

Imperial College London

EEE Lunar Rover

Project Report

ELEC40006 – Group 27

11th June 2026

Words: 9,909

Morgan Bleasdale

Justin Huang

Junchen Jiang

Aditya Shivam

Jack Ward

Fahad Khan

Edited by Morgan Bleasdale

Contents

Abstract	4
Introduction	5
Project Brief	5
Subsystems.....	6
Subtasks	6
Project Management.....	8
Initial Project Plan.....	8
Project Management Methodology (Agile)	8
Roles and responsibilities	9
Sensing: Radio (Age)	10
Background	10
Antenna Design	11
LTSpice - Full Radio Circuit	13
Implementation and Testing	19
Software.....	22
Critical Evaluation.....	24
Sensing: Infrared (Type).....	25
Background	25
Photodiodes and Darlington Pair	27
LTSpice Circuit Design	28
Implementation and Testing	31
Software.....	32
Sensing: Ultrasound (Type).....	33
Background	33
Circuit Design	34
Implementation and Testing	38
Software.....	40
Sensing: Magnetic Field (Type)	41
Background	41
Circuit Design	42
Implementation and Testing	44
Software.....	45
Rover Movement	46

Background	46
Hardware Components.....	46
Software (Movement Library)	48
Implementation and Testing	49
Networking	50
Requirements.....	50
HTTP Setup.....	50
HTTP Software	51
User Interface.....	52
Communicating Instructions and Data	54
Evaluation: Attempted Upgrades.....	55
Chassis Design	56
Subsystem Integration.....	60
Evaluation	62

Abstract

This report presents the design, construction and testing of the EEE Lunar Rover, a remotely controlled rover with the goal of surveying an artificial lunar surface and classifying rocks by age and type. Age was decoded from an 89 kHz ASK carrier signal carrying UART data, while rock type was identified by combining three other measurements. An ultrasonic sensor detected the presence or absence of a 40 kHz signal, a magnetic sensor measured field polarity, and an infrared sensor detected the average infrared pulse rate. The rover integrated four sensing subsystems whilst being controlled remotely over Wi-Fi, within a 750g weight limit, successfully classifying all rock types and meeting the design specification.

Introduction

Project Brief

The project involved the design of a remotely controlled rover, which could survey an artificial lunar surface. On the test area are placed rocks emitting a variety of electromagnetic signals. The signals represent the type and age of the rock. The rover must navigate the test area and classify each rock's type and age.

Rock age is encoded using ASCII character codes in UART packets, transmitted via radio waves, at a 89 kHz carrier frequency, using two-level amplitude-shift keying (on or off).

Rock type is given by a combination of magnetic field direction, ultrasound and infrared signals.

There are four rock types, each has a magnetic field pointing either up or down, and infrared pulses emitted at different Poisson distributions. Two of the four types have a ultrasound signal at 40 kHz.

Table 1.1 shows the different rock types and their expected sensor data:

Rock Type	Infrared	Ultrasound	Magnetic Field
Basaltoid	$\lambda=547 \text{ s}^{-1}$	40kHz	Down
Gravion	$\lambda=312 \text{ s}^{-1}$		Down
Regolix	$\lambda=312 \text{ s}^{-1}$	40kHz	Up
Lunarite	$\lambda=547 \text{ s}^{-1}$		Up

The rover must also be designed with realistic weight constraints to model a “Weight Sensitive Zone” which has loose rock. For this, the complete rover must weigh a maximum of 750 grams.

The quality of design will also be assessed on:

- Rover manoeuvrability
- Reliability and robustness of rover structure
- Understandable and easy to use user interface and rover controls

Approaching the project brief

After reading the project brief we decided to group the project into 3 general objectives.

- **Age Detection** – building our own radio antenna and detecting a binary number representing age, emitted via radio waves from the rock.
- **Type Detection** – identifying type based on combination of sensor inputs
 - Detecting magnetic field direction (up or down)
 - Detecting infrared pulse average (312 s^{-1} or 547 s^{-1})
 - Detecting ultrasound presence
- **Rover Movement** – design of a rover based on EEEBug design that can navigate test area

Subsystems

We broke down our 3 general objectives and the project brief into 7 distinct subsystems.

Each subsystem could be worked on independently by a different member of the team, which would allow us to parallelise our workflow and make more efficient progress on the project.

Subsystems had to take into consideration the wider project requirements and work with the rest of the subsystems to ensure interoperability. We planned on using a single Arduino, so subsystems had to work together within the same circuitry (no interference).

The 7 subsystems we identified were:

- Sensing: Radio – Age Detection
- Sensing: Infrared – Type Detection
- Sensing: Ultrasound – Type Detection
- Sensing: Magnetic Field – Type Detection
- Rover Movement
- Networking
- Chassis Design

Other areas of work that could be considered a subsystem, such as rover assembly and testing, were considered throughout.

Subtasks

The table below shows the subsystems we initially identified and their corresponding subtasks.

Sensing – Radio (Age)	
Antenna assembly	Building air-cored inductor antenna and finding matching capacitor value to resonate at carrier frequency.
Amplifier	Amplify signal so it is not distorted after voltage drop across diode in the envelope detector.
Envelope detector	Filter out carrier signal and extract the data signal.
Comparator	Clean output from envelope detector by driving HIGH/LOW binary signals to clearly defined 3.3 V and 0 V logic levels.
UART Decoding	Decode binary pulses into ASCII for age determination.
Sensing – Infrared (Type)	
Sensor selection	IR Photodiode provided from EEEBug or find more sensitive alternative online and order, such as a phototransistor.
Amplifier	Amplifier needed so signal is strong enough for Schmitt trigger.
Filter	Low pass filter to block low-frequency noise.
Schmitt Trigger	Clean pulses into clearly defined HIGH/LOW logic signals.
Sampling and calculation	Sample pulses and calculate average λ . Classify as 312 or 547.
Sensing – Ultrasound (Type)	

Sensor selection	Find sensor with a centre frequency of 40 kHz and a narrow beam angle to focus on target rock
Filter	Band-pass filter to attenuate signals larger or smaller than 40 kHz
Amplification	Amplify signal so it is large enough to be detected by Arduino ADC
Sampling and detection	Sample value of input signal to detect if ultrasonic is present
Sensing – Magnetic (Type)	
Sensor selection	Find suitable Hall effect sensor
Amplifier	Amplify signal so it can be detected by Arduino ADC
Detection	Determine magnetic field direction
Rover Movement	
EEEBug assembly	Validate that the EEEBug assembled in January still works and is sufficient to act as the base of the rover
Movement control functions: D-pad	Software for movement controls that take binary (on/off) button signals and adjusts motor speed and direction
Movement control functions: joystick	Software for movement controls that take continuous signal representing joystick position and adjusts motor speed and direction
Networking	
HTTP server	Set-up HTTP server as described in GitHub docs
Device connection	Connect laptop or another device to rover through HTTP network
User interface	Design and program user interface for giving movement controls to rover and displaying rock type and age. UI is hosted on laptop
Movement controls	Establish communication from UI to rover to send movement controls
Rock data	Establish communication from rover to UI to send rock age and type

Project Management

Initial Project Plan

Area	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Radio	Simulation	Building				Interview and demos
Ultrasound	Simulation	Building				
Infrared	Simulation	Building				
Magnetic	Simulation	Building				
Movement	Basic rover movement		Optimisation			
Networking		Movement Control		Data display		
Chassis Design				Design	Assembly	

We planned to spend the first week conducting research and simulating our initial ideas for circuits. By week 2, when we started building circuits, we already had high confidence that our circuits would work.

Some tasks like networking and chassis design were dependent on other parts of the project being completed first. We waited until after rover movement was completed to begin networking, and we waited until all circuits had been built for chassis design.

Report writing, research, and testing was done throughout and so were not placed as specific tasks on the Gantt chart.

Project Management Methodology (Agile)

We adopted the Agile framework. At the start of each week, we held a sprint planning meeting, during which the team identified and defined subtasks for the upcoming week. Each subtask was assigned to a specific team member within the relevant subsystem, ensuring clear ownership and accountability.

We tracked progress continuously throughout each sprint. We held midweek team progress meetings, where we would update the list of subtasks based on current progress. If necessary we would also reallocate the task or assign additional team members to it.

After each sprint we would review the completion of the subtasks for the week. Any outstanding tasks were redefined, to identify why they weren't completed, and moved to the next sprint.

Overall the Agile framework was well-suited to this project. It allowed our planning to be iterative and dynamic, enabling the team to remain responsive to unexpected circumstances. For example, subtasks that proved harder than we first anticipated could easily have their timeline readjusted.

Roles and responsibilities

The table below shows the responsibilities of each team member. Even though these subsystems were assigned, all team members had consideration to the wider project. We would often help out with each other's work, even if we weren't assigned to work on that subsystem.

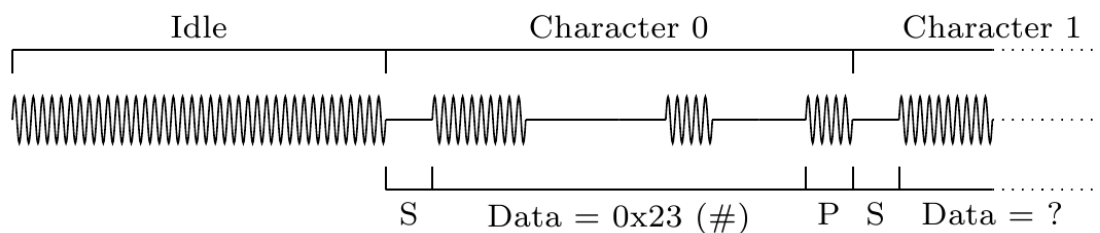
Team member	Responsibilities
Morgan Bleasdale	Project management, report writing, interim presentation, radio sensor
Junchen Jiang	Magnetic sensor, ultrasound sensor
Jack Ward	Radio sensor and report writing
Fahad Khan	Magnetic sensor
Aditya Shivam	Rover movement, networking, and infrared sensor
Justin Huang	Chassis design and rover assembly

Sensing: Radio (Age)

Background

The rock transmits its age as a radio signal at a carrier frequency of 89 kHz, modulated using Amplitude-Shift Keying (ASK), where the presence of the carrier represents logic 1 and the absence of the carrier represents logic 0.

The age itself is encoded as a string of ASCII characters, each framed as a UART packet.



S = Start Bit (Logic 0)

P = Stop Bit (Logic 1)

Data transmitted LSB first

During idle, the carrier wave is at full amplitude. This means the rock is always broadcasting, sustaining a logic 1 until a character is ready to send.

Transmission of each character begins with a start bit (S), where the carrier drops to zero amplitude, signalling to the receiver that a data frame is beginning.

The 8 data bits then follow sequentially, each occupying a fixed time slot determined by the UART Baud rate in which the carrier is either present (logic 1) or absent (logic 0), where the LSB is transmitted first.

Bit	0	1	2	3	4	5	6	7
Value	1	1	0	0	0	1	0	0

Example waveform. Equal to 0x23, a hashtag (#) in ASCII.

Once all 8 data bits are transmitted, a stop bit (P) returns the carrier to full amplitude for one bit period, terminating the frame.

As shown in the diagram, a second character begins immediately after with another start bit, representing the next ASCII character of the age string yet to be decoded

In this way, the rock continuously broadcasts its age character-by-character, with each ASCII character wrapped in a UART frame and encoded onto the 89 kHz carrier using ASK modulation

The data rate is 600 bits per second, and each rock's age is four characters long (a hashtag followed by 3 numbers).

Antenna Design

Antenna Construction

To receive the radio signal from the rock, an antenna was required. At this frequency, the most appropriate type of sensor is a tuned coil antenna. Essentially an air cored inductor that when placed in a tuned LC circuit, becomes resonant at the desired frequency and selectively sensitive to it.

The coil was wound using 1.5 mm copper wire, wrapped around a metal bottle of approximately 7 cm diameter. The coil was kept as tight as possible to maximise inductance.

The inductance was then measured using an LCR bridge, which gave a reading of exactly 500 μH . Before measurement, the ends of the wire were sanded down to remove the coppers oxidation layer, ensuring a good electrical contact.



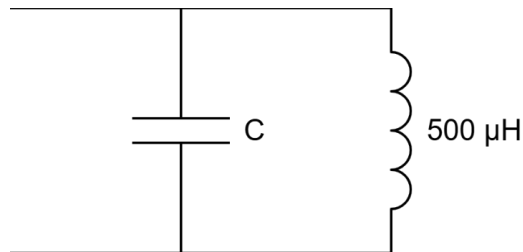
Image of our antenna

Resonant LC Circuit

We needed an LC circuit tuned for a resonant frequency of 89 kHz, as this is the same as the carrier frequency of the radio signal.

The coil antenna had an inductance of 500 μH , so we needed to find a matching capacitor value.

Assuming a parallel LC circuit, as shown below:



Using the formula for resonant frequency:

$$\omega_0 = \frac{1}{\sqrt{LC}}$$

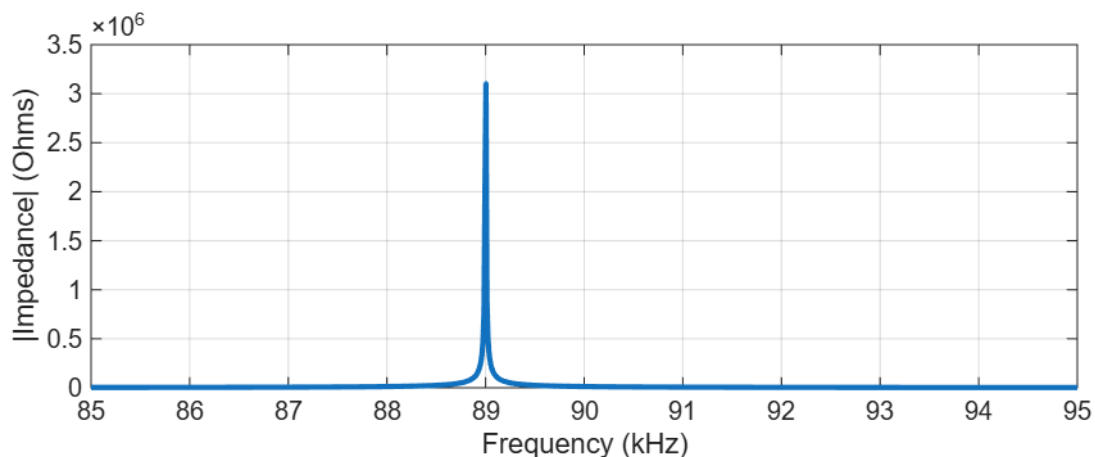
Rearranging for capacitance and using $\omega_0 = 2\pi f_0$

$$C = \frac{1}{L(2\pi f_0)^2}$$

$$C = \frac{1}{(500 \times 10^{-6})(2\pi \cdot 89 \times 10^3)^2}$$

$$C = 6.4 \text{ nF}$$

The calculated capacitance of 6.4 nF will bring the LC circuit to resonance at 89 kHz.



Resonance frequency occurs at 89 kHz using the 6.4 nF capacitor. Using MATLAB.

At resonance, the LC circuit has maximum impedance at 89 kHz, meaning the circuit is most sensitive to the rock's carrier frequency and naturally rejects signals at other frequencies.

At this point the carrier signal broadcast by the rock has been successfully picked up by the coil antenna and isolated from background noise by the tuned LC circuit, producing a weak but frequency selective signal ready to be processed.

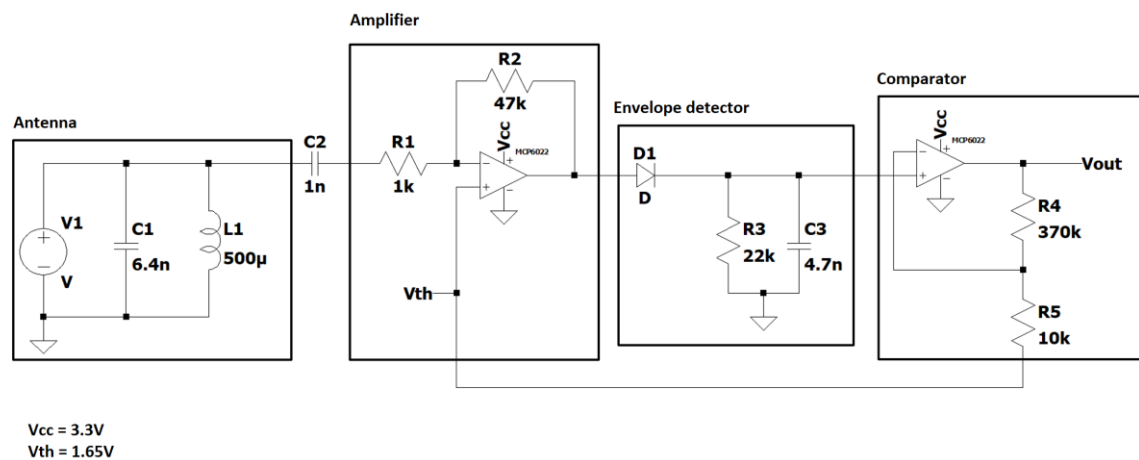
LTSpice - Full Radio Circuit

We first designed the full radio circuit on LTSpice and simulated it using the MCP6022 op-amp to verify the design before physical implementation.

Before simulation, the antenna was tested in the lab by placing it approximately 5cm away from the rock (the maximum expected operating distance) and measuring the received signal amplitude on an oscilloscope. This gave an input signal of approximately 500 mV peak, which was used as the basis for all subsequent design decisions and set as the amplitude of the input source in the simulation.

The input signal was modelled as an 89 kHz sine wave at 500 mV amplitude, switched on and off to replicate the ASK modulated UART signal broadcast by the rock.

This is our final design for the radio circuit.

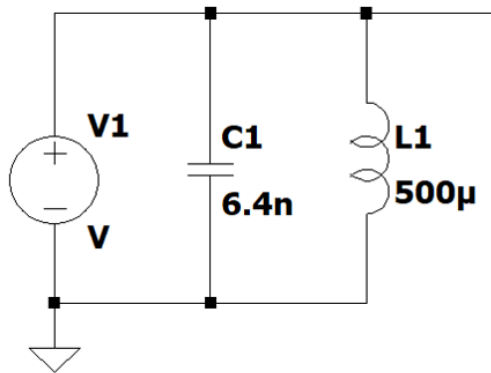


Annotated radio circuit in LTSpice. $V_{cc} = 3.3 V$ and $V_{th} = 1.65 V$.

Stage 1: Antenna

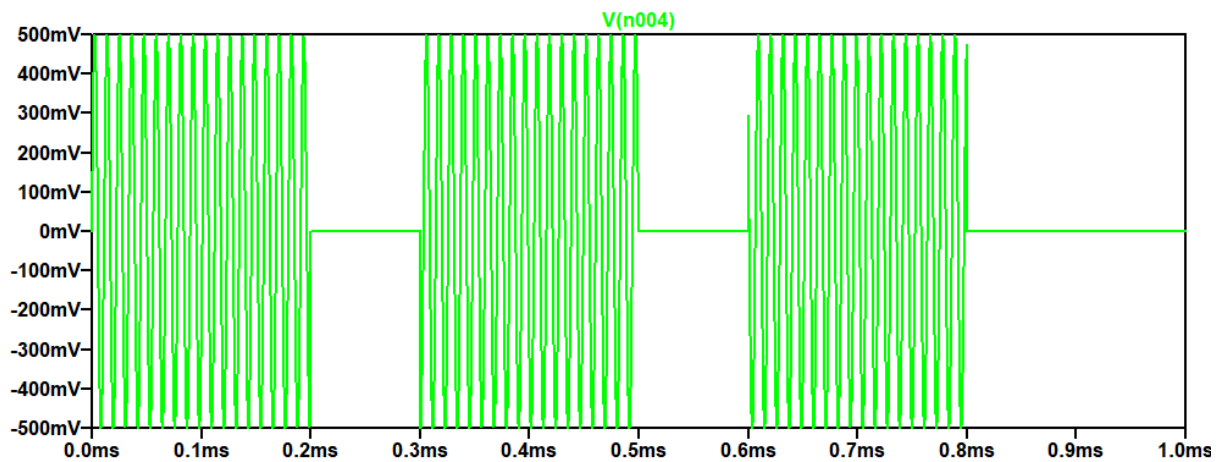
The antenna consists of the coil in parallel with capacitor C1 (as measured before).

At resonance, the LC circuit presents maximum impedance at exactly 89 kHz, making it most sensitive to the rocks carrier frequency whilst naturally rejecting signals at other frequencies. This improves the signal to noise ratio before any amplification takes place.



LTSpice circuit for antenna, modelled as an LC circuit and a voltage source.

Below is the output from the antenna circuit. This models the radio output from the rock, with the antenna held about 5 cm away. The ASK is visible, when a logic level of HIGH is asserted the carrier signal is on, otherwise it is 0 for a LOW logic level.

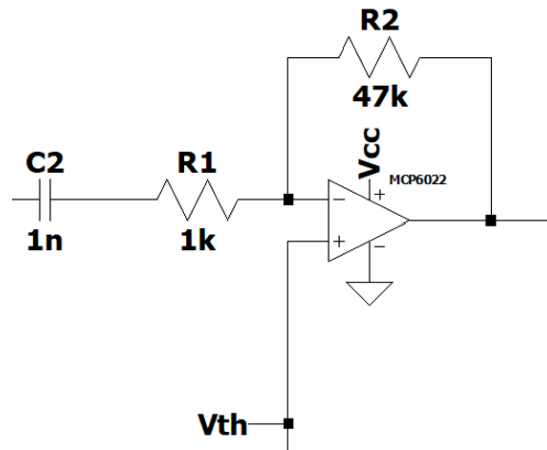


LTSpice simulation trace of antenna output (set input signal)

Stage 2: Amplifier

The amplifier stage takes the weak 500 mV antenna signal and amplifies it to a level such that the envelope detection works correctly.

The diode requires a minimum forward voltage of 0.7 V to conduct; hence we need the amplifier to ensure the signal exceeds this threshold, allowing for envelope detection to take place. With the 500 mV input, amplification is needed before the signal reaches the diode.



Inverting amplifier stage used to amplify antenna signal

The signal first passes through C2 (1 nF), a coupling capacitor which blocks DC offset from the antenna, ensuring only the 89 kHz signal reaches the amplifier input.

The signal then enters the inverting terminal of the op-amp through resistor R1 (1 kΩ), with feedback resistor R2 (47 kΩ) setting the gain:

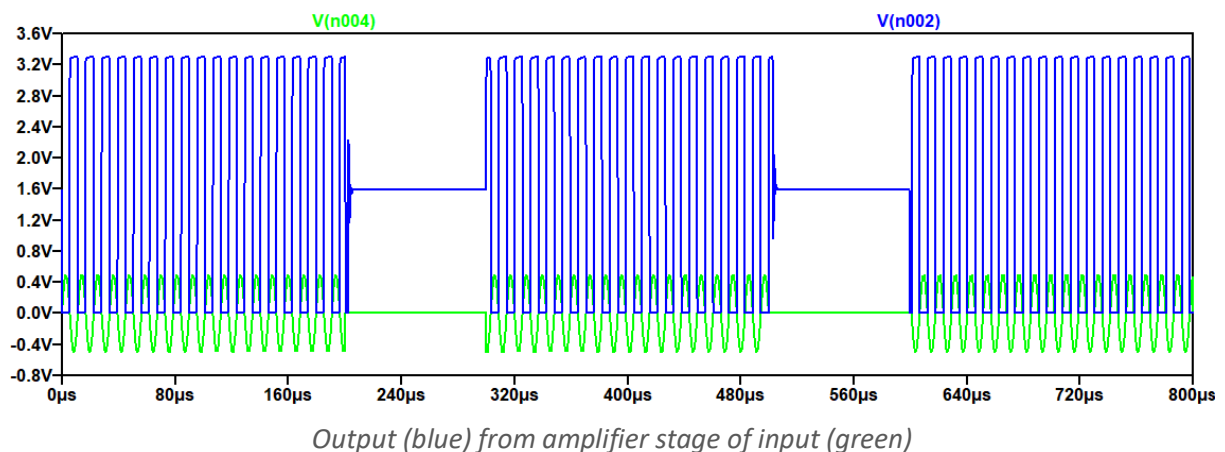
$$Gain = -\frac{R_f}{R_1} = -\frac{47}{1} = -47$$

The gain of 47 was chosen as a value with a high gain.

The op-amp output cannot exceed its 3.3 V supply rail, so it saturates and clips at the supply rail, which is desirable. Rather than linearly amplifying to an exact voltage, the high gain ensures the output is driven hard into saturation during carrier bursts, producing a clean square wave switching between 0 V and 3.3 V at 89 kHz.

This gives a well-defined signal with maximum amplitude at every carrier cycle, guaranteeing the diode is driven well above its 0.7 V threshold on every positive half cycle.

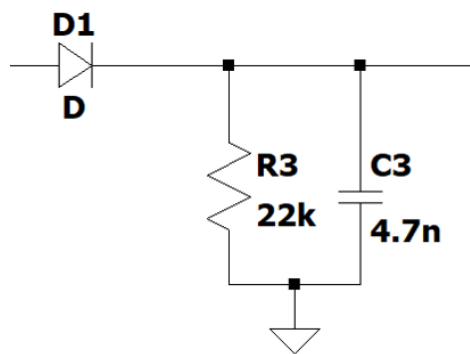
The inverting amplifier configuration was used as it is simpler to implement and was the natural starting point of the design. As a result of this, there is a 180° phase difference of the output with the input. To account for this, an inverting comparator was used in the last stage introducing a second phase inversion, which cancels out the first.



Stage 3: Envelope Detection

The amplified signal from the previous stage is still has the 89 kHz carrier signal, which the microcontroller cannot sample.

The purpose of the envelope detector is to extract the slower on/off pattern of the data from the carrier, producing a slowly varying voltage that reflects whether the carrier is present or absent at any given time.



Envelope detector stage

Diode, D1, only allows the positive half cycles of the carrier through, blocking the negative portions.

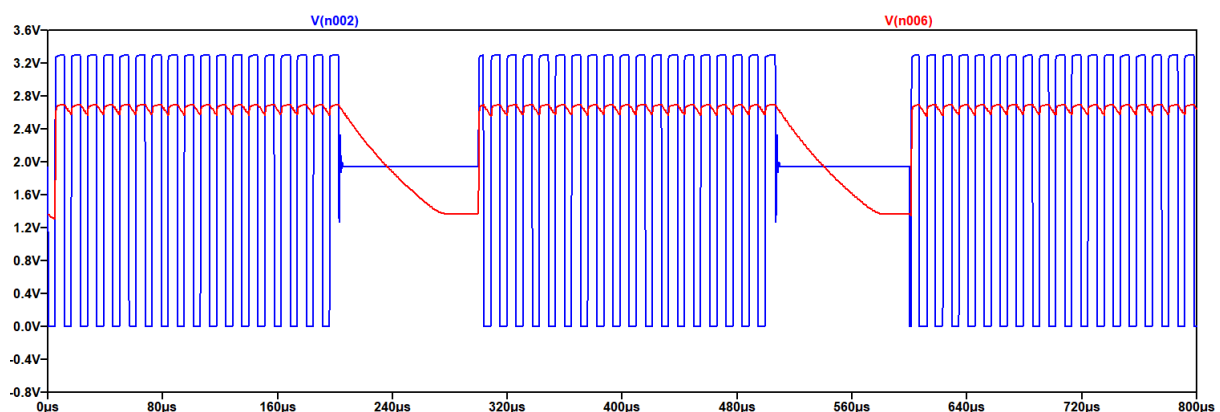
The amplified signal saturates at around 3.3 V (above the diodes threshold), therefore it will conduct reliably on every positive half cycle, passing a series of pulses to the RC filter.

R3 (22 kΩ) and C3 (4.7 nF) then form an RC low pass filter that smooths out these pulses into a slowly varying envelope voltage. The time constant of this filter is:

$$\tau = RC$$

$$\tau = (22 \times 10^3)(4.7 \times 10^{-9}) = 103 \mu\text{s}$$

The time constant was chosen such that it is greater than the carrier period, so the capacitor does not fully discharge between individual carrier cycles.



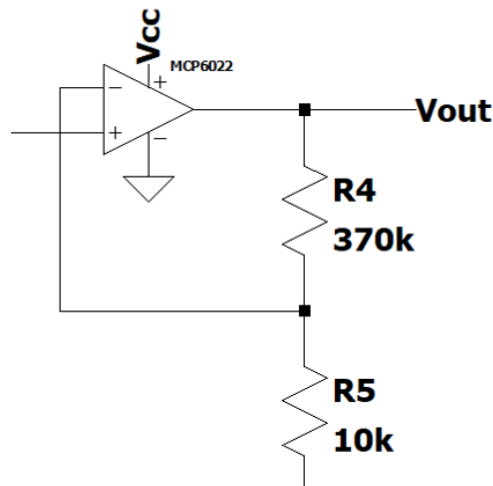
Output from envelope detector (red)

When the carrier switches off, the capacitor begins to discharge through R3 and the envelope slowly decays. The envelope signal now directly reflects the binary data pattern of the original ASK transmission and is ready to be converted into a clean digital signal by the comparator stage.

Stage 4: Comparator

The envelope signal from the previous stage is a slowly varying analogue voltage, rising and falling smoothly rather than switching cleanly between logic levels.

This cannot be directly processed by the microcontrollers UART peripheral. The comparator stage converts this analogue envelope into a clean digital output.

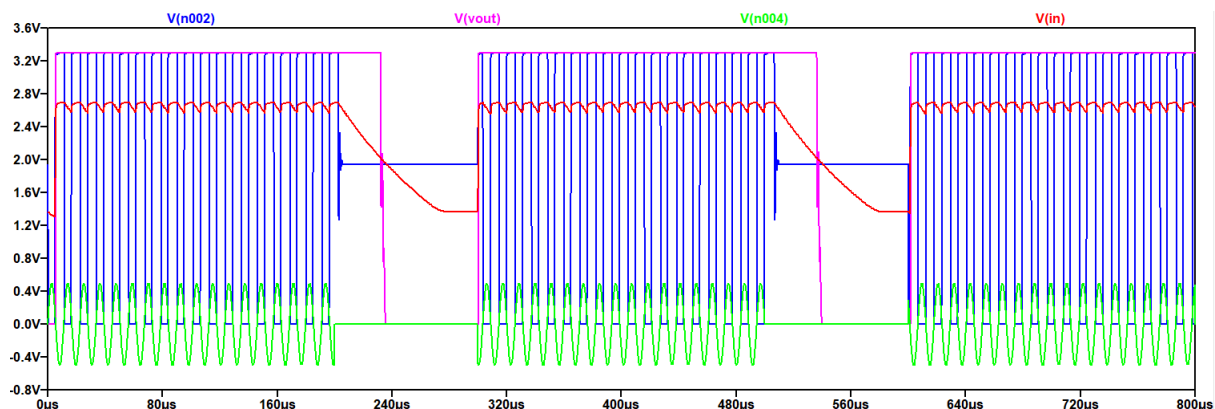


Inverting comparator. Drives HIGH voltages to 3.3 V and LOW to 0 V

The second op-amp is configured as an inverting comparator to cancel out the phase inversion introduced earlier in stage 2. This means the final digital output reflects the original signal polarity.

R4 (370 k Ω) is used to feed a small portion of the output voltage back to the threshold reference. This creates a band of hysteresis around the switching point.

R5 (10 k) connects the non-inverting input to ground, setting the baseline reference voltage. Together these two resistors set the switching threshold and the width of the hysteresis band.



Comparator output (pink). See also original input (green), amplified input (blue), envelope detector (red).

In the simulation you can see a clean the pink output trace as a clean digital output switching between 3.3V and 0V, directly mirroring the UART framed ASK data pattern transmitted by the rock.

The threshold voltage was initially set to 2 V in simulation. The value was chosen by looking at the envelope signal in the simulation, the envelope detector output (red trace) sits at approximately 2.6 V when the carrier is present and decays to approximately 1.3 V when the carrier is absent.

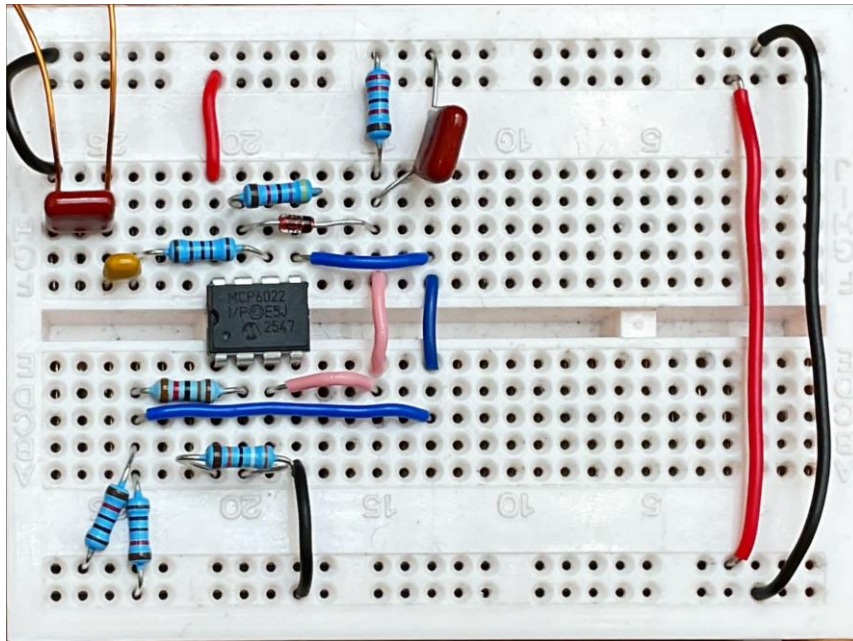
Setting the threshold to 2 V places it roughly midway between these two levels, ensuring the comparator switches cleanly at the correct point.

This threshold was chosen based on the simulated envelope levels, which may not exactly match the real circuit. Therefore, once the circuit is built and the envelope is observed on an oscilloscope, the threshold can be adjusted accordingly.

Implementation and Testing

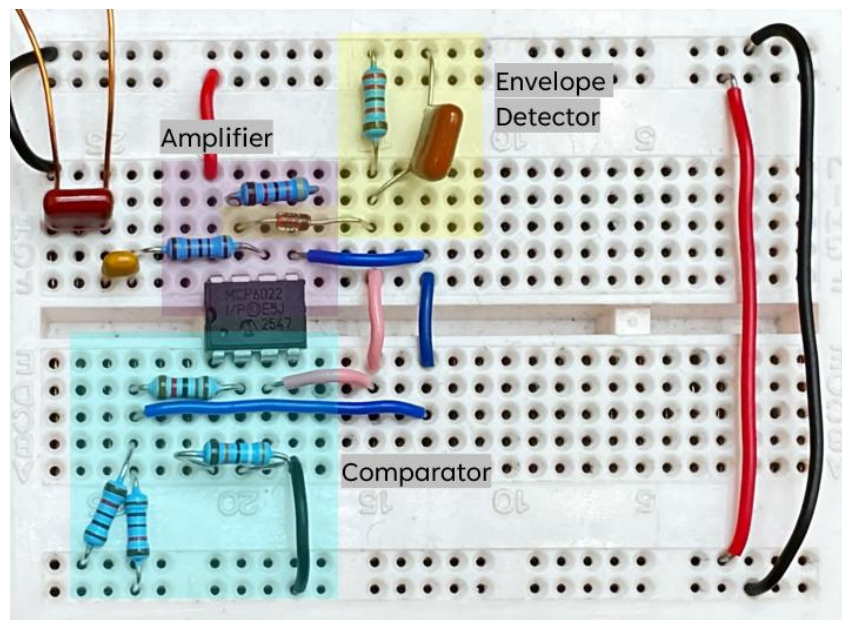
We initially built our circuit on a breadboard without consideration of how much space the circuit will occupy. This allowed us to test the physical implementation.

See below our initial circuit.



Physical implementation of radio circuit on a breadboard

See below the main features of the circuit annotated. See the LTSpice simulation for explanations.



Annotated radio circuit

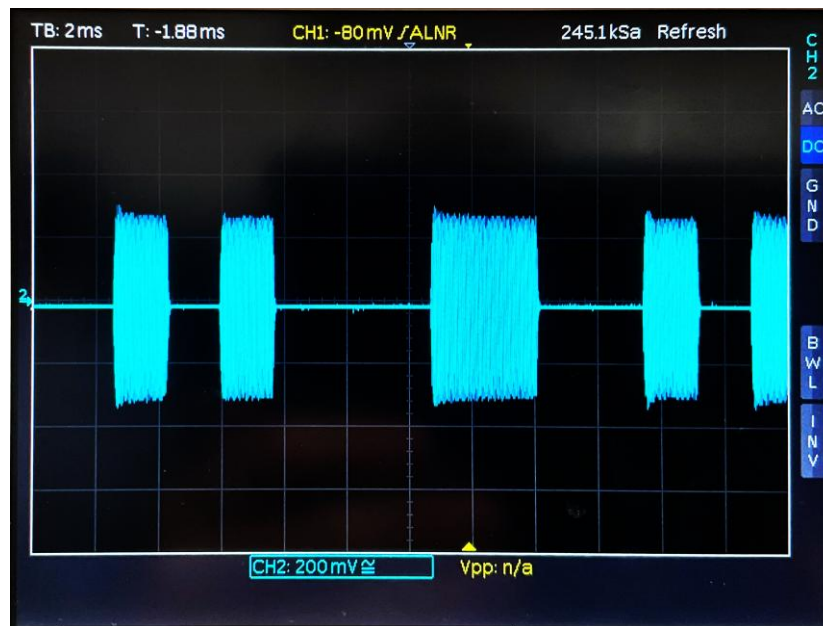
The MCP6022 op-amp was used in this circuit. It was chosen due to its high slew rate of 7 V/ms and the 10 MHz gain bandwidth product.

The high op-amp slew rate was needed so the op-amp can respond fast enough at the 89 kHz frequency. A low slew rate would cause the amplified waveform to distort, reducing the effectiveness of the envelope detection stage.

Once built, each stage of the circuit was probed on the oscilloscope to verify correct operation.

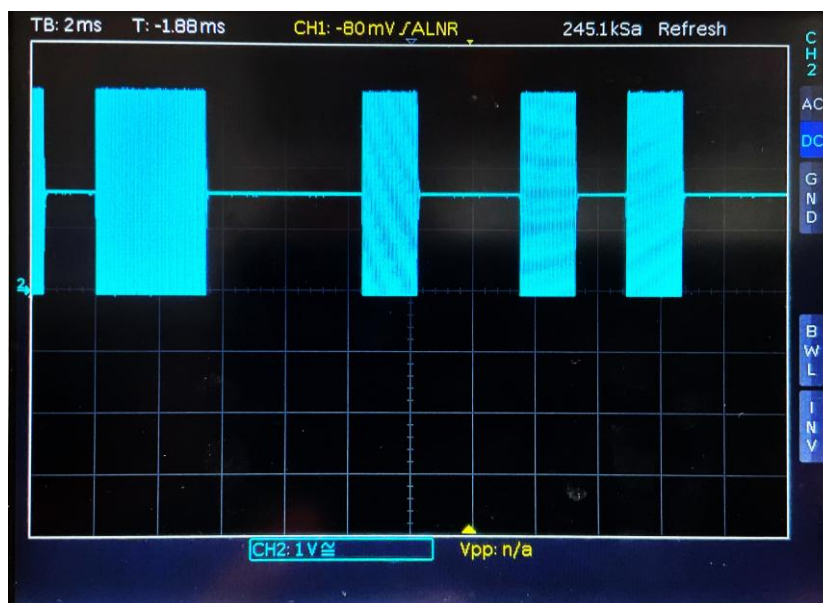
Oscilloscope Waveforms

Firstly, the input signal. This is the raw antenna output at a distance of 5 cm from the rock, consistent with what we expected.



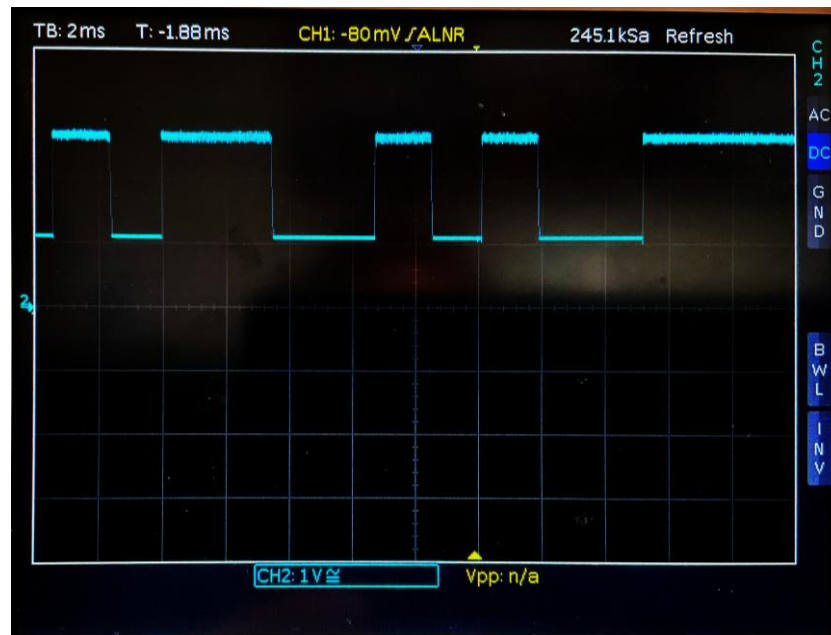
(1) Input signal from the rock radio

This signal is then amplified by the inverting amplifier. The output is driven to the supply rails, producing a clean near-square wave.



(2) Radio signal from rock amplified by inverting amplifier

The amplified signal enters the envelope detector. This filters out the carrier signal. It produces a slowly varying voltage rising and falling in correspondence with the presence and absence of the carrier.



(3) Output from envelope detector

The output from the envelope detector then enters the comparator, which clearly defines logical HIGHs at 3.3 V and LOWs at 0 V.

The envelope was slightly different to what we expected, so we adjusted the threshold to 1.65 V. This ensured the comparator switched cleanly at the correct point, producing a valid digital output ready to be fed into the UART module.



(4) Output from comparator. Final output of the radio circuit

The output from the comparator is then fed into the Arduino to be decoded.

Software

The main aspect of software for the radio was decoding the UART age.

The clean digital signal from the comparator was fed into the Metro microcontroller for UART decoding. The Metro has two independent UART ports. `Serial1` was used to receive the rocks signal, with the output of the comparator connected to pin 0 (RX).

The UART was initialised at a baud rate of 600 as defined in the code later on. This was chosen to match the rocks transmission rate of 600 bits per second.

```
#define RX 0
#define TX 1

void init_radio() { Serial1.begin(600); }

int extract_buffer(char *buffer) {...}

int age() {
    char buffer[20];
    int buff_index = 0;

    while (buff_index < 20) {
        if (Serial1.available()) {
            char c = Serial1.read();
            buffer[buff_index] = c;
            buff_index++;
        }
    }

    return extract_buffer(buffer);
}
```

Snippet of the code used to decode UART

The `age()` function reads incoming characters into a 20 character buffer using `Serial1.available()` and `Serial1.read()`, filling the buffer one character at a time until 20 characters have been received.

The buffer is then passed to `extract_buffer()`, which processes the raw character stream to extract age value.

This gave the following output:

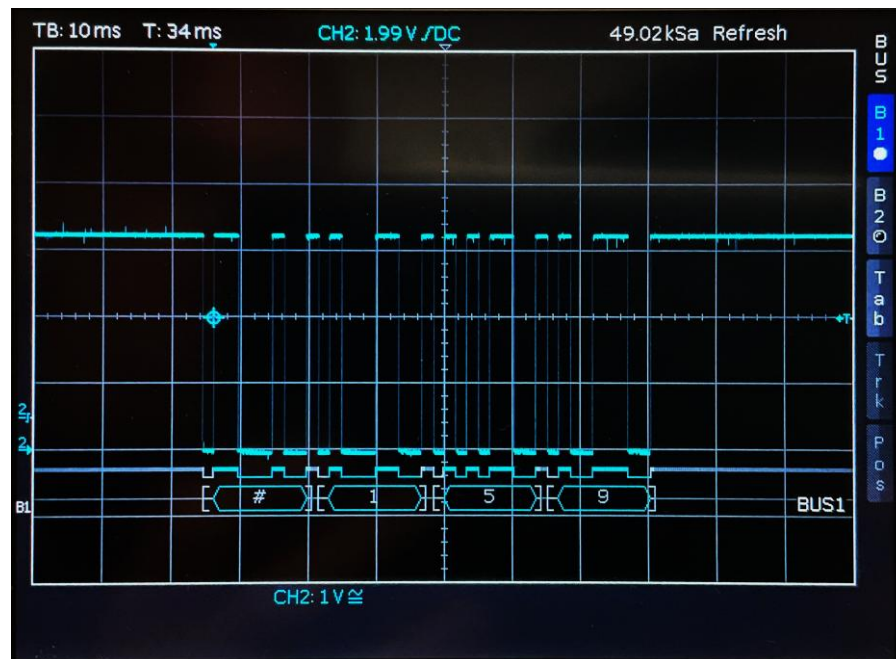


Arduino output of the ASCII decoding of the UART signal. This is the rock's age in millions of years.

The serial monitor output confirms the circuit and code are working correctly, showing the decoded output as #159 repeating, meaning the rocks age is 159 million years old.

The #character acts as the start delimiter of each transmitted value. The age is continuously broadcasted and decoded with each repetition confirming a successful reception from the 89 kHz radio signal all the way to a readable age value on the serial monitor.

We can also decode the UART signal into ASCII using the oscilloscope, which also confirms our code is working correctly. The oscilloscope output was also #159.



Oscilloscope decoding of the UART radio signal. The value was the same as the value calculated by our Arduino code.

Critical Evaluation

Overall, the receiver met its requirement. It recovered the 89 kHz ASK signal at the 5cm operating distance and decoded age correctly, confirmed independently on both the serial monitor and the oscilloscopes UART decode.

Inverting Amplifier Stage

We chose an inverting amplifier configuration because it was the simpler starting point, but this introduced a 180 degree phase inversion which we then had to cancel out with a second inversion at the comparator stage.

The inverting design worked but in hindsight it added complexity to compensate for a choice made for convenience rather than performance.

Comparator Threshold

In LTSpice the envelope sat at roughly 2.6 V high and 1.3 V low, so we set the threshold to 2 V for the comparator.

On the physical circuit the envelope behaved differently, so we changed the threshold to 1.65 V for clean switching.

This discrepancy is most likely because LTSpice assumed an idealised op-amp model. This showed us that the simulation is useful for establishing rough values but is not capable for establishing a precise threshold.

Antenna Design

In the design of our antenna, we initially wound a coil of inductance 20 μH , but a coil this small picks up a weaker signal since fewer turns produced a smaller induced EMF.

The weak antenna output would have required substantially more total gain to lift the envelope above the diodes 0.7 V conduction threshold, which is more gain than a single op-amp stage can deliver at 89 kHz, given the finite gain bandwidth product of the MCP6022.

By rewinding the antenna with more coil, we achieved a 500 μH inductance. This gave a much stronger starting signal which allowed for a single amplification stage.

This was a deliberate trade-off between a larger, heavier coil in exchange for a significantly smaller amplifier circuit.

Sensing: Infrared (Type)

Background

The rock emits infrared (IR) light in the form of a pulse-train signal. The body of the rock is porous, allowing the IR light to escape through multiple pores.

The emitted pulse-train follows a Poisson distribution, where the rate parameter λ depends on the type of rock. The probability distribution is given by:

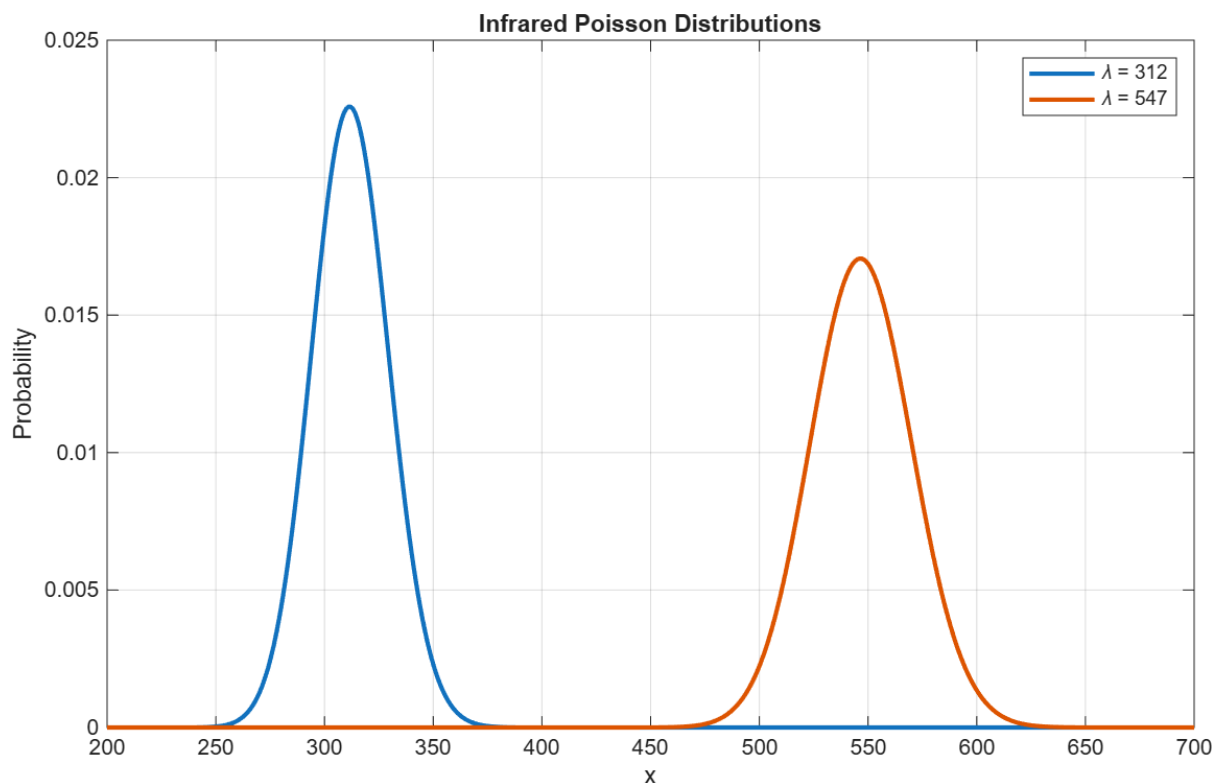
$$P(X = x) = \frac{e^{-\lambda} \lambda^x}{x!}$$

Note that $P(X)$ denotes the probability of x pulses occurring in that second.

This means the pulses are not evenly spaced in time and do not occur at exactly fixed intervals of λ , although pulses are more likely to occur at intervals closer to the expected average.

Each pulse has a width of approximately $50 \mu\text{s}$. Over a sufficiently long observation period, the average pulse frequency converges toward the corresponding λ value.

For the intended application, the different rock types have sufficiently separated λ values that the closest matching frequency can be reliably identified (e.g., 312 Hz vs. 547 Hz).



Poisson distributions with $\lambda = 312$ and $\lambda = 547$. Generated on MATLAB.

Sensor Selection

Initially, the **SFH300 phototransistor**, available in the lab, was used for testing. However, during evaluation it was found to be overly sensitive to ambient light and insufficiently sensitive to the IR pulse signal, resulting in a weak electrical response that was difficult to distinguish from noise.

Following this, the **TEFT4300 phototransistor was selected** and ordered. This sensor includes a built-in daylight-blocking filter and is specifically tuned for 940 nm IR light, making it significantly more suitable for reliable detection of the IR pulse signal.

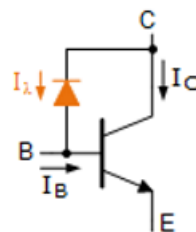
Photodiodes and Darlington Pair

A photodiode is a semiconductor device that can act as a light sensor. When incident photons with sufficient energy strike the depletion region of the junction, they generate electron-hole pairs within the silicon. The built-in electric field across the depletion region separates these charge carriers, producing a photocurrent approximately proportional to the incident light intensity.¹

$$I_{photo} \propto \text{Light intensity}$$

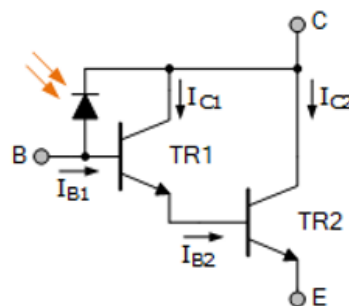
A phototransistor is a photodiode integrated with a bipolar junction transistor (BJT), in a CE configuration. The photocurrent generated by incident light acts as the transistor's base current, which is then amplified by the transistor's current gain.

Phototransistors produce significantly larger output currents than photodiodes, improving sensitivity. However, this increased gain comes at the cost of slower response times due to transistor charge storage and increased capacitance.



Circuit diagram of phototransistor

For applications requiring greater sensitivity, the output current can be further amplified using a Darlington pair configuration, where two BJTs are cascaded to achieve very high effective current gain with relatively low circuit complexity and cost.



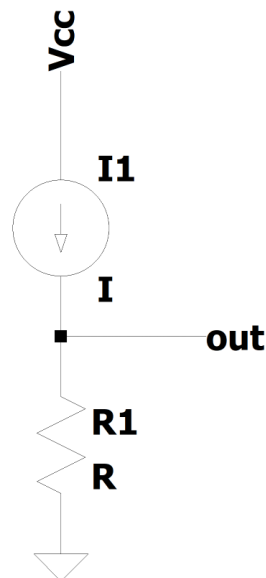
Darlington pair configuration

¹ <https://www.electronics-tutorials.ws/transistor/phototransistor-basics.html>

LTSpice Circuit Design

Initial circuit design – No rock casing

The initial design for the circuit was to reuse the light-sensing rover circuit developed during the lab skills sessions. When exposed to light the phototransistor acts as a current source. The resulting current signal can be passed through a resistor to convert it to a measurable voltage signal that can be processed by the microcontroller.



Phototransistor modelled in LTSpice as a current source

The voltage signal could be sampled by the microcontroller. The software counted the number of signal spikes received per second while ensuring that individual pulses were not double-counted.

The ADC sampling performance of the Adafruit Metro M0 Express was tested at 10-bit resolution. The maximum achievable sampling rate was approximately 35 kHz. In comparison, when using `digitalRead()` for 1-bit threshold-based detection, the sampling rate could exceed 100 kHz even within the standard Arduino framework.

We decided to continue using the default Arduino ADC functions for simplicity and maintainability, with the option of reducing ADC resolution if higher sampling speeds became necessary.

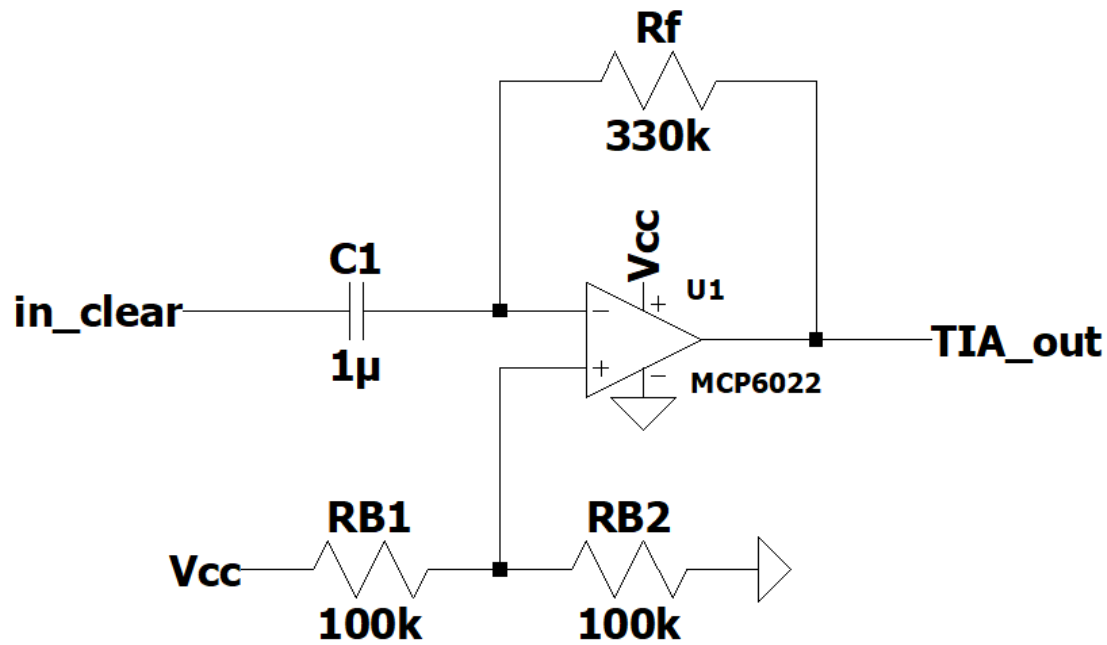
This approach worked well initially, without the rock casing and with the sensor placed directly above the emitter. The signal spikes were sufficiently large to be clearly distinguished from noise and reliably detected by the microcontroller.

However, when testing with the rock casing, the IR signal had to pass through the pores of the rock. This resulted in significantly weaker spikes that were much smaller and nearly indistinguishable from background noise.

Transimpedance Amplifier

We considered using a passive filter to reduce noise, followed by an amplifier. However, this distorted the signal making spike detection more difficult. Additionally, excess capacitance smooths the sharp edges of the pulse-train, causing slower signal decay which distorts pulse timing.

A transimpedance amplifier (TIA) could be used to convert the current signal into a voltage signal.



Transimpedance amplifier in LTSpice

The amplifier performed well in simulation. However, when implemented on a breadboard, the expected performance could not be reliably reproduced

In addition, accurately modelling the system in SPICE proved difficult, as the real-world signal was not a pure current source and included multiple noise components. These ranged from low-frequency mains interference to higher-frequency noise caused by parasitic inductance and electromagnetic coupling.

Unaware of these limitations at the time, a complete circuit design was developed based on the transimpedance amplifier approach. This design included an additional amplification stage, followed by a half-wave rectifier and a Schmitt trigger to produce a clean digital output signal.

Although the full system functioned correctly in simulation, the TIA stage could not be made to operate reliably in the physical implementation.

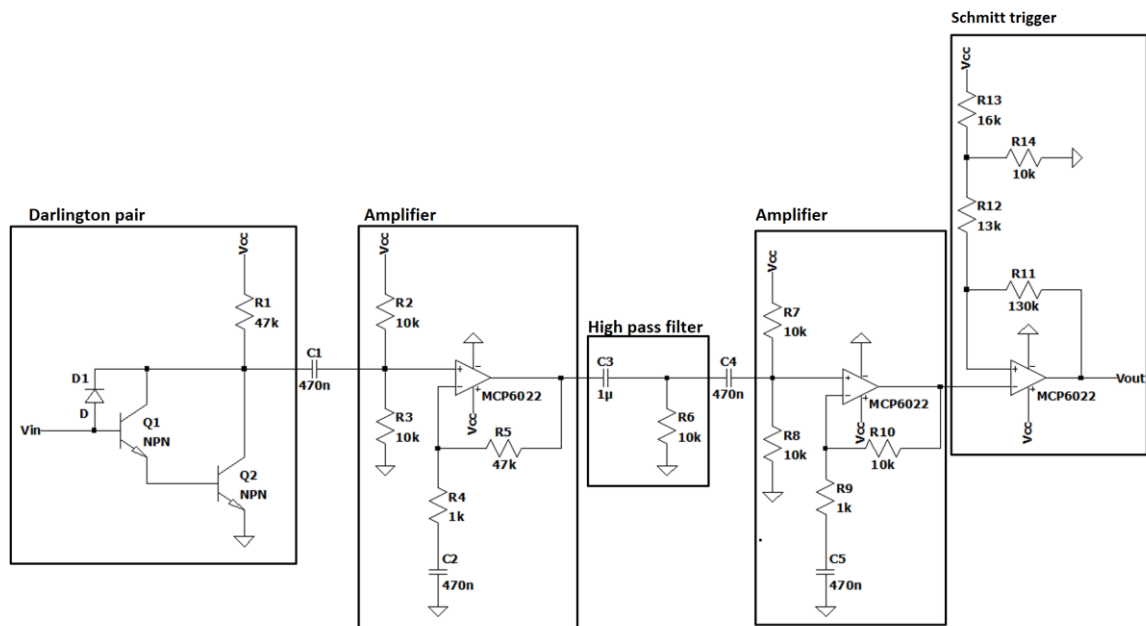
Within this design, both the half-wave rectifier was also evaluated, since the presence of capacitive elements introduced unwanted opposite polarity transients (dips and rises) that needed to be addressed.

Complete IR Circuit

From the previous iterations, we identified several key improvements.

- Capacitance in the signal path was minimised as much as possible to preserve the sharpness of the pulse edges.
- Non-inverting amplifiers were used, as they do not load the input directly but instead draw current from the biasing network, preserving signal strength.
- A high-pass filter was introduced between the two amplification stages to prevent low-frequency noise from being excessively amplified. A higher gain stage ($\approx 48\times$) was placed before the filter, followed by a lower gain stage ($\approx 11\times$) after it.
- The Schmitt trigger thresholds were adjusted to ensure the circuit was not triggered by noise while still reliably responding to the amplified signal.

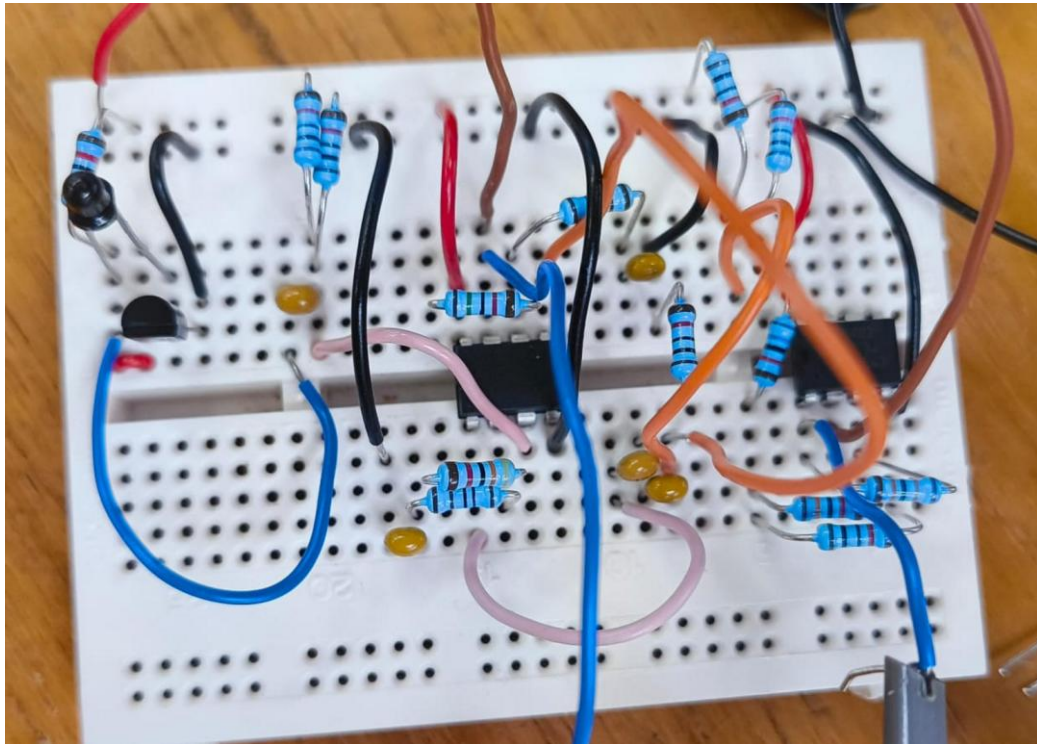
An additional improvement was made in the current-to-voltage conversion stage. Instead of using only a resistor at the phototransistor emitter, a Darlington pair based common emitter amplifier configuration was implemented. This provided higher initial gain at the sensing stage with minimal additional complexity, improving overall signal detectability from the outset.



Complete IR sensing circuit built in LTSpice.

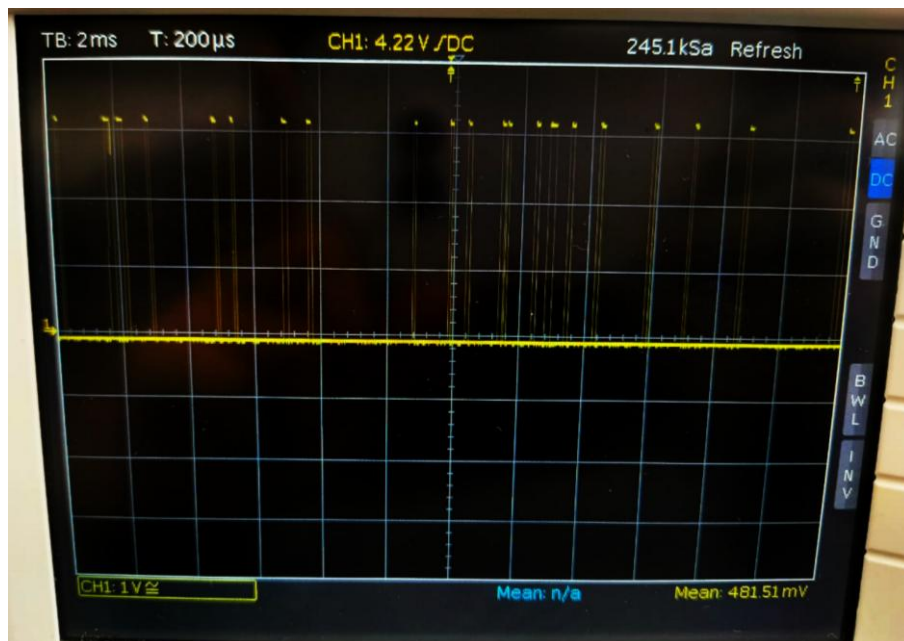
Implementation and Testing

This was the initial circuit built on a breadboard. It was later optimised and combined with other sensor circuits.



Initial physical implementation of IR circuit on a breadboard

We tested the circuit and obtained a clear reading with pulses being driven to 3.3 V.



Oscilloscope output of IR sensing circuit. Pulses are clearly visible.

Software

The infrared detection code continuously reads the digital output produced by the circuit for a duration of 1 second, while operating at a throttled sampling rate of approximately 50 kHz. Pulse counting is implemented using an `in_pulse` state variable to ensure that individual pulses are not counted multiple times.

The code counts the number of pulses in 1 second, this value is λ .

Since the rock type determination is based on a lookup with only two possible IR count values, 312 and 547, an additional function was implemented to call the pulse-counting function and return whichever reference value is closest to the measured result.

```
int infra_closest() {  
    int count = infra_actual();  
    return ((count - MID > 0) ? 547 : 312);  
}
```

IR λ value lookup function

Sensing: Ultrasound (Type)

Background

Two of the four rock types emit an ultrasonic signal at 40 kHz. Therefore, the classification only requires detecting the presence or absence of this signal. No amplitude or frequency analysis is needed.

This can be done using an ultrasonic transducer. If ultrasound is present, there will be a sinusoidal signal of frequency 40 kHz. If not, there will be a constant 0 V.

Transducer Selection

The table below summaries the available transducers. They all have a centre frequency of 40 kHz.

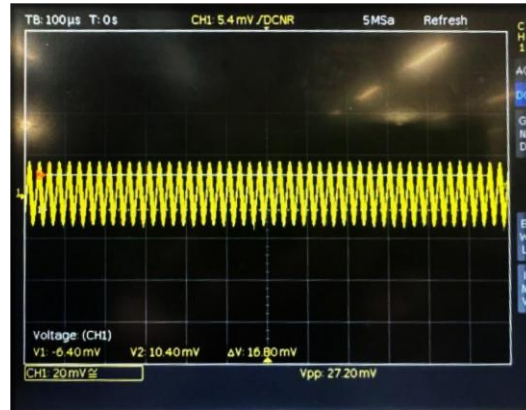
Transducer	Price (GBP)	Beam angle (°)	Sensitivity (dB)
400SR100	3.94	55	-65
400SR160	5.86	75	-70
MA40S4R	4.33	80	-63

We selected the **400SR100**, since it had a 55° beam angle. The angle is narrow, allowing us to selectively detect the ultrasound signal from the target emitter with minimal interference from other emitters outside the sensing range.

Circuit Design

We measured the initial signal from the transducer to be about **0.3 V**.

Therefore, an amplifier was required so the signal was large enough to be detectable by the Arduino.



Oscilloscope measurement of transducer output shows an amplitude of 27.20 mV

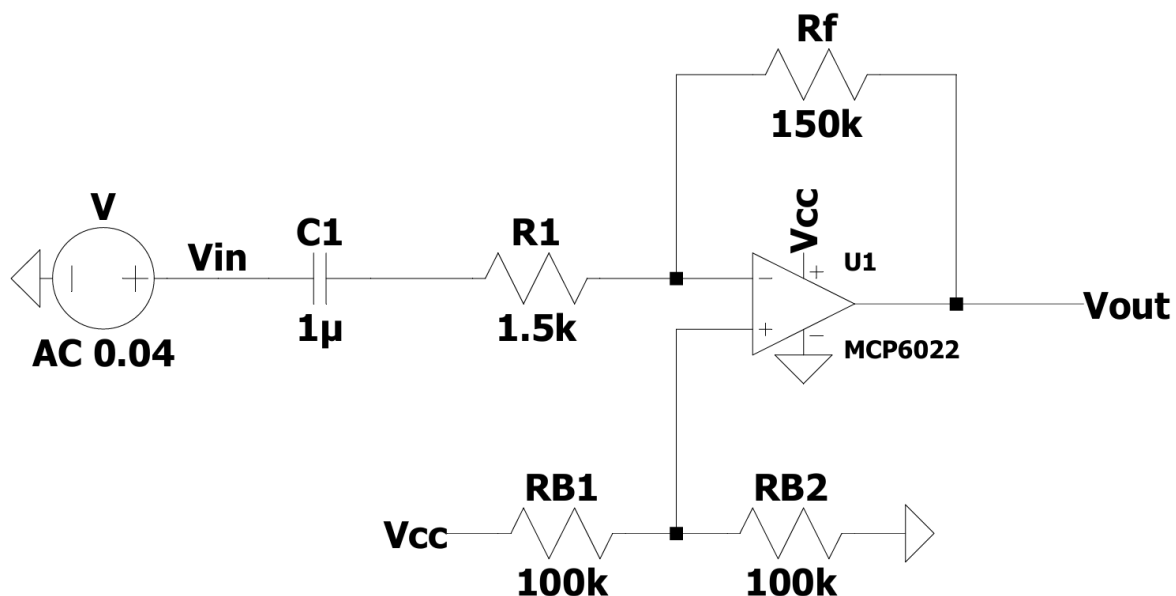
Initial circuit design

We initially set the voltage gain to 100x in an inverting amplifier configuration. With a feedback resistor R3 (150 kΩ) and an input resistor R4 (1.5 kΩ) we calculated gain as:

$$Gain = -\frac{R_f}{R_1} = -\frac{150}{1.5} = -100$$

We set an offset of 1.65 V, half way between the upper rail (3.3 V) and lower rail (0 V). This was done using a potential divider with resistors R1 and R2. The offset prevented the output exceeding the supplied voltage range and clipping.

A capacitor (C1) was added to block the DC offset.



LTSpice circuit of our initial amplification circuit. Vcc is 3.3 V.

When this circuit was built physically, the maximum achievable gain was 30. We achieved a peak voltage of **1.02 V** instead of the expected 3V.

This is not caused by the roll-off effect due to the gain-bandwidth product (GBW). The MCP6022 op-amp has a GBW of **10 MHz**. We can calculate the bandwidth (BW) using the following formula:

$$BW = \frac{GBW}{A_V}$$

$$BW = \frac{10,000}{100} = 100 \text{ kHz}$$

The amplifier bandwidth of **100 kHz** is well above the ultrasonic signal frequency of 40 kHz.

A single-stage amplifier would not be able to achieve the required gain of 100. Instead, a two-stage configuration could be used, each with a gain of 10.

However, with a simple amplifier alone, noise at other frequencies cannot be effectively filtered out, as the transducer still exhibits noticeable sensitivity outside the 40 kHz band. This indicates that a filter stage should be incorporated into the circuit.

Improved circuit

To make the circuit more sensitive to signals at 40 kHz, and less sensitive to other frequencies, we constructed a parallel RLC circuit with a resonance frequency of 40 kHz.

We used the formula for resonance, with $f_0 = 40 \text{ kHz}$, to determine a value for L and C .

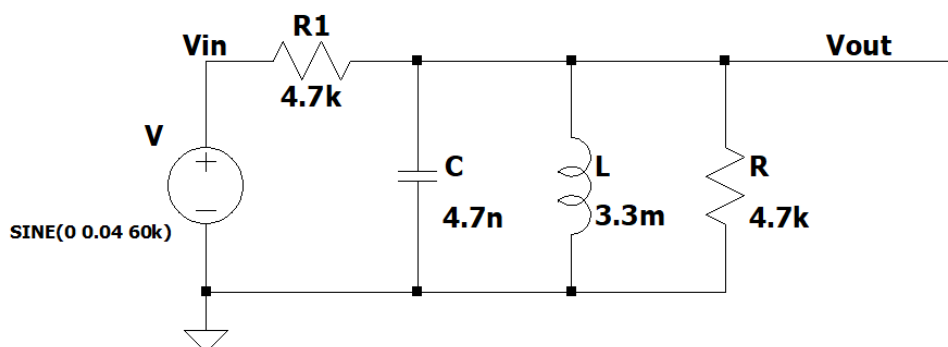
$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

We selected $L = 3.3 \text{ mH}$ and $C = 4.7 \text{ nF}$ as these values were available in the lab.

We selected a resistor of value $R = 4.7 \text{ k}\Omega$. We calculated the bandwidth using the following formulae:

$$Q = R\sqrt{\frac{C}{L}} \quad \text{and} \quad BW = \frac{f_0}{Q}$$

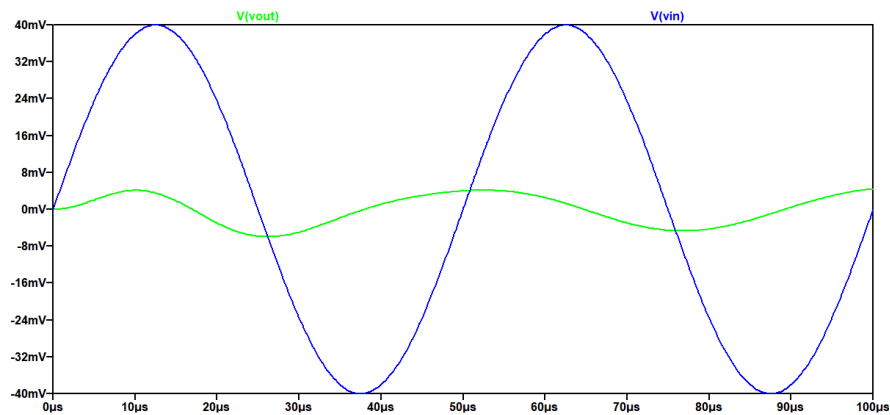
This gave a bandwidth, $BW = 7.2 \text{ kHz}$. This provides some frequency tolerance around 40 kHz, rather than filtering to a single strict frequency.



LTSpice circuit of the parallel RLC filter with $f_0 = 40 \text{ kHz}$

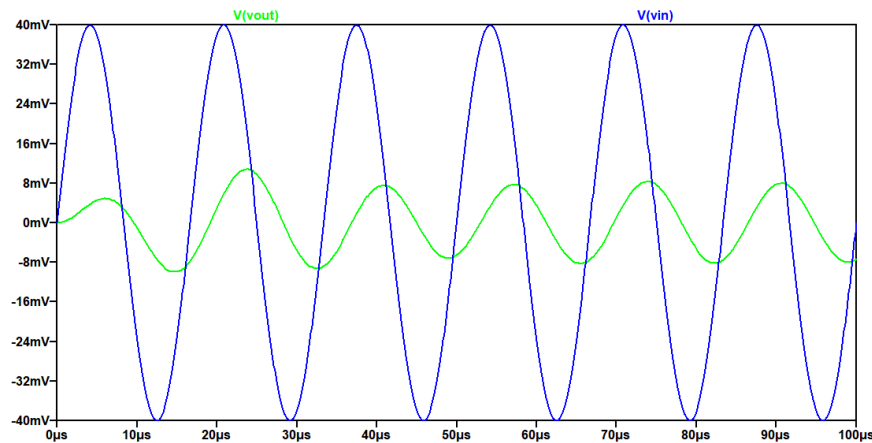
We simulated this circuit at frequencies 20 kHz and 60 kHz to validate filter behaviour. The input amplitude was set to 40 mV.

First 20 kHz, which have an amplitude of 0.71 mV:



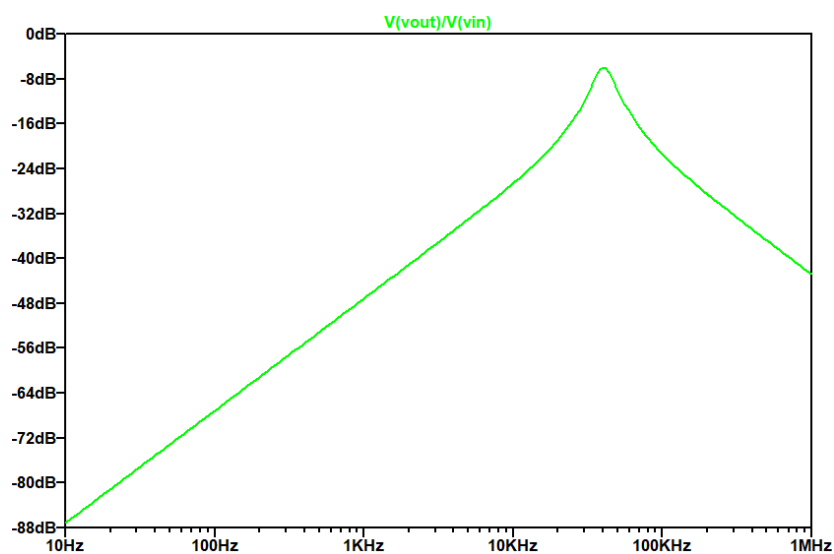
Output (green) and input (blue) at 20 kHz

At 60 kHz, the amplitude is 16 mV.



Output (green) and input (blue) at 60 kHz

We also plotted the gain of the RLC filter, with varying frequencies:



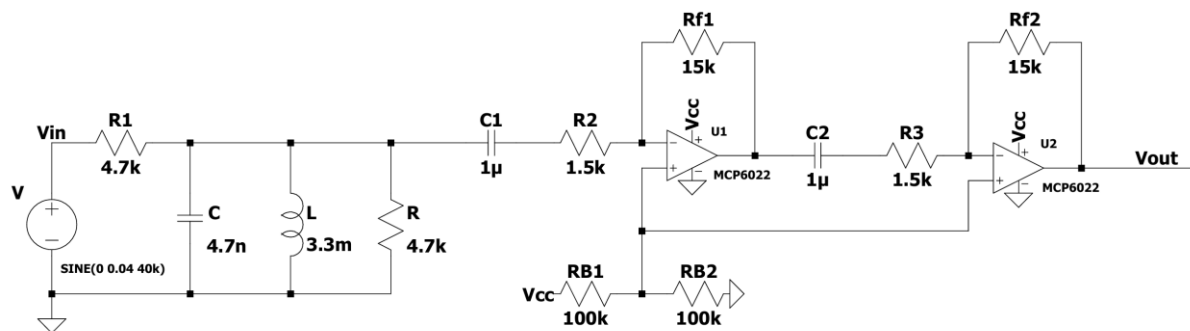
Resonance curve showing an AC frequency sweep of gain. Peak gain occurs at 40 kHz.

The resonance curve shows that maximum gain occurs at 40 kHz, confirming that the filter effectively attenuates noise at other frequencies.

We decided to replace the single-stage amplifier with a two-stage amplifier, each with a gain of 10, giving a total gain of 100. This ensures a higher operating bandwidth and mitigates the roll-off effect. We combined the two-stage amplifier with the RLC parallel filter.

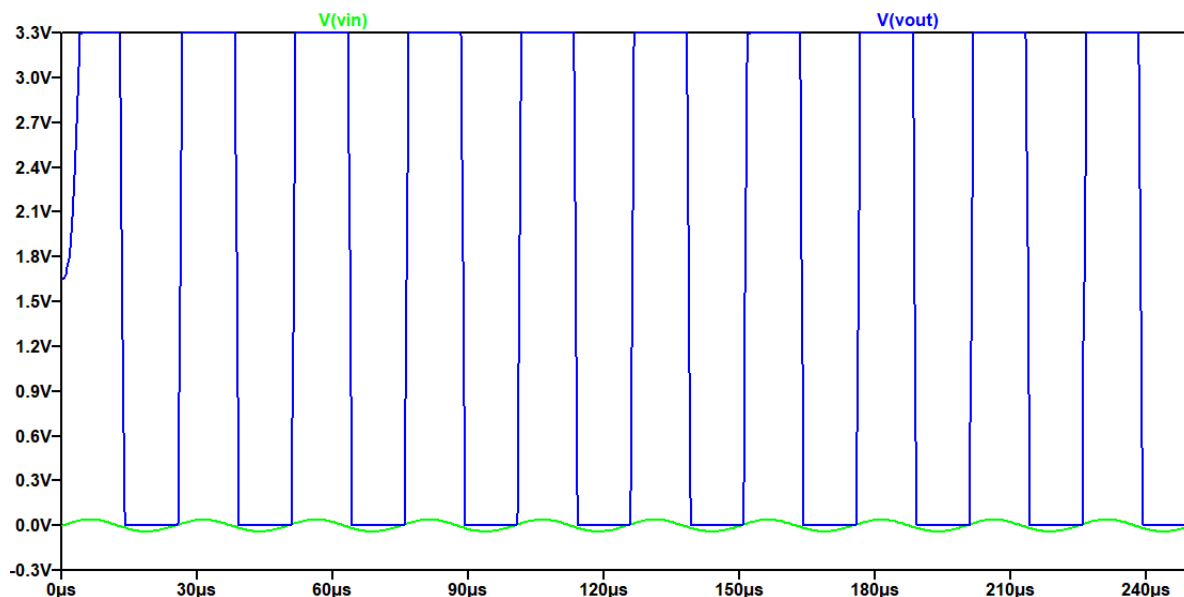
The complete circuit is shown below. Due to the amplification, clipping occurs at the op-amp's supply rail (3.3 V). Since the objective is only to detect the presence or absence of the ultrasonic signal, preserving the original waveform is not necessary.

Clipping here is actually beneficial, since it ensures that maximum amplification is reached and improves detection sensitivity.



Complete circuit for ultrasonic detection. Including RLC filter and a two-stage amplifier. The input from the transducer is modelled as a sine wave with amplitude 40 mV and frequency 40 kHz.

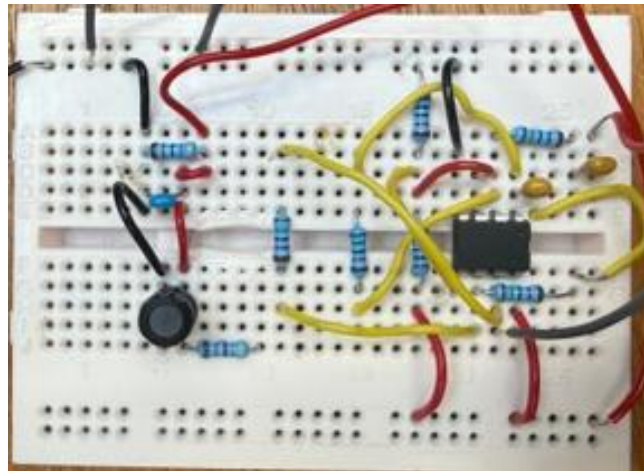
The output trace is shown below. The output clips at 3.3 V as described previously.



Output (blue) and input (green) traces of the ultrasound detection circuit

Implementation and Testing

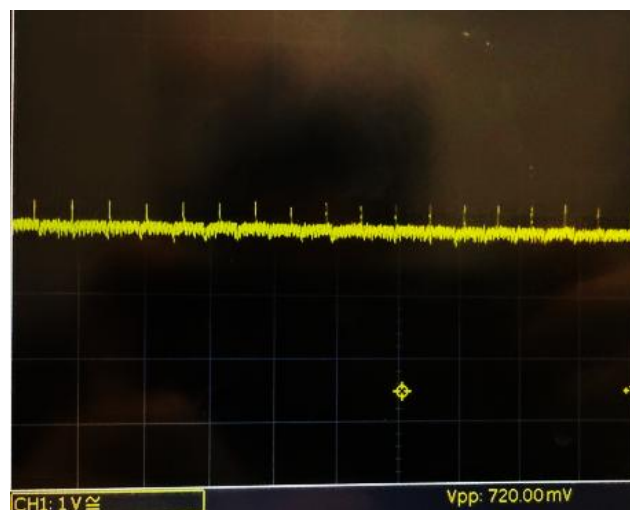
We constructed the circuit on a breadboard.



Ultrasound circuit constructed on a breadboard

When testing, the RLC filter did not behave as expected. It produced noise and reduced amplification by a factor of 10, making transducer signal hard to detect. Increasing amplification would also increase the filter's noise. Therefore, we discarded the filter and decided to use the Arduino's ADC threshold to filter low amplitude noise.

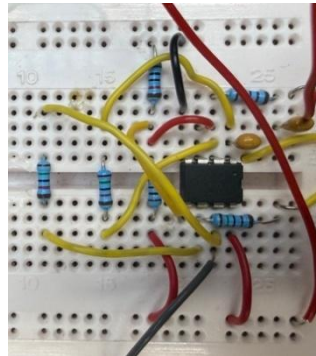
The following trace shows filter noise.



Noise from RLC filter

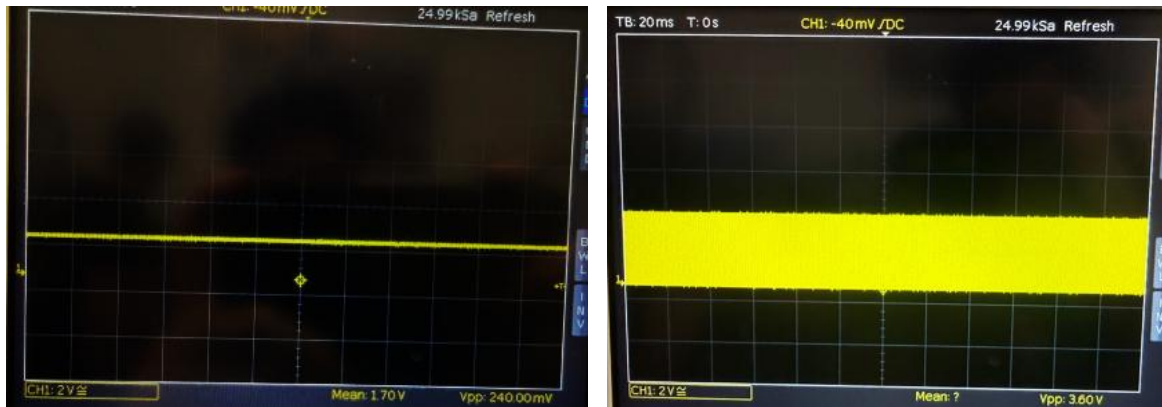
Complete Improved circuit

We removed the filter, keeping only the second-order amplifier. We assembled the new circuit on a breadboard.



Final, complete ultrasound circuit

The oscilloscope traces are shown for no ultrasound (no signal measured) and with ultrasound (40 kHz signal with 3.6 V amplitude).



Left: Output with no ultrasound. No signal detected

Right: Output with ultrasound, 40 kHz signal with 3.6 V amplitude

The effective vertical distance of the sensor is 16 cm. We will place the transducer 1 cm above the rock to obtain a better reading. At this distance, the signal can be detected with a 5 cm horizontal offset.

Software

A comparator was not required for the circuit since the threshold voltage for detection can be set directly in the Arduino via the Analog-to-Digital Converter (ADC).

The detection window is set as 1 second, and divided into 5 sub-windows of 200 ms each. Within each sub-window, 300 ADC samples are taken. A threshold ADC value of 800 is used, which corresponds to a peak-to-peak voltage of approximately 0.64 V. Above this threshold, the ultrasonic signal is considered detected.

A 12-bit ADC is used, giving an ADC resolution of 4095. Using the following formula the voltage threshold can be calculated:

$$V_{th} = \frac{ADC\ threshold}{ADC\ resolution} \times V_{supply}$$
$$V_{th} = \frac{800}{4095} \times 3.3 = 0.64\ V$$

Within each 200 ms sub-window, the maximum and minimum value of voltage are recorded. Samples are taken 300 times, and the highest and lowest values are used respectively. If the difference between the maximum and minimum voltage values reach the threshold value (0.64 V), ultrasound is recorded as being present for that window.

Ultrasound is marked as present for the rock when at least one sub window records an ultrasonic signal above the threshold.

Noise alone is insufficient to produce a peak-to-peak voltage exceeding the threshold, ensuring that any single threshold crossing is a reliable indicator of signal presence.

Sensing: Magnetic Field (Type)

Background

All rock types possess a magnetic field with a direction facing up or down. The magnetic sensor is responsible for determining field direction.

The image below shows the magnet that will be placed inside the rock.



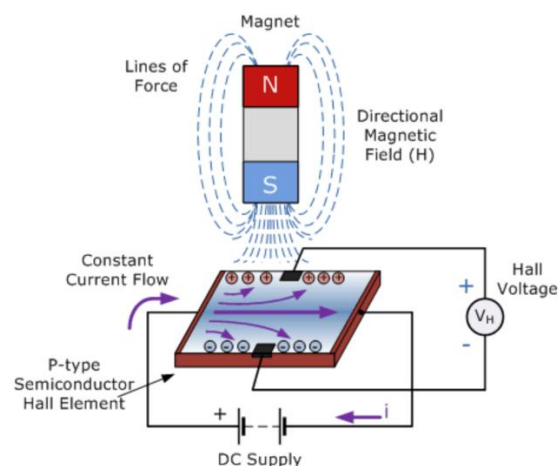
Magnet that will be placed inside the rock

Hall Effect Sensor

We chose to use a hall effect sensor to determine field direction.

The hall effect sensor is an electronic device which employs the hall effect, using magnetic flux to transport charge carriers to the edge of the semiconductor material. This causes a potential difference known as the hall voltage.

They are made of p-type semiconductors. When placed in a magnetic field, the field exerts a force on charge carriers (electrons and holes). Carriers each move to one side of the semiconductor causing a charge build up, and hence a potential difference. The Hall Voltage is this potential difference.²



Hall Effect with a p-type semiconductor material, as in a Hall sensor

² <https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html>

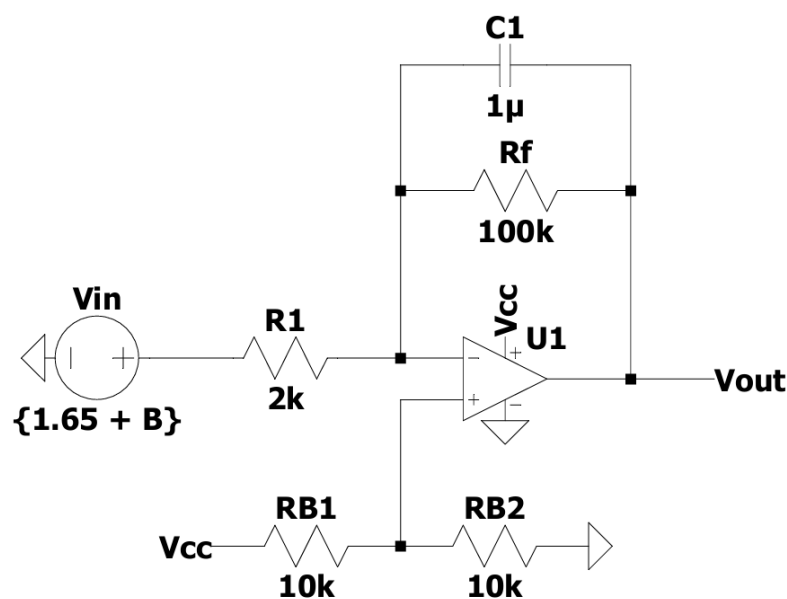
Circuit Design

Our approach was to use the Hall voltage to determine field direction. A logic HIGH (3.3 V) would indicate the field direction was up, a logic LOW (0 V) would indicate down.

We simulated the circuit in LTSpice, modelling the Hall sensor as a voltage source of value $1.65 + B$, where B is the magnetic flux density. B was set as a SPICE parameter.

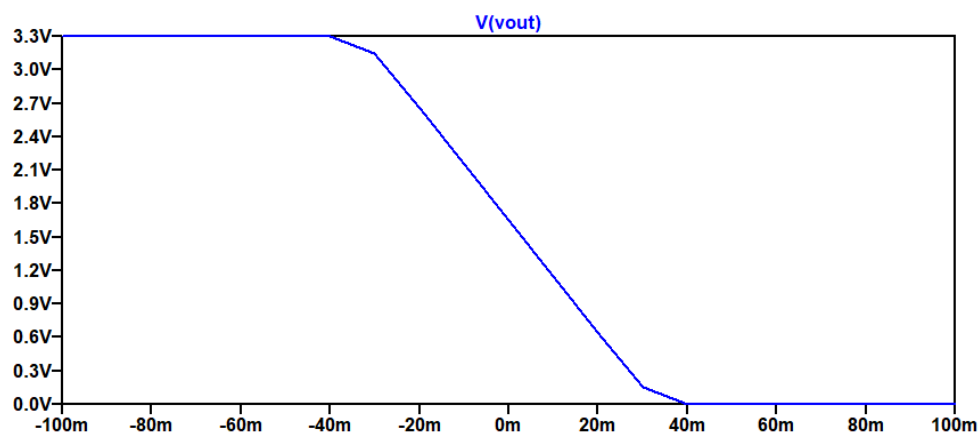
The initial signal from the sensor was weak, and so an amplifier was required for a detectable signal. An inverting amplifier configuration was used. This was a simple amplification setup that could be constructed easily. A bypass capacitor was placed parallel to the feedback resistor to form a low-pass filter, which attenuates high frequency signals from noise.

The Hall sensor output is DC.



Complete magnetic field detection circuit in LTSpice

The following output trace shows the output voltage with varying values for B , the magnetic field strength. When B is negative (polarity is south), the output is HIGH (3.3 V). When it is positive (polarity is north), the output is LOW (0 V). At $B = 0$ T, the output is 1.65 V.



Output trace of the field detection circuit plotted against magnetic flux density (B)

Between $B = -40 \text{ mT}$ and $B = 40 \text{ mT}$, the output is not clearly defined as HIGH or LOW. Therefore, we needed to set a threshold value. An appropriate choice was 1.65 V , as this was the output voltage at $B = 0 \text{ T}$.

Our original approach was to use a Schmitt trigger to drive outputs above 1.65 V to 3.3 V and outputs below to 0 V . However, during testing we found that our circuit was far less sensitive to the magnetic field. The sensor had to be placed very close to the rock to obtain an accurate reading.

Instead, we used the Analog-to-Digital Converter (ADC) and set the 1.65 V threshold through software.

Limitations

The Hall sensor must be oriented such that the magnetic flux lines pass perpendicularly through its active sensing face to achieve maximum sensitivity. If the sensing face was placed parallel to the magnetic field lines, the output would be negligible and field direction would be impossible to determine.

Therefore, the sensor must be mounted directly above each rock, vertically aligned with the magnet, to ensure the flux lines intersect the sensing face perpendicularly.

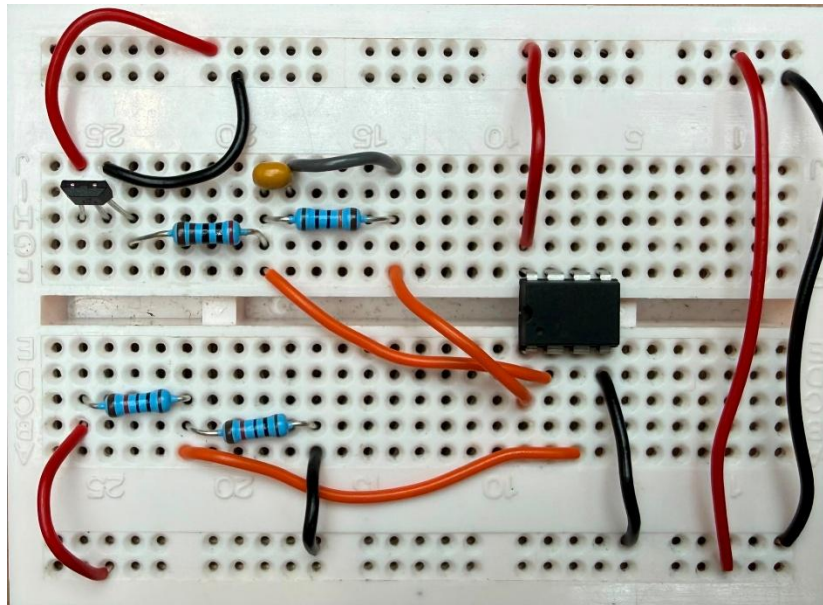
Selecting a value for gain

Initially the inverting amplifier had a gain of 100. We reduced this to 50 by changing the feedback resistor from $200 \text{ k}\Omega$ to $100 \text{ k}\Omega$. Decreasing gain reduced the effective distance of the sensor circuit from 6 cm to about $2\text{-}3 \text{ cm}$. Since the sensor had to be placed above the rock anyway (see above), this had no impact.

A lower gain reduced the likelihood of early saturation from noise or other magnetic sources, such as other rocks.

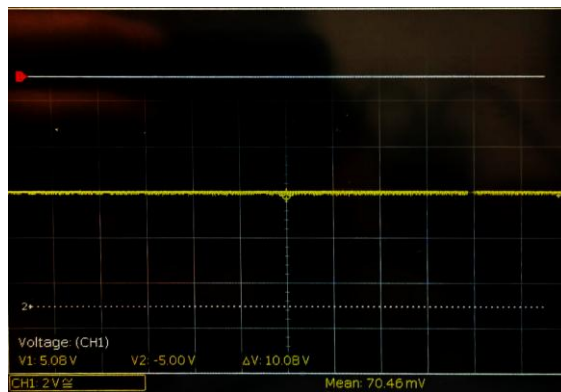
Implementation and Testing

This was the initial circuit built on a breadboard.

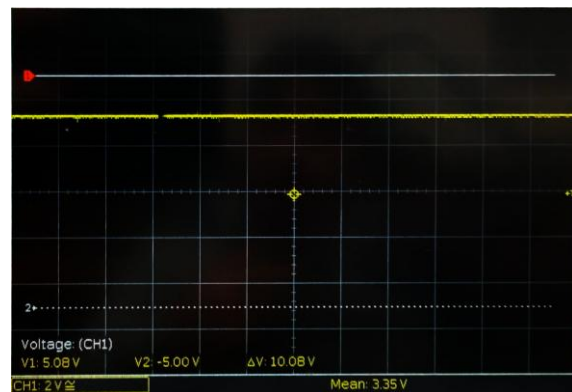


Magnetic field direction sensor circuit on breadboard

We tested the physical implementation of the circuit by orientating the test magnetic so its field was facing up and down. When the field was up, we measured an output of ~ 0 V. When the field was down we measured an output of ~ 3.3 V.



Left: oscilloscope output when magnetic field faces UP (mean value 70 mV)



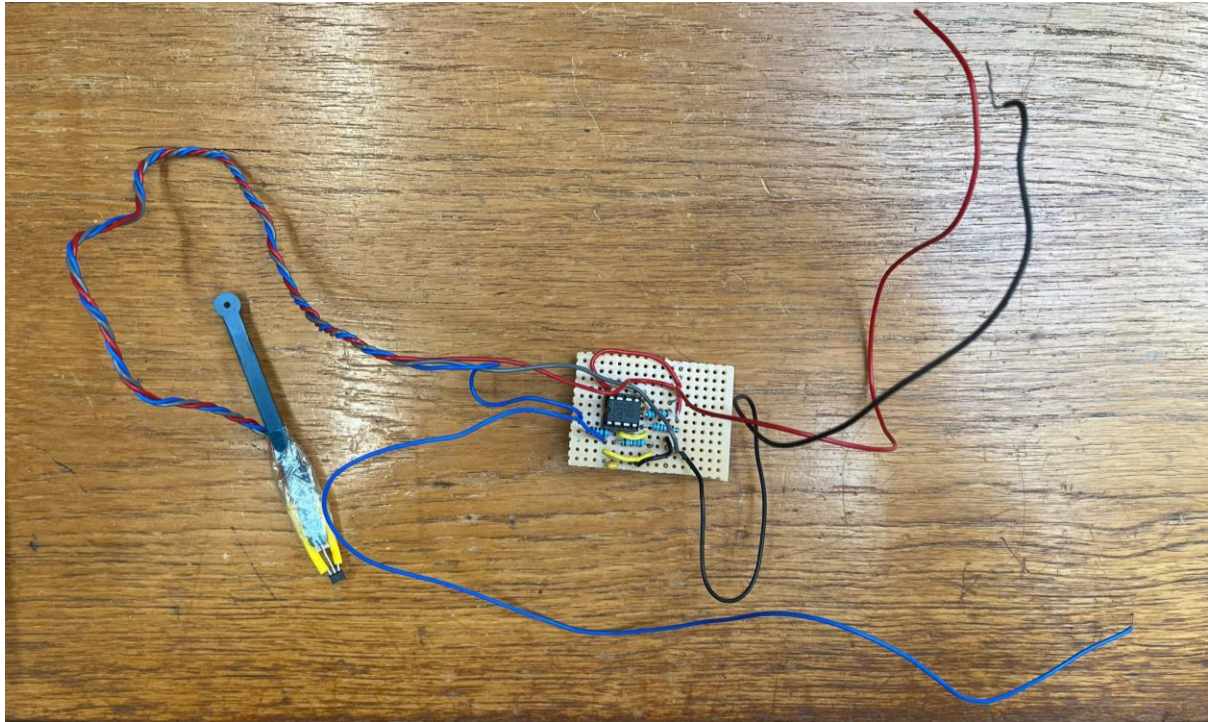
Right: oscilloscope output when magnetic field faces DOWN (mean value 3.35 V)

Prototyping board

Since sensor orientation was very important in order to get an accurate measurement, the sensor was attached to a servo that was provided as part of the EEEBug kit. This allowed us to control the orientation of the Hall sensor for each rock.

We soldered the circuit shown in the breadboard on prototyping board. This was done to reduce the space the circuit occupied in order to make it easier to attach to the rover in a way that allowed us to use the servo to control sensor orientation.

The complete circuitry is shown.



Complete magnetic field direction sensor circuit on prototyping board and attached to the servo

Software

The output from the circuit is fed into the ADC input of the microcontroller. The software compares the magnitude of the input to a threshold and determines which direction the magnetic field is facing.

`analogRead()` has a value for the magnitude of its input ranging from 0 to 1023, with a midpoint of 512. We selected a threshold of ± 200 in order to prevent false readings from other magnetic fields, such as from other rocks.

Input magnitude	Magnetic field direction
712	North (up)
312	South (down)

Rover Movement

Background

To control rover movement, the voltage supplied to individual motors can be regulated.

The system is required to support smooth bidirectional control (forward and reverse), alongside smooth and quick turning functionality.

A low-level firmware library can be implemented to abstract motor control into high-level movement functions. These functions are then used within a polling-based control loop, which interfaces with the networking subsystem to enable remote operation of the rover.

Hardware Components

The rover uses two motors and two wheels, allowing each side to be controlled independently. This differential drive configuration enables efficient turning, including on-the-spot rotation, by driving both wheels with equal magnitude but opposite direction, resulting in zero net translation.

For electrical control, the system uses the EEEBug PCB, which integrates power distribution and motor driver circuitry.

The board includes:

- ✓ a 6 V battery input,
- ✓ headers for V_{batt} ,
- ✓ a 5 V linear voltage regulator,
- ✓ regulated 5 V output headers.

A power switch is used to control the delivery of V_{batt} to the rest of the circuit.

Motor connections are routed through the PCB, where the positive motor terminals are connected to M+ through a 1 Ω series resistor. Both motor terminals are also decoupled to ground using 10 μ F capacitors for noise suppression.

Motor control is implemented using the **TC78H620FNG** motor driver IC, which provides two independent H-bridge circuits for controlling the left and right motors.

The PCB manages motor direction control through a digital state machine, while motor speed is controlled independently using PWM signals applied to the EN pins of the H-bridge driver.

The board exposes MR+, MR-, ML+, and ML- pins. Depending on the state of the DIR control signals, M+ is routed through the appropriate enable path, allowing current to flow through the motor via the series resistor and filtering capacitors, enabling controlled bidirectional motor operation.

In addition to the electrical components, the mechanical system consists of EEEbug DC motors, wheels, and four 1.5 V cells connected in series to provide a total battery voltage of 6 V.

This diagram summarises the rover circuitry.

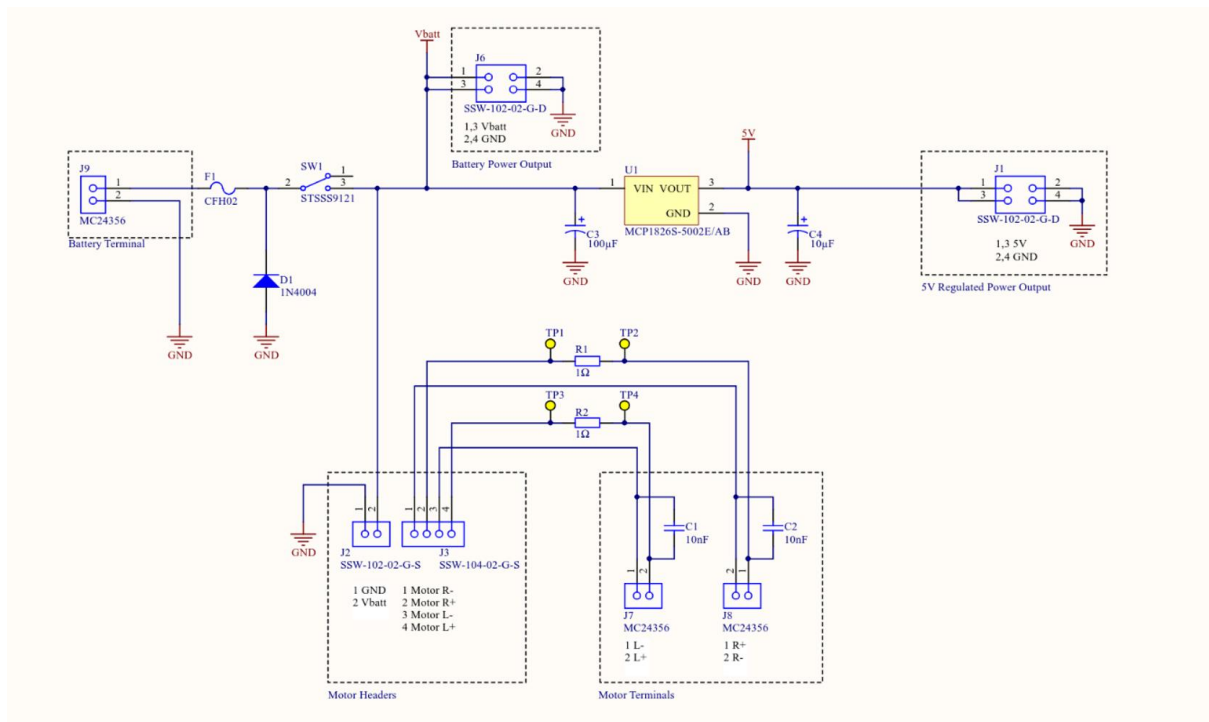


Diagram of rover movement operation

Below is an image of the motor driver component.



Motor driver component. See left and right EN and DIR pins.

Pin	Purpose
EN	Motor Speed
DIR	Motor Direction (Forward or Backward)

Software (Movement Library)

The low-level functions were first implemented using the Arduino framework functions `digitalWrite()` and `analogWrite()` to set the voltages at individual pins for specific motor-control tasks.

These tasks included setting both motors to a proportion of their maximum speed, driving the motors in equal or opposite directions, controlling each motor independently, and stopping the motors entirely.

`digitalWrite` sets the motor direction:

```
digitalWrite(LEFT_DIR, FORWARD);  
digitalWrite(RIGHT_DIR, REVERSE);
```

`analogWrite` sets the motor speed:

```
analogWrite(LEFT_EN, speed);  
analogWrite(RIGHT_EN, -speed);
```

The previous code is used to make the rover turn sharply left. It is used here as an example only. In reality, to turn left only `LEFT_DIR` is set to forward and `RIGHT_EN` is set to 0.

Joystick Control

The rover takes 2 forms of input for control, described by the JSON field `IN_METH`. These are a D-pad input or joystick input.

A function was then implemented to process a JSON object containing the input and joystick fields as arguments, enabling overall movement control through a single function call.

```
void Remote_movement(String JSon){  
    if (extrJS("IN_METH",JSON) == "DP")  
        remote_drive(Mv_in, Pr Mv_in);  
    else  
        analog_drive(extrJS(StickX), extrJS(StickY));  
}
```

Function responsible for handling movement control from joystick or button inputs.

Since the input mechanism uses an analogue joystick, the lower-level input can be represented as a two-dimensional vector whose components range from -1 to +1. The input is normalised to a circular domain so that diagonal joystick movements are scaled appropriately and constrained to produce smooth motion.

The control logic provides one motor with the sum of the joystick inputs `StickX` and `StickY`, while the other motor receives their difference. This achieves smooth and intuitive joystick-based control of the rover.

Implementation and Testing

Integration with other subsystems

Integration of the movement subsystem was straightforward due to the modular structure of the software. The codebase was organised into separate C++ libraries, with each subsystem implemented independently and included in the main Arduino .ino sketch through header files.

This modular approach allowed the final .ino file to remain minimal, with subsystem functionality exposed through clean interfaces. As a result, the main loop primarily consists of high-level function calls, including the Remote movement handler.

Testing and Validation

For testing the movement controls, a minimal version of the code was developed, including only the networking and movement subsystems.

The rover was then driven over the artificial moon terrain to evaluate its basic locomotion performance under representative conditions.

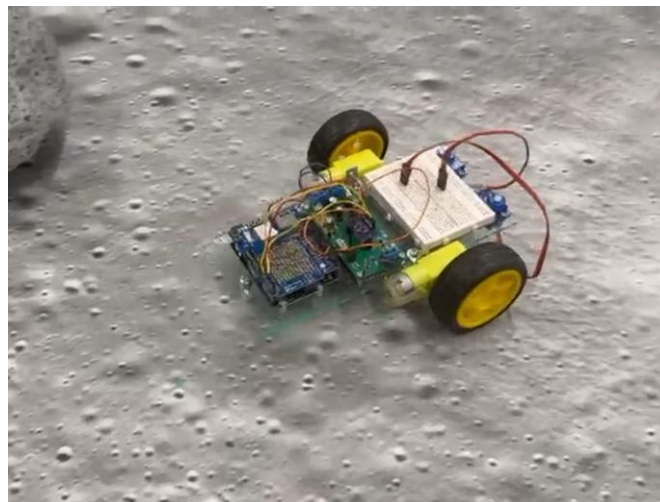


Photo of working rover on artificial lunar surface

Networking

Requirements

The objective of this subsystem was to establish communication between the microcontroller and the user's device.

The user interface needed to support sending instructions to the rover for movement control, as well as receiving and displaying rock type classification and age determination. The UI also needed to display operational status of the rover.

HTTP Setup

The **WiFiWebServer** library, recommended by the department, was used to implement the networking functionality. Using the code provided as a reference, additional request handlers were implemented for the server root page.

The microcontroller hosts an HTTP web server on the static IP address assigned to the group, following the format: `192.168.0.<group_number + 1>`

Initially, the root endpoint was intended to serve the complete webpage directly from the microcontroller. However, this approach proved impractical due to memory limitations. Since the HTML page, would need to be stored as a C-style character array, the webpage content would occupy a significant amount of the microcontroller's limited memory.

We also considered hosting separate web servers on both the rover and the user's device. However, this approach was unnecessarily complex for the project requirements.

The final architecture consisted of an HTTP server hosted on the rover, together with a locally stored webpage running on a device connected to the same network (EEERover), such as a laptop. The webpage communicates with the rover by sending HTTP `POST` requests to transmit control commands and HTTP `GET` requests to retrieve telemetry data.

By storing the webpage locally rather than on the microcontroller itself, the need to store large HTML, CSS, and JavaScript strings in the microcontroller's limited memory was eliminated, which also improved system stability.

HTTP Software

We aimed to encapsulate all networking setup functionality within a single high-level function that could be called once in the Arduino sketch during `setup()`.

We created a function `setup_wifi()`. This handled the setup of our HTTP network.

This function was responsible for:

- ✓ Waiting until the microcontroller successfully connected to the network.
- ✓ Stalling execution if the networking hardware, provided through the WINC1500 Wi-Fi shield, was not detected.
- ✓ Configuring the static IP address assigned to the group.
- ✓ Registering handlers for different HTTP request types, including POST, GET, and OPTIONS, across multiple server endpoints. For example:
 - o for the root handler,
 - o input for processing user control inputs, and
 - o data for transmitting telemetry data.
- ✓ Initialising and starting the HTTP web server.

HTTP Endpoints

The available HTTP endpoints were:

Endpoint	Request Type	Purpose
/		Returns nothing
/input	POST	Handles input data from user interface
/data	GET	Returns rock age and type to user interface

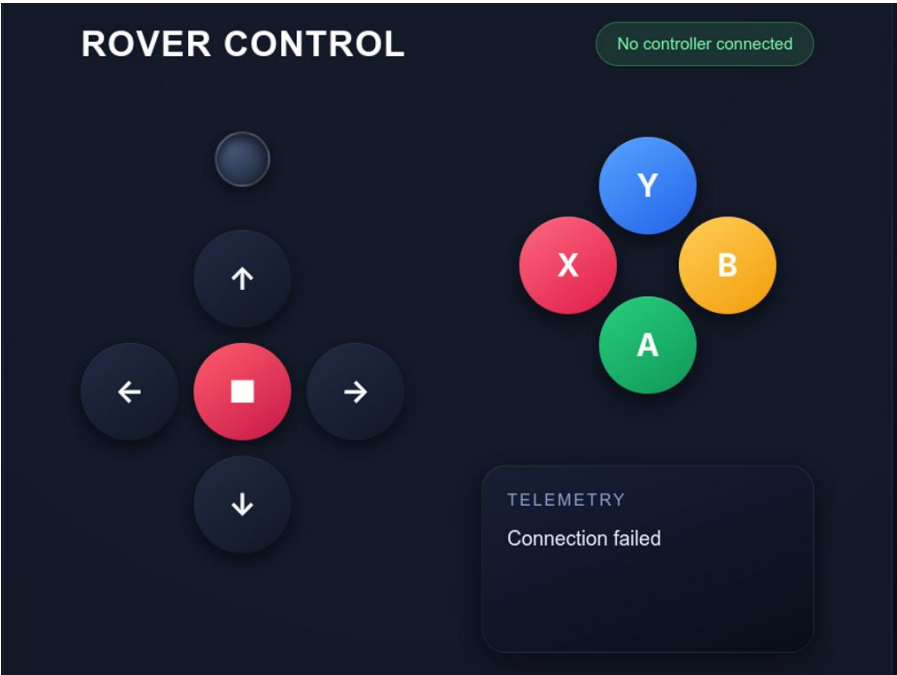
User Interface

The webpage was required to support both the transmission of HTTP `POST` requests and the display of telemetry data.

This functionality was implemented using on-screen control buttons, a joystick indicator, and a telemetry text box that periodically polled the `/data` endpoint every 0.5 seconds to retrieve updated rover information.

For rover control, multiple input methods were supported to improve usability and accessibility.

Users could control the rover using on-screen buttons with a mouse, keyboard inputs (WASD), or an Xbox game controller. As a result, the webpage interface was designed to visually mimic the layout of a standard game controller.



User Interface for rover control

The JavaScript implementation for the webpage included event listeners for button presses, controller connection and disconnection events, and a `sendPost` function responsible for constructing the JSON payload transmitted to the microcontroller.

Event Listener	Purpose
keydown	Keyboard key pressed. Identify key and determine instruction.
keyup	Keyboard key released. Stop sending movement instructions for direction associated with the WASD key.
click	Mouse click for handling onscreen button press.
gamepadconnected	Xbox controller has connected to UI. Store ID.
gamepaddisconnected	Xbox controller disconnected. Reset ID variable.
load	On page load, query if controller is already connected.

Optimisations

Several optimisations were made to reduce unnecessary network traffic.

- Sending requests only when current control state differs from previously transmitted state and throttling requests in case of continuous stick input.
- Game controller input is handled through continuous polling, since controllers do not provide event-driven updates and instead expose their state to the browser on request.
- Joystick input is normalised and constrained to a circular domain to ensure consistent scaling and smoother control response, as opposed to the default rectangular input range.

Communicating Instructions and Data

There were 2 streams of communication:

- ✓ **Movement instructions:** from user interface to rover
- ✓ **Rock age and type:** from rover to user interface

Movement instructions (UI -> Rover)

There were 2 types of inputs responsible for movement instructions (D-pad and Joystick).

Pressing a button or a changing stick position sends POST requests with JSON containing:

- Button pressed ("input")
- Joystick X value ("StickX")
- Joystick Y value ("StickY")

```
POST → ▶ Object { input: "BB", stickX: 0.04, stickY: -0.06 }
POST → ▶ Object { input: null, stickX: 0.04, stickY: -0.06 }
POST → ▶ Object { input: "BK", stickX: 0.04, stickY: -0.06 }
POST → ▶ Object { input: null, stickX: 0.04, stickY: -0.06 }
POST → ▶ Object { input: "ST", stickX: 0.04, stickY: -0.06 }
POST → ▶ Object { input: null, stickX: 0.04, stickY: -0.06 }
POST → ▶ Object { input: null, stickX: 0.03, stickY: -0.04 }
```

JSON payload sent to HTTP server via POST request

The microcontroller receives the JSON payload as a raw text string.

The `handleInput()` method is responsible for parsing this data and extracting the relevant fields into global variables.

These parsed values are then passed to the movement and detection APIs for further processing and execution of the required actions.

Rock Data (Rover -> UI)

When the microcontroller receives a GET request from the webpage to the `/data` endpoint, the `handleData()` function is invoked.

This function constructs a plain-text payload containing the rock's age and type, along with any other relevant telemetry information, and sends it back to the webpage.

The webpage then parses this response and displays the updated telemetry data to the user.

```
String text = "";
text += "Dir: ";
text += direction_query();
text += "\nAge: ";
text += lastAged;
text += "Type: ";
text += RockType();
```

Assembly of the plain-text payload sent to the UI. Executed as part of the `handleData()` function

Evaluation: Attempted Upgrades

Dual-MCU Architecture

Since the analogue stick input is continuous, it can generate a high rate of POST requests, potentially overwhelming the web server with frequent updates.

One proposed solution was to split the system across two microcontrollers. In this architecture, one MCU would handle rover operations such as movement and sensing, while the second MCU would host the web server and act as a mediator between the user interface and the rover control logic.

This separation could reduce load on the main control MCU, as the networking workload would be isolated. However, this approach was ultimately deemed unnecessary and overly complex for the requirements of the project.

The issue of excessive request frequency was instead addressed more simply by throttling POST requests directly on the webpage.

WebSocket Upgrade

Since continuous bidirectional communication is required between the rover and the user interface, upgrading the system to use a WebSocket connection initially appeared advantageous.

Compared to HTTP polling, WebSockets would reduce microcontroller load by eliminating repeated request–response handshakes. After the initial connection setup, data could be exchanged continuously without explicit requests from either side.

This approach would reduce latency, as the client and server could transmit data whenever ready, instead of relying on periodic polling cycles.

However, for the Adafruit Metro M0 Express, available WebSocket libraries were too resource-heavy or incompatible with the platform.

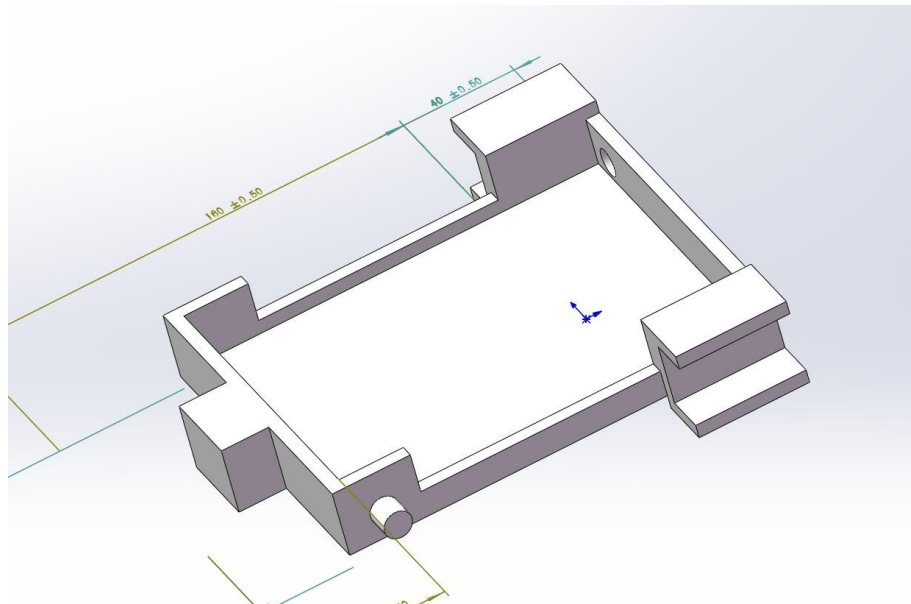
As a result, the decision was made to retain the existing HTTP-based request–response architecture rather than adopting WebSockets.

Chassis Design

Chassis

The chassis is the main structural platform of the rover. It supports the drive system, electronics and detection tower.

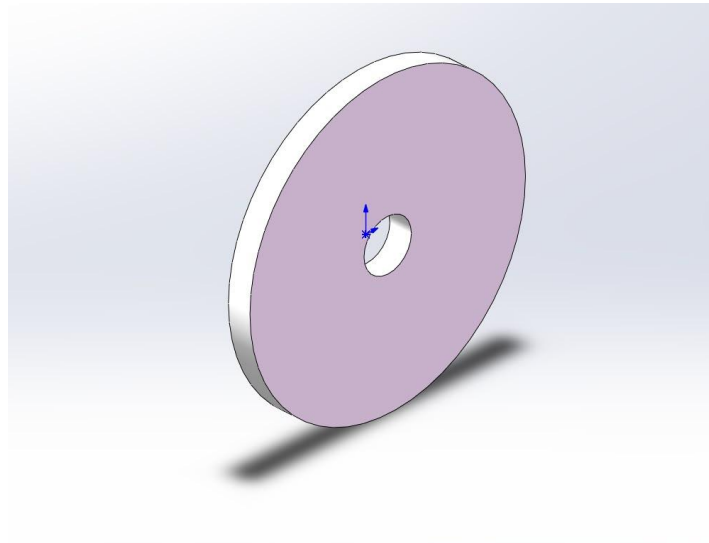
At the rear, small platforms mount the drive motor, with two holes behind the mount, allowing motor wires to pass through. The large central compartment houses the main electronic systems, providing space for breadboards, prototyping boards, and a microcontroller. At the front, a large circular mounting hole accommodates the detection tower's support shaft.



CAD of double-layer chassis

Auxiliary Wheel

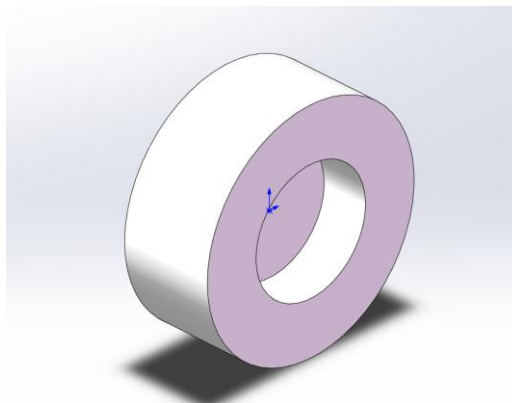
The auxiliary wheels have the same diameter as the drive wheel, ensuring that the rover remains level. This arrangement provides a balanced platform for sensors.



CAD of auxiliary wheel

Wheel Stopper

Additional wheel stoppers fixed on two axles to keep the wheels securely in place.



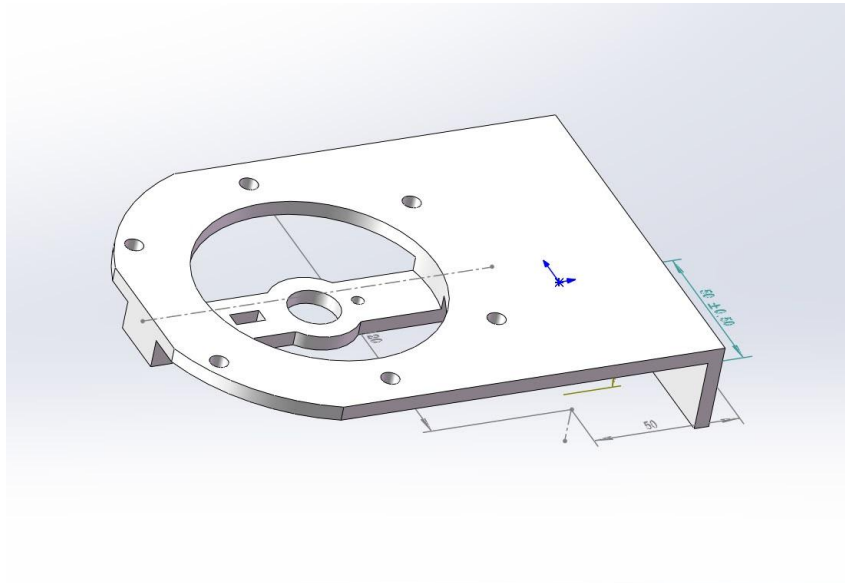
CAD of wheel stopper

Detection Tower

The detection tower supports the sensing systems. It mounts to the chassis through a central support shaft that fits into the front mounting hole. The tower is raised above the chassis to maintain an appropriate sensing distance.

A circular mounting area on top holds the radio detection antenna. Multiple holes around the platform allow the antenna to be securely fixed using brackets.

The lower sensing platform contains a central opening for the ultrasonic transducer and a smaller one for the infrared sensor. These are placed lower because they have a smaller effective distance.

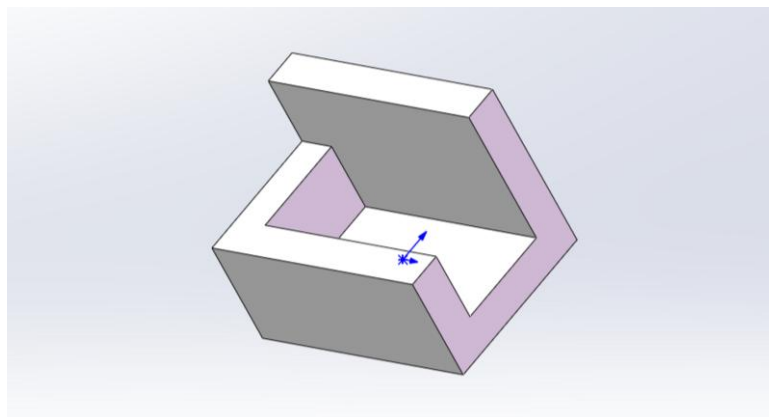


CAD of detection tower

Rotating Sensor Mount

A rotating magnetic sensor mount is designed to support the magnetic detection system. The component acts as a housing for a small DC motor mounted on the detection tower. The Hall-effect sensor is attached to an axle connected to the motor.

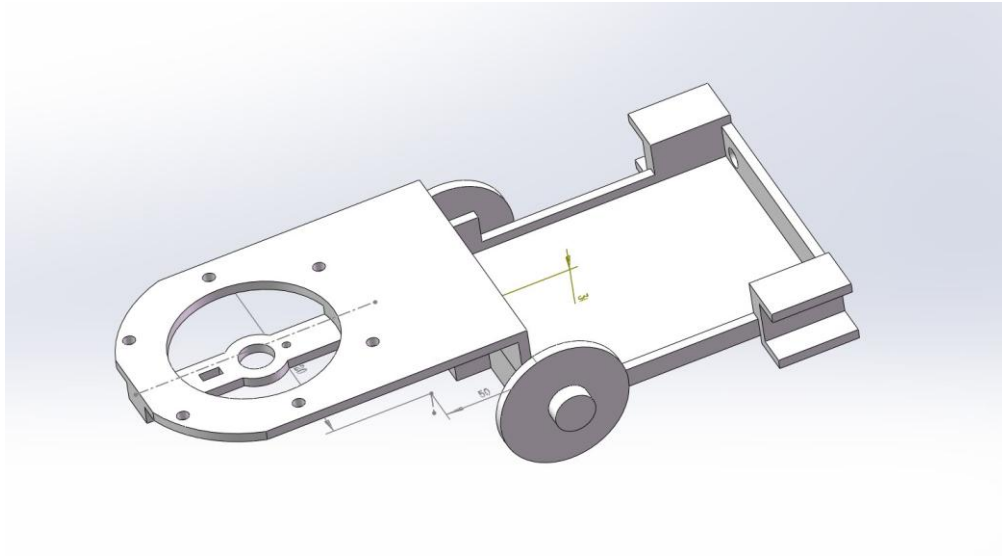
During operation, the motor rotates the sensor. See magnetic field section of the report for detail on why this is done.



CAD design for the rotating sensor mount

Complete Design

The final CAD design for the rover is shown below, along with the final 3D printed parts



CAD of complete rover

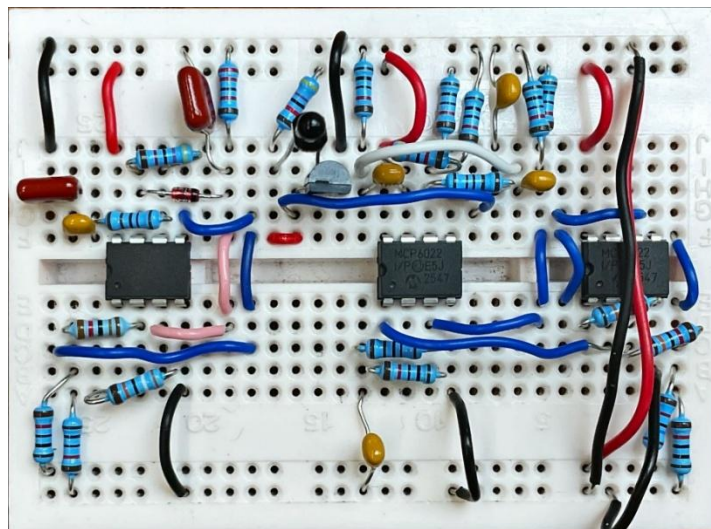
Subsystem Integration

Type Determination

The type determination logic was implemented as a simple lookup table with an additional “no match” option for invalid readings, allowing the user to rerun the sensing subroutine after repositioning the rover if necessary. This is done by the networking subsystem.

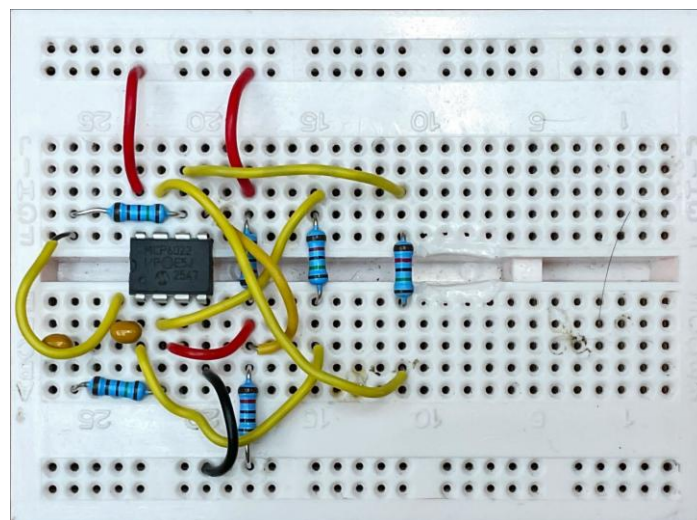
While all 3 sensor circuits are used, for each rock type the system only requires 2 of 3 sensor inputs to determine type. This ensured redundancy.

Radio/Infrared Integrated Circuit



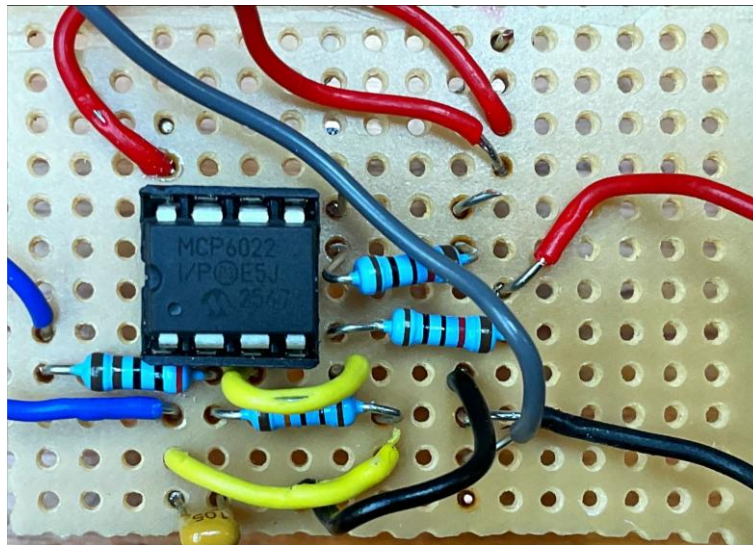
Integrated Radio/Infrared circuit

Ultrasound Circuit



Ultrasound circuit

Magnetic Circuit



Magnetic circuit

Next steps

Our next step is to assemble the respective 3D printed components, integrate the sensors onto the chassis and practise rover operations ahead of demo day.

Evaluation

Comparison of progress against brief

Brief Requirement	Our progress
Age detection: decode each rock's age from an 89 kHz ASK radio signal carrying ASCII characters in UART packets	Met – the antenna and receiver circuit decode the age reliably
Infrared detection: distinguish between the two infrared pulse rates (312 s^{-1} and 547 s^{-1})	Met – the final circuit produces clean pulses that can be matched to the nearest rate
Ultrasound detection: detect the presence or absence of a 40 kHz ultrasonic signal	Met – the final circuit reliably detects signal presence
Magnetic field detection: determine whether the rocks magnetic field points up or down	Met – the final circuit correctly determines magnetic field direction
Rock type classification: combine the magnetic, ultrasound and infrared readings to classify each rock	Met – working code combines sensor reading to classify each of the rocks correctly
Remote navigation: remotely drive the rover and receive data back over a network	Met – the rover drives successfully over the artificial lunar surface under remote control
Weight limit: keep the complete rover under 750 grams	Met
User interface: provide a clear, easy to use interface and controls	Met – multiple input methods provide a clear, accessible control system

All core requirements of the brief were successfully met. The completed rover navigates the artificial lunar surface under remote control and correctly classified each rock by both its age and its type. It is operated wirelessly through a simple interface. Together all the subsystems work as one complete, functioning rover that meet the full design specification.