must be located above the centre of the lizard's body. This was because while the radio frequencies and magnetic flux lines originated from in the lizard's mouth, the lizard's IR and magnetic transmitters are located further back on the lizard PCB and emit vertically unlike the magnetic and ultrasound waves which were emitted horizontally.

This led us to consider a 'Z' arm design, where ultrasound and magnetic would be attached to the top part, located below infrared and radio.

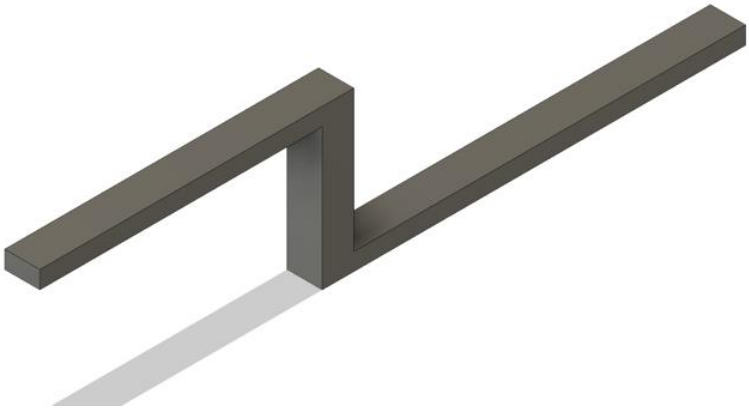


Figure 5731: Initial 'Z' sensor arm design

Final Design

With the addition of a 'platform' for the former two, we finalized our sensor arm design as shown below in *Figure 58*. The addition of this element allows us to position each of the sensors optimally such that the distances between sensor and transmitted are minimized, and that the orientation is optimized for each individual sensor.

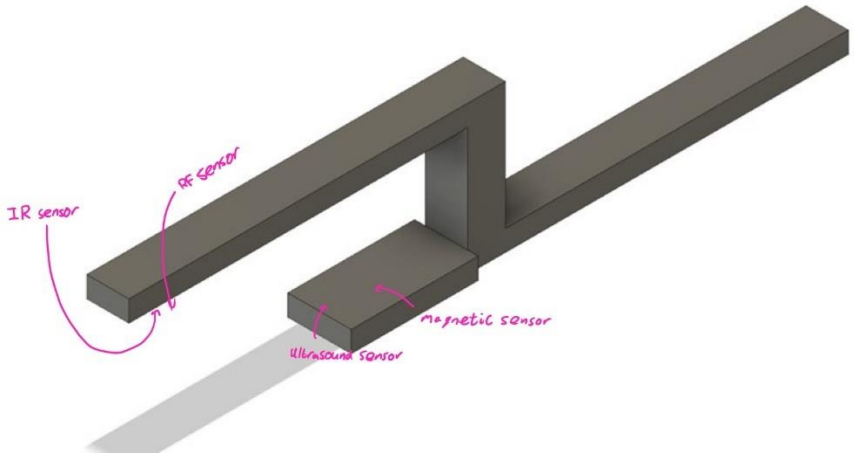


Figure 5832: Final sensor arm design

Full Assembly

After creating the relevant parts, we assembled the rover using screws, nuts and hot glue to quickly but securely attach components. To ensure the motors and therefore wheels would not move, we added black pieces of acrylic fitted to the rover to act as spacers. We also added white stabilisers as can be seen in *Figure 59* below. All materials required to build the chassis were found in the scrap box in robotics which we then upcycled for our project.

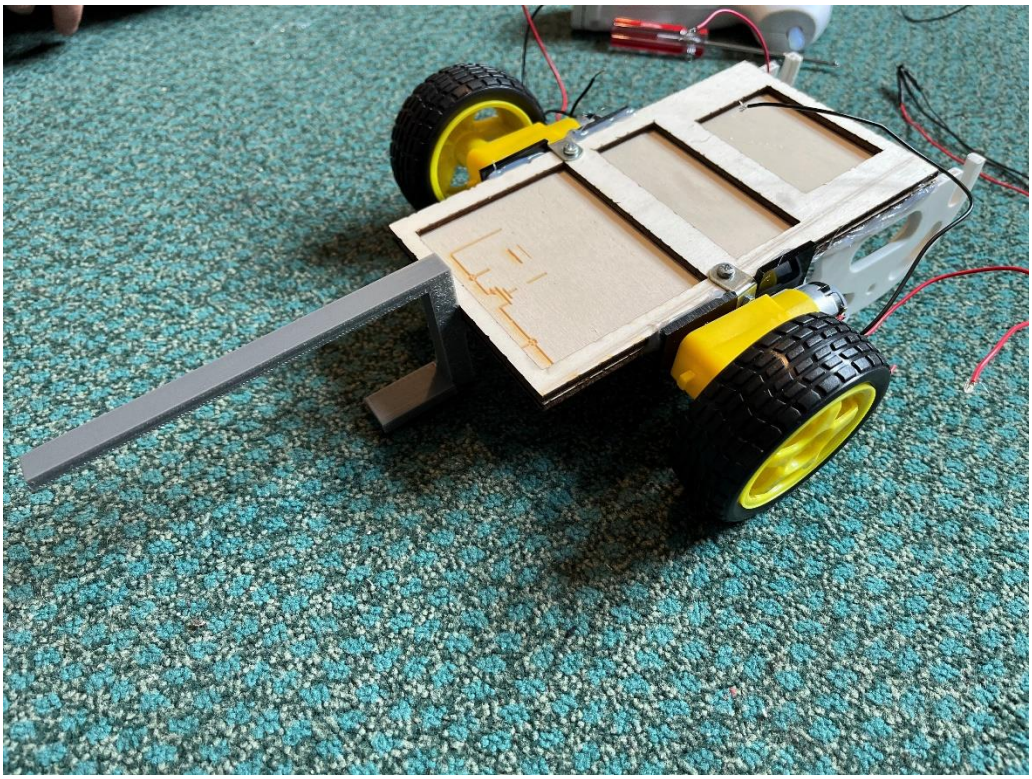


Figure 5933: Full assembly of the chassis, sensor arm and wheels

Control and Programming

Overview

The control and programming section should provide a user-friendly webpage the client can interact with to control the rover's movement and sensors. This subsystem should automatically interpret the output of the sensors into a user-friendly form and display it on the webpage when the user presses the scan button. This should be easy for the user to interpret, for an example the output of the ultrasound receiver should be the lizard's name in ASCII rather than the displayed waveform which most users would not be able to understand. The inputs of the system should be user input through the webpage and sensor outputs from the rover. The outputs of the system should be the displayed information about the lizards on the webpage and the movement of the rover.

Function Origin	Input	Output
Radio Receiver	Digital square wave input of a particular frequency (0V low / 3.3V high)	Frequency of the square wave and whether this is close to 120Hz or 200Hz
Ultrasound Receiver	Inverted digital input of UART packets representing the characters or the lizard's name (0V low/3.3V high)	4 character lizard name starting with a #
Infrared Receiver	Digital pulse input of a particular frequency (0V low / 3.3V high)	Frequency of the pulse and whether this is close to 571Hz or 353Hz
Magnetic Receiver	Analogue input which increases or decreases depending on the presence of a magnetic pole	Type of pole detected is displaced
Webpage movement buttons	Directional movement buttons pressed on the webpage	Motors drive in the correct direction
Webpage scan button	Scan button pressed on the webpage	Calls all the sensor functions and displays the results on the webpage

Figure 6034: Function specification table for the rover

Initial Ideas

As specified in the overview specification, the webpage had to control the rover remotely while receiving and interpreting data from its sensors. Initially we designed the motor control with basic webpage functionality to verify we were able to control the rover's movement remotely. After analysing the starter code and understanding how HTTP requests worked, we created buttons to allow the user to control the rover as they were easy to implement and allowed us to test each movement independently. We uploaded our code to the Metro M0 board and verified that it would move with user input through our buttons.

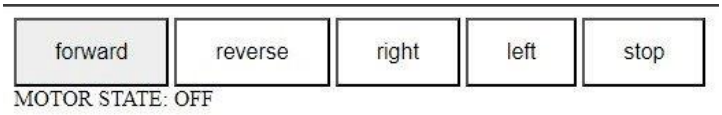


Figure 6135: Initial HTML build

Next, we quickly generated different ideas for control schemes focusing on usability and ease-of-use. We found the buttons made it difficult for the user to quickly change the rover's direction as the user had to move their mouse in between each button and click which caused a considerable time delay. As a result, we settled on two main ideas for the user to control movement: a joystick or a gamepad-like UI with keyboard functionality to eliminate the delay from the mouse moving. After further research, we settled on the joystick as people found it easier to control and visualise how the rover was going to move in response to the joystick input.

Joystick Design

Initially, we planned to allow the user to control two features with the joystick: the rover's direction by the angle of the joystick and the rover's speed by the level at which the joystick was pulled up. When the user drops the joystick, it returns to the centre. We displayed the X and Y co-ordinates alongside the angle above the joystick to test its functionality and allow the user to have both a visual and number representation of their input.

The inner working of the joystick is split into 3 main sections: the HTML structure and CSS styling, JavaScript functionality and the PHP Backend and Arduino code:

HTML Structure and CSS Styling: This defined the canvas element which is primarily how the user would interact with the joystick allowing them to control the rover. The user's movements on the canvas would be captured and processed to determine the joystick direction and position which are then displayed as the joystick's X and Y coordinates controlling the speed and direction of the rover. We also define a scan button which would allow the user to activate the sensors and display an output in the table.

JavaScript Functionality: The movement of the joystick works by the JavaScript code constantly redrawing the circle on the canvas each time the user interacts with it through the Draw() function which is called whenever the user clicks their mouse on the canvas. The JavaScript code also calculates the joystick position, displaying it and the motor state changes in the Document Object Model while ensuring the joystick never leaves its movable area ('is_it_in_the_circle'). It then sends the computed motor state to the server.

PHP Back-end and Arduino Code: The PHP back-end then receives the motor state via GET request and sends the relevant commands to microcontroller on the rover by constructing a URL with the state parameter which is then accessed via a URL request. The Arduino code defines what the motors should do in response to the URL, each function triggering the digital and pulse width modulation (PWM) outputs to control the motor's speed and direction. It also initialises the server and allows the rover to continuously handle incoming requests to adjust the motor outputs based on the received commands.

```
function Draw(event) {
    if (paint) {
        getPosition(event);
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        background();
        var angle_in_degrees, x, y;
        var angle = Math.atan2((coord.y - y_orig), (coord.x - x_orig));

        if (Math.sign(angle) == -1) {
            angle_in_degrees = Math.round(-angle * 180 / Math.PI);
        } else {
            angle_in_degrees = Math.round(360 - angle * 180 / Math.PI);
        }

        if (is_it_in_the_circle()) {
            joystick(coord.x, coord.y);
            x = coord.x;
            y = coord.y;
        } else {
            x = radius * Math.cos(angle) + x_orig;
            y = radius * Math.sin(angle) + y_orig;
            joystick(x, y);
        }

        var x_relative = Math.round(x - x_orig);
        var y_relative = Math.round(y_orig - y);

        document.getElementById("x_coordinate").innerText = x_relative;
        document.getElementById("y_coordinate").innerText = y_relative;
        document.getElementById("angle").innerText = angle_in_degrees;

        updateMotorState(angle_in_degrees);
    }
}
```

Figure 62: draw function of the joystick

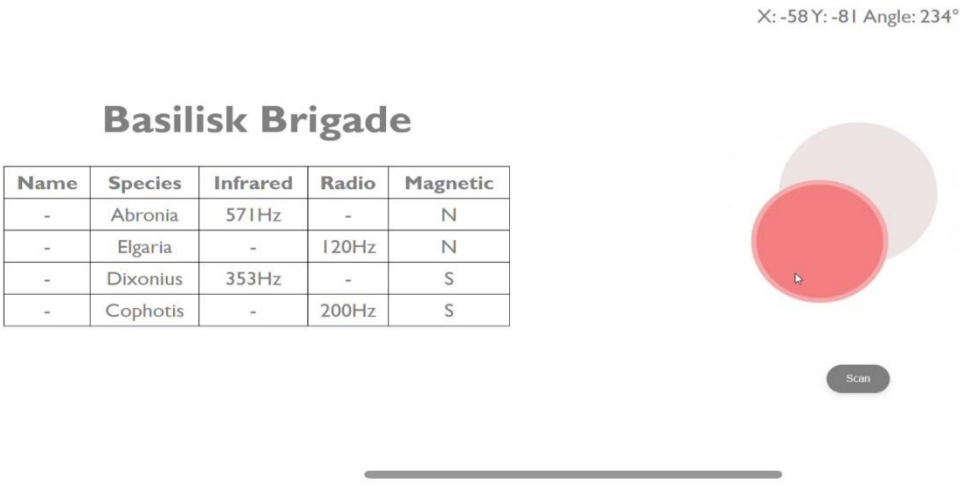


Figure 63: Initial build showing the joystick movement and database

Motor Control

Each motor has a PWM and direction control (DIR) pin to control its speed and direction. The PWM pin uses an analogue input from 0 to 255 while the DIR pin receives a digital input. A logic table showing how these pins control the rover's movements is shown below in *Figure 64*.

DIR	PWM	A	B	C	D	Motor
x	0	V_m	V_m	V_m	V_m	off
0	1	0V	V_m	0V	V_m	forwards
1	1	V_m	0V	V_m	0V	reverse

Figure 6436: logic table for the motor pins

After initially gradually turning based on the user's current position to create smoother movements, we decided to create simpler rover turning functions as we found turning based on the user's current position was slow and given the timing constraints given in our specification, this would be undesirable.

```
//left
void left() {
    digitalWrite(in_1, HIGH);
    digitalWrite(in_2, LOW);
    analogWrite(analogOutPin, 255); //initially set PWM to 255 for testing
    server.send(200, F("text/plain"), F("left"));
}

//right
void right() {
    digitalWrite(in_1, LOW);
    digitalWrite(in_2, HIGH);
    analogWrite(analogOutPin, 255);
    server.send(200, F("text/plain"), F("right"));
}
```

Figure 65: code snippet showing how we initially tested how the rover moved left and right

Variable Speed and Keyboard Functionality

After initially testing the rover, we decided to add complete variable speed from a range of 'Speed: 1' to 'Speed: 100' controlled by how far the user pulled the joystick. This was controlled by the PWM signal as

specified above. This allowed the user to quickly accelerate towards the lizard to cover a greater distance quicker but also slow down so that they can more accurately control the rover when approaching the lizard.

We also added keyboard functionality using arrow keys to allow the user to more easily control smaller movements when approaching the lizard as they could click a key once to control a very small movement.

Server Setup

Initial Development

The first major problem we encountered in development was that the maximum packet size for the Metro M0 board was only 1500 bytes which was easily surpassed by the html webpage with the joystick. As a result, when we uploaded the joystick with the motor movement functions the webpage stopped working.

We investigated many ways to solve this, but the solutions mainly fell into two different categories: reduce the amount of data that we are transmitting and offloading the computation to a different device which can handle more complex operations than the Metro M0 microcontroller and WINC1500 Wi-Fi shield.

Hosting the Server on a Laptop

Our initial idea was to host the server on a laptop as opposed to the Metro M0 as it had much greater data handling capabilities and we could offload any computation to the laptop therefore minimising the amount of data that needs to be transmitted.

Upon testing the laptop server setup, it became clear that using a laptop added additional latency to the command processing and data transmission as it acted as an intermediary between the microcontroller and the server. As a result, there was a noticeable delay between the user interacting with the joystick and the rover responding making it harder to control the rover accurately as the user would have to take the extra delay before the rover responded to their input into account. It also added another point of potential failure as the laptop hosting the server would always have to be running and within range. Considering the above, we decided to develop a different solution.

Final Solution

ESP32

Instead of having a separate Wi-Fi shield and microcontroller, we decided to invest in a single microcontroller which had both Wi-Fi capabilities and the ability to manage larger packets of data more efficiently to eliminate the need to host the server on a laptop but still have a clean UI with all the functionality we wanted to include.

After further research we settled on the ESP32 as it was well known for its WiFi capabilities, dual-core processor and better memory management which meant it could handle data and more complex operations better without requiring frequent packet fragmentation. It was also extensively documented

so we were able to learn how to use it very quickly. Additionally, it was significantly lighter than the combination of the Metro M0 and WINC1500 WiFi shield which meant our rover could be slightly faster and under 750g as per the specification.

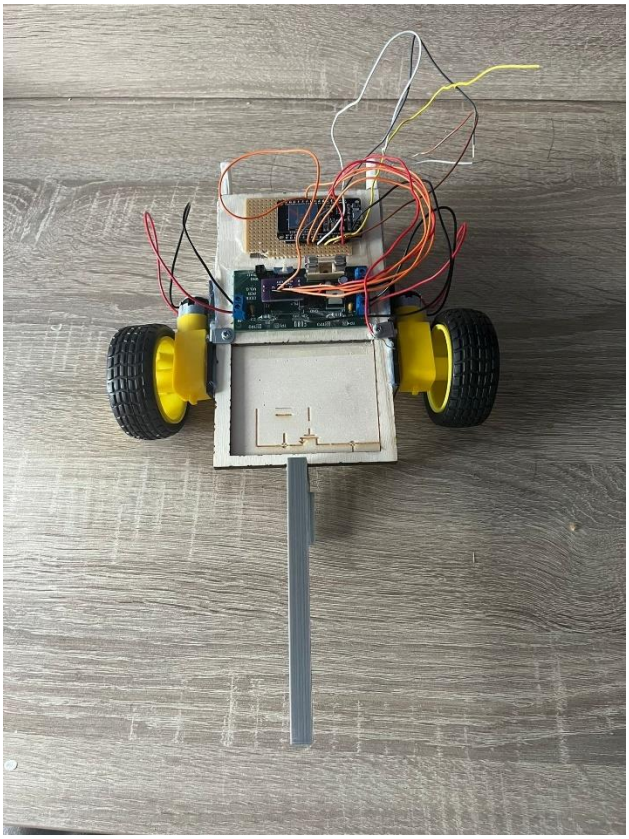


Figure 6637: ESP and rover assembled together

Updating States

In addition to upgrading our hardware, we also decided to reduce the amount of data the ESP had to handle by reducing how often the server sends updates to the rover. We found that by updating the rover every half second, we were able to reduce the amount of data the ESP had to process while also not compromising on any noticeable performance. As a result of our changes, the ESP was able to seamlessly and simultaneously handle both setting up the server and controlling the rover in response to user interactions with the webpage.

```
window.addEventListener('load', () => {
    canvas = document.getElementById('canvas');
    ctx = canvas.getContext('2d');
    resize();

    document.addEventListener('mousedown', startDrawing);
    document.addEventListener('mouseup', stopDrawing);
    document.addEventListener('mousemove', Draw);

    document.addEventListener('touchstart', startDrawing);
    document.addEventListener('touchend', stopDrawing);
    document.addEventListener('touchcancel', stopDrawing);
    document.addEventListener('touchmove', Draw);
    window.addEventListener('resize', resize);

    document.getElementById('scan-button').addEventListener('click', scanData);

    setInterval(() => sendMotorState(document.getElementById("motor_state").innerText), 500);
});
```

Figure 67: Rover update timer

Eliminating the PHP Back-end

After further research, we found we could eliminate the PHP back-end and instead handle the GET requests directly by using the Fetch API reducing latency and reducing the overall complexity of our system. By removing the PHP layer, we reduced the number of components in the communication pathway of our system making it less complex and as the request does not need to be processed by a separate server software stack, our rover responded quicker giving an improved real-time performance.

```
function sendMotorState(state) {
    fetch(`/${state.toLowerCase()}`)
        .then(response => response.text())
        .then(data => console.log(data))
        .catch(error => console.error('Error:', error));
}
```

Figure 6838: GET requests handled directly with the Fetch API

Displaying Real-Time Data

We display the real-time data from the sensors through a table and upload and delete buttons as shown below in *Figure 69*.

Figure 3969: UI with table of displayed real-time data

Sensor Integration

Radio and Infrared Receiver Integration

The microcontroller has an in-built Interrupt Service Routine (ISR) which measures the frequency of a given input pin. After initially testing with this, we found it worked however ultimately decided to create our own function shown in *Figure 70* as we did not want the frequency function of our rover to interrupt other more important tasks such as if the user suddenly decides to move away to avoid a crash with another rover.

```
18
19  void loop() {
20    //get the current state of the input pin
21    currentState = digitalRead(inputPin);
22    //Serial.println(currentState);
23
24    //check for a rising edge
25    if (currentState == HIGH && lastState == LOW) {
26      pulseCount++;
27    }
28
29    //update the last state
30    lastState = currentState;
31
32    //get the current time
33    unsigned long currentMillis = millis();
34
35    //check if the interval has passed
36    if (currentMillis - previousMillis >= interval) {
37      // Calculate the frequency
38      frequency = pulseCount * (1000.0 / interval);
39
40      //send the frequency to the Serial Plotter
41      Serial.println(frequency);
42
43      //reset the pulse count
44      pulseCount = 0;
45
46      //update the previousMillis variable
47      previousMillis = currentMillis;
48    }
49  }
```

Figure 4070: code snippet of the frequency measure loop

Ultrasound Receiver Integration

The microcontroller has a built-in function which automatically converts the digital input into ASCII characters which we utilised in order to decode the name of the lizard as shown below in *Figure 72*. We programme the microcontroller to read a string from the serial input until it reaches the '#' which marks the end of the lizard's name. If we are not able to read the lizard's name, it will output 'error' indicating to the user that they should try again.

```
void setup() {
  Serial.begin(9600);

  }

  Serial1.begin(600, SERIAL_8N1, 12, 13); // Initialize UART:RX pin 12, TX pin 13, baud rate 600

  Serial.println("UART is ready!");
}

void loop() {
  if (Serial1.available()) {
    String name_received = Serial1.readStringUntil('#');
    Serial.print("Name received: ");
    Serial.println(name_received);
  }
  delay(1000);
}
```

Figure 71: Ultrasound receiver integration with software (not inverted output)

Figure 7241: ultrasound name is detected and decoded

Magnetic Receiver Integration

To integrate the sensor into software and yield a single output containing the field polarity from the Arduino, the following code was written which consists of a main loop that continuously checks for the analogue output voltage of the sensor being above or below the threshold voltage, and outputting 'South Pole' or 'North Pole'. The voltage tolerance value and delay between readings was adjusted until output readings were consistent and occurred frequently enough whilst not utilising unnecessary processing power.

```arduino
const int hallSensorPin = A0;  // Analog pin A0

const float centerVoltage = 2.05;  // Center voltage
const float tolerance = 0.03;      // Tolerance range for the center voltage

void setup() {
  Serial.begin(9600);
}

void loop() {
  // Read the analog voltage from the Hall effect sensor
  int sensorValue = analogRead(hallSensorPin);

  // Convert the analog reading (0-1023) to a voltage (0-5V)
  float voltage = sensorValue * (5.0 / 1023.0);

  // Determine the magnetic field direction
  if (voltage < (centerVoltage - tolerance)) {
    Serial.println("North Pole");
  } else if (voltage > (centerVoltage + tolerance)) {
    Serial.println("South Pole");
  } else {
    Serial.println("No Magnetic Field Detected");
  }

  // Wait for a short period before reading again
  delay(1000);
}
```

Figure 7342: Test magnetic field detection function

Testing

Upon testing the webpage alongside our sensor integration, we found that while the ultrasound and magnetic inputs gave accurate answers, the radio and IR inputs gave frequencies between 0 and 2 instead of the lizard's emitted frequencies. After further investigation, we found we had not given the loop enough time to complete and as a result it did not measure enough pulses to accurately calculate the frequency. We fixed this by adding a duration as shown below in *Figure 74*.

Starting from 2 seconds, we repeatedly tested different durations which would be as small as possible while giving an accurate result until we decided on a duration of 0.5 seconds. This was not noticeable for the user but also long enough to ensure we had more than enough time to give an accurate result.

```
void handleRadio() {
  // Get the current state of the input pin
  unsigned long duration = 500;
  unsigned long startMillis = millis();
  pulseCount = 0; // Reset pulse count at the start

  while (millis() - startMillis < duration) {
    // Get the current state of the input pin
    currentState = digitalRead(radioPin);

    // Check for a rising edge
    if (currentState == HIGH && lastState == LOW) {
      pulseCount++;
    }

    // Update the last state
    lastState = currentState;
  }

  // Calculate the frequency
  frequency = pulseCount * (1000.0 / duration);

  // Send the frequency as a response
  Serial.println("Start Millis: " + String(startMillis));
  Serial.println("End Millis: " + String(millis()));
  Serial.println("Pulse Count: " + String(pulseCount));
  Serial.println("Frequency: " + String(frequency));
  server.send(200, "text/plain", String(frequency));
}
```

Figure 74: new radio and IR handle code with the stated duration of 500ms

We decided to add the feature of uploading each sensor up separately as this meant if there was an error uploading a single sensor, we could update only that sensor and nothing else.

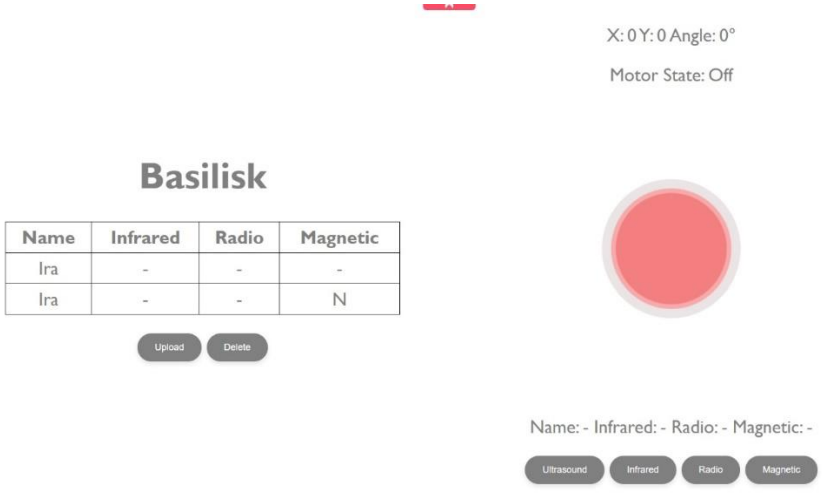


Figure 75: Table showing how individual sensor updates can be uploaded

Evaluation

Overall, our rover and webpage met all of our specification; we found everyone who tested our rover could control it accurately and found the UI very intuitive. The rover could be controlled with no noticeable delay between the webpage and the rover's response, and it responded correctly to the joystick, scan buttons and table control buttons. If we had more time, we could have perhaps investigated different protocols to see if this could improve performance such as the Websocket protocol. As our rover and webpage achieved integration with no noticeable delay, investigating other protocols was not a high priority for this project.

Full System Testing

For our full system test, we positioned the rover in the arena in the lab with a lizard in the opposite corner. We then proceeded to drive the rover across the arena and detect the relevant signals from the lizard to allow us to identify its name and species. We repeated this process 5 times and tried multiple different settings on the lizard so we could detect all the signals. The rover passed all these tests, correctly identifying all 5 lizard settings. This test also helped us to practice controlling the lizard so that we are more capable of delivering a good result on demo day. For our full system testing we also used a standard green lizard. It is important to note that on the demo day a modified yellow lizard with thinner skin will be used to reduce attenuation of the signals, and therefore the rover should be expected to perform even better.

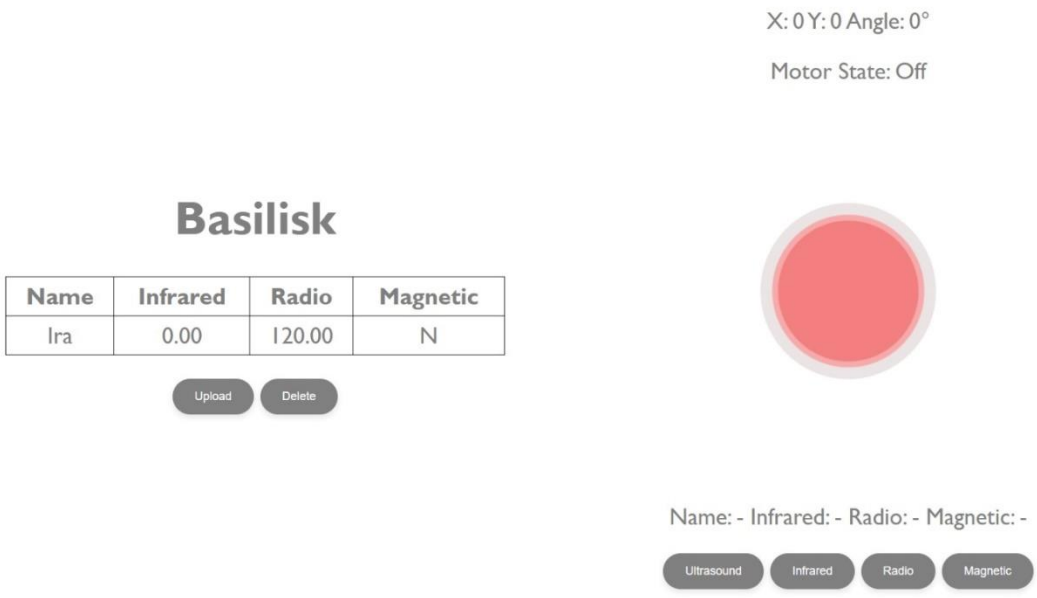


Figure 76: Example test case from full system testing – shows the name correctly as Ira and the species as Elgaria

Bill of Materials

Item name	Quantity	Part number	Cost
WiFi module	1	ESP32-C3-32S	£5.24
DIODES INC. AH49FZ3-G1 (Hall Effect Sensor)	2	AH49FZ3-G1	£2.06
TL071 Operational Amplifier	1	TL071ACP	£0.76
LM358P Operational Amplifier	3	LM358P	£0.27
70x25mm Wheel and Tyre	2	MC02755	£3.08
PROWAVE 400SR160	1	400SR160	£2.51
Optek OP593B IR Phototransistor	2	OP593B	£1.20
PCB	1		£24.66
Total			**£39.78**

Figure 7743: Team Basilisk's bill of materials

Evaluation and Future Work

Our rover worked well and achieved all the necessary points in the specification. It could detect all the relevant signals from the lizards, from a distance greater than the minimum distance of 6cm.

In the future it could have been better if the sensors had a greater range to reduce the precision required when operating the rover. This was especially important for IR as the lizard attenuated the signal significantly. It could have been improved by increasing the gain of the differential amplifier and increasing the saturation voltage of the op amp so there was a greater range of output voltages from the differential amplifier. For this project we chose not to prioritise this to have extra budget to spend on the PCB.

The rover weighed 478g which while under our goal of 700g, could have been lighter. We could have used a 9V battery to power the buggy, due to its smaller size and weight, and a load regulated voltage regulator to achieve the same 6V output the current batteries give. To save weight, the original EEEBug PCB could have been integrated within the new custom PCB. This would have helped simplify the power connections and saved space on the chassis reducing the overall size and therefore weight and increasing the manoeuvrability of the rover. We could not integrate this in our project as this would have expanded the PCB making it too expensive to fit within our budget; we expected this would have added £15-20 to our PCB cost. We would then not have been able to access spare components for testing or emergencies.

The rover had good manoeuvrability but could have been improved by using four motors instead of two. We decided against this due to the weight limit of 750g. The UI was very user friendly and multiple people who were given no instruction were able to drive the buggy and interpret the output of the sensors without help.

If we had more time, we would have preferred to investigate other protocols such as the Websocket protocol to try and improve the rover's real-time performance in software. Despite this, with our current solution the rover performed with no noticeable delay.

References

1. 'GitHub: EEERover2024' - https://github.com/hakanmerdan/EEERover2024/blob/main/doc/brief.md
2. 'PROWAVE 400SR160 Datasheet' – https://www.farnell.com/datasheets/3109335.pdf
3. 'OP593B Datasheet' - https://docs.rs-online.com/1c44/0900766b814b5188.pdf
4. 'AH49F Datasheet' - https://www.diodes.com/assets/Datasheets/AH49F.pdf
5. “Mecanum wheel” - https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.pinterest.com%2Fpin%2F372039619199299604%2F&psig=AOvVaw3NensBNN1tCw-0y5bxQqeq&ust=1718396809788000&source=images&cd=vfe&opi=89978449&ved=0CBQQjhxqFwoTCIj114612YYDFQAAAAAdAAAAABAE
6. “MC02755 Wheel” - https://onecall.farnell.com/unbranded/mc02755/wheeltyre-70-x-25mm/dp/MC02755