Imperial College London

Team Crossobamon

EERover Report

Jacob Jenner - 01492281
Arman Fidanoglu - 01512561
Rhys Johnson - 01526052
Adrian Kwan - 01507296
Justin Kim - 01514370
Vasilis Manginas - 01542774

Word Count: 6637

‘

Contents

‘

Abstract

The aim of this report is to explain the design process and technical progress of Team Crossobamon's work in developing a rover in line with the EERover project brief.
The remotely controlled rover must be able to explore a forest and identify unusual lizards through the reception and processing of signals emitted by them.
The description of the design process focuses on how high-level design ideas were first created, how they were subsequently refined into detailed design ideas, what was learned through implementation of these ideas in the prototyping phase and how they have developed further since their original conception.
The account of the team's technical progress features the specifications of the sensors designed in order to detect the unique signatures of the jungle lizards. The interface used to control the rover and receive data from it is also explained alongside the intelligence operations of the rover used the process this data.
To date, all project work has been influenced by key limitations such as budget (limited to £50), size and weight. The influence of these factors is therefore outlined as well.

‘

Introduction

Technical Problems to be Solved:

Problem: A rover is required to explore the rainforests to identify a lizard's radio wave, infrared, acoustic signals, magnetic field patterns and subsequently its species with the major features as follows:

- The rover will need to be able to detect signals emitted by the lizards and use the information to classify the lizards by species.
- There are four types of signal: Radio (AM with UART encoded messages), Infrared, Acoustic and Magnetic.
- The rover must have some way of processing the data in order to determine what each signal is.
- The rover will need to have an interface in order to communicate to the user what lizard it has found.
- The rover is limited by size and weight. It must weigh less than 750g, while also being large enough to carry all of the necessary breadboards for the circuitry. A budget of £50 is also in place, meaning that no components can be too costly.
- Terrain is also an issue, the rover must be able to traverse the jungle floor, hence it will need sufficient clearance from the ground.

‘

Product Design Specification (PDS):

In order to create the initial product design specification, we had to select the most important elements to consider, and then define the relevant characteristics that the rover should have for each individual element.
This PDS contains information on the most important elements. For more elements see Appendix 5.

Performance

- Variable speed (Top Speed ~ 0.5 ms^{-1}?)
- Electrically powered by DC battery source
- Necessary sensors for:
    - Radio signals
    - Acoustic signals
    - Infrared Signals
    - Magnetic Fields
- Able to filter out background noise while measuring signals.
- Using the information collected from the sensors, the rover should be able to successfully identify each type of lizard.

Environment

- Environment not fully known
- The demo environment will have a floor that is mostly smooth with defects less than 2mm high
- Some obstacles must be avoided entirely (e.g. trees and plants), while others with a height of up to 15mm can be crossed to reduce travel distance (e.g. roots and creepers).
- Some areas may be inaccessible to rovers weighing more than 750g.
- Possibility of a river which may have to be crossed

Target Product Cost

- Total budget below £50.
- Prototype budget below £15.
- Final product budget £25.
- £10 left over for maintenance and additional parts.

Manufacturing Facilities

- Limited to facilities in the lab

‘

- Due to budgetary constraints, it may be necessary to design and manufacture some parts of the rover as opposed to buying parts.

Size

- Max dimensions are defined as follows:
  - Length: 25 cm
  - Width: 20 cm
  - Height: 15 cm
- Wheels large enough to provide enough traction to manoeuvre over 2 mm high defects with ease
- Able to cross 15 mm obstacles
- Chassis ideally sits at least 4 cm above ground

Weight

- Maximum weight: 750 grams
- Ideal weight range: 400 to 550 grams
- Centre of mass in the middle for a balanced vehicle
- Heavy enough to maintain traction when crossing river (and on all other surfaces)

‘

Design Process

Concept Generation

In the early stages of the project, the group needed to generate different ideas and concepts to fulfill the requirements. Multiple methods were explored, including brainstorming, mind maps and concept matrices. Figures 1 and 2 show some of the ideas generated during a brainstorming session based on movement and mechanical design.

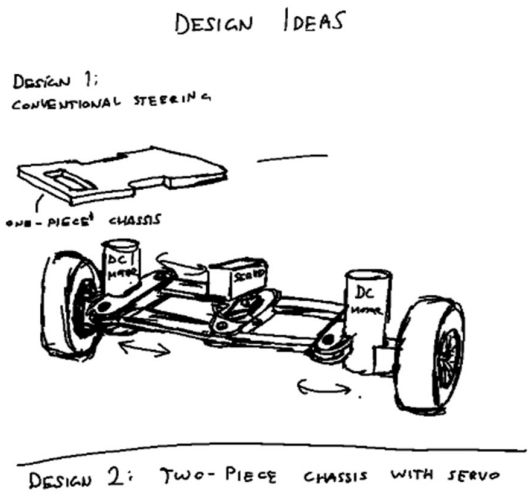
Figure 1

Figure 2

‘

Any idea was viable, each team member wrote down any idea that came to mind and posted them on the wall. This led to some very extreme and unrealistic ideas but allowed all members to engage with the concept generation phase and allowed for many creative ideas.

We eliminated any ideas which were completely infeasible and focused on expanding and visualizing ideas which we felt could possibly be implemented. Sketches were drawn of multiple ideas so that all members of the group understood the concepts that were being suggested and considered. Some of these included a servo motor, tank tracks as well as a 4-wheel system, with 2 wheels driven by the motors and are shown in Figures 3, 4 and 5.



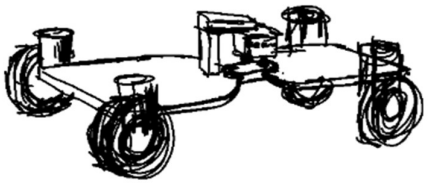


Figure 3 Servo motor design idea

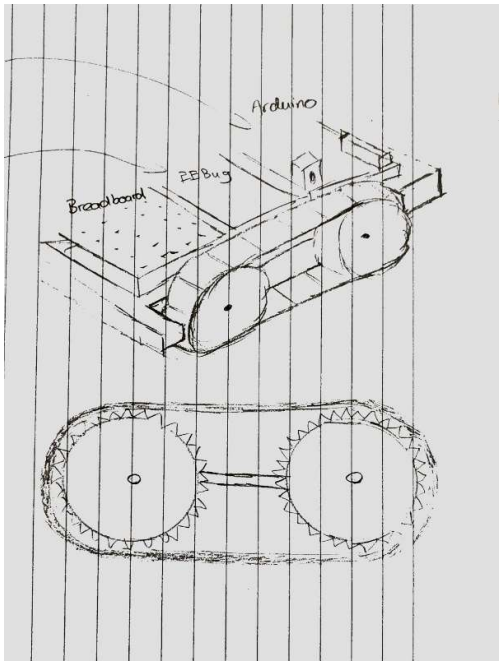


Figure 4 Tank Track design

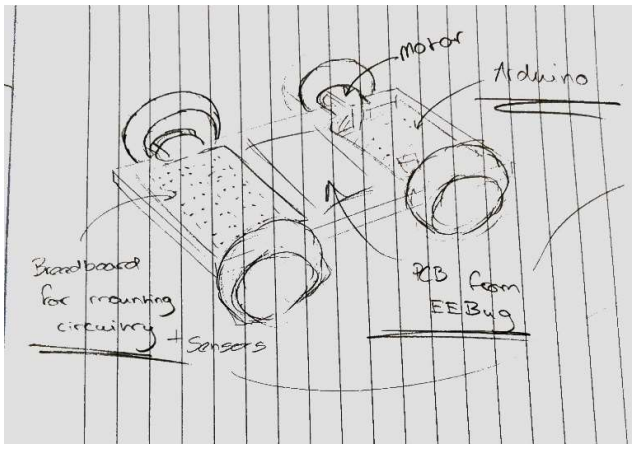


Figure 5 4-Wheel Drive

‘

Concept Selection

Directly following the process of concept generation, concept selection is essential so that all group members democratically agree on how to move forward and which ideas to implement. Each subsystem for sensors and mechanical design was put through the selection process. Although only the wheel design choice will be highlighted in this report.

A concept matrix method was used to logistically decide on the most appropriate wheel choice, considering necessary criteria determined by the PDS. This is shown in Figure 6.

Criteria	EE bug wheels with balance tip	Tank Tracks	4 motor powered wheels	4 wheels (2 motor powered)	Servo Motor
Speed	**Datum**	1	1	0	-1
Maneuverability		1	1	0	0
Cost		-1	0	1	-1
Accessable with current facilities		0	0	1	-1
Suitable diameter		1	1	1	0
Weight		0	-1	1	1
Robustness		1	1	1	-1
Power consuption		0	-1	0	1
Score	0	3	2	5	-2
Rank	4	2	3	1	5
Continue or combine	No	Combine	No	Combine	No

Figure 6 Concept Matrix

The process showed that the best option moving forward was to combine aspects of the tank tracks with the 4-wheel system. The 4 wheels can be designed to have improved traction with tank like treads, while maintaining the ease of assembly which 4 separate wheels provides.

‘

High Level Design

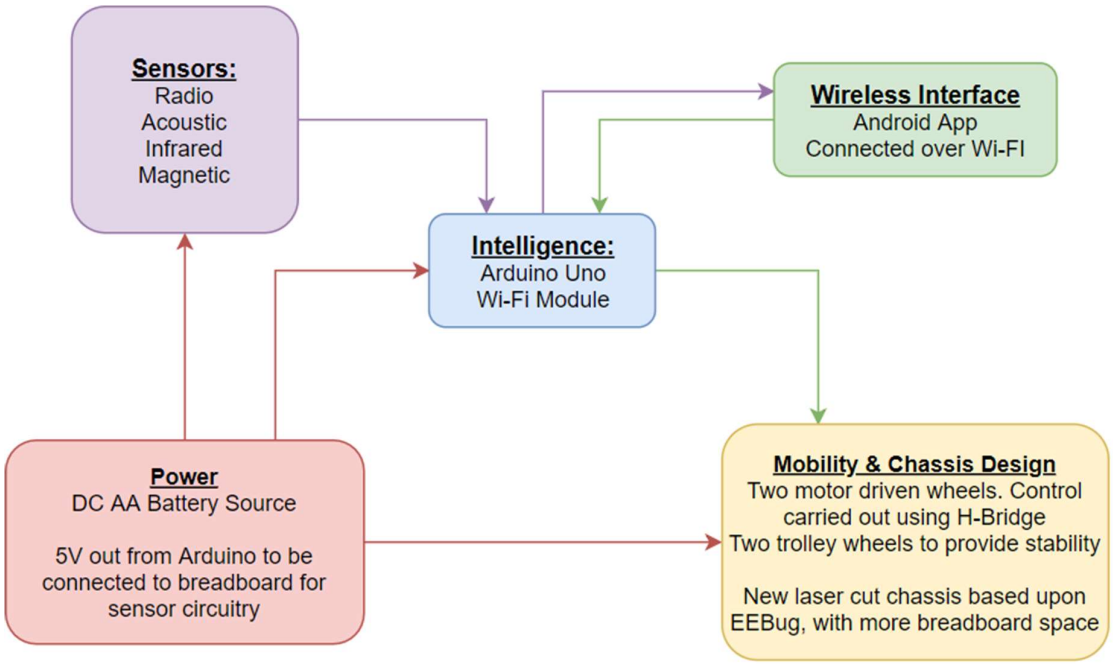


Figure 7 High-Level Design Flow Diagram

The diagram above (Fig. 7) gives a simple high-level design of the rover. It has been broken down into it's 5 major subsystems. The objective for the project was to build each subsystem separately before combining them all together to create a working rover.

The explanation provided by the picture above can be further elaborated on:

The sensor block contains each of the sensors on board the rover. These will then send any signals to the intelligence block. Here the Arduino will process the signals as necessary, before sending the results through to the app via Wi-Fi. In the app, a guess will be made to detect what the lizard is, however the processed signals will also be shown (e.g. the radio sensor might show "#NUC").
The interface is also how the rover is controlled, as the Arduino in the intelligence block has a connection to the h bridge controlling the motors. As the such the person using the app will be able to control the speed of the rover and the direction in which it is travelling.

‘

Sensor Design

Radio Sensor

Four out of the six lizards that are being detected generate radio signals, and each signal also contains unique information specifically identifying the lizard. This information was specified in the Project Brief, with each identifying message being on-off modulated with UART coding at 300bps.

Gaborus	61kHz radio modulated ‘#GAB’
Nucinkius	61kHz radio modulated ‘#NUC’
Durranis	89kHz radio modulated ‘#DUR’
Pereai	89kHz radio modulated ‘#PER’

Figure 8 Encoded signals from the brief

As such, it was decided that the radio sensor would be a priority in the EERover, and that it would be necessary to decode all signals received from it in order to identify the signals with maximum effectiveness.
In order to receive the EM Radio signals, an air cored inductor of diameter 9cm was constructed (Fig. 9), to be used as an antenna.

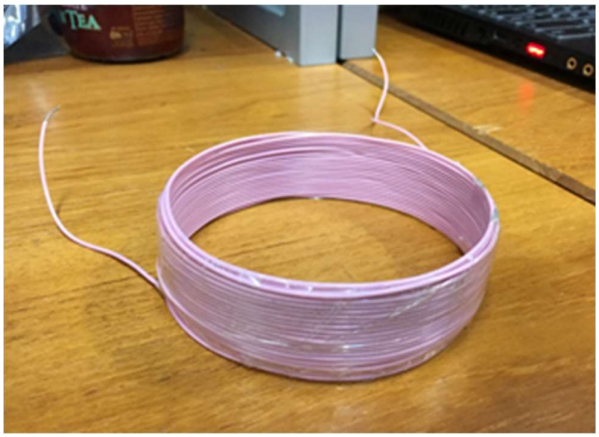


Figure 9 Air Cored Inductor

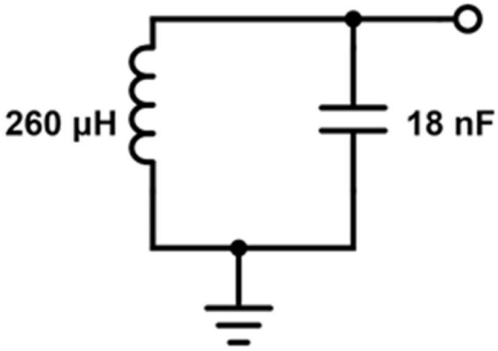


Figure 10 LC Circuit

The method for detecting a radio signal was to build an LC circuit that would resonate at a frequency equal to the carrier frequency of the radio signals. Circuits were constructed for each, and as the same inductor coil was used for both, the resonant frequency was varied by changing the capacitors used (12nF for 89kHz and 27nF for 61kHz).
Subsequently, another circuit was built. It used a capacitance of 18nF to produce a resonant frequency of 73.6kHz - a value in the middle of the two carrier frequencies. Through testing, it was determined that this circuit was able to pick up radio signals at both frequencies, and therefore could be used for both.
The proved to be advantageous, as a single circuit could be used for the amplification

‘

and processing stages, reducing the cost and the amount of space required for the circuitry.

Figure 11 demonstrates the ability of the circuit to detect the 89kHz signal.

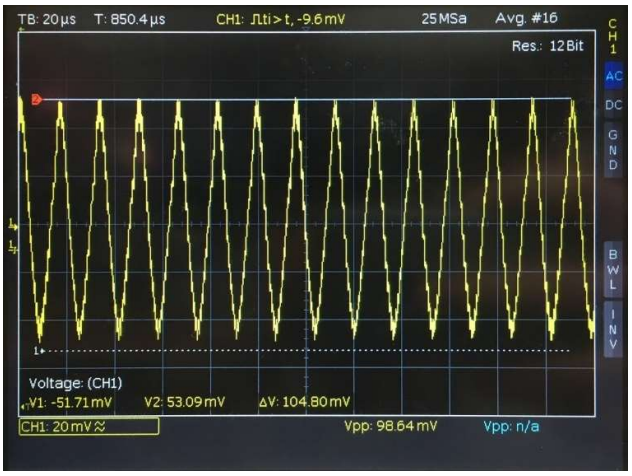


Figure 11 Screenshot of unamplified radio signal

As revealed by the screenshot, the signal is picked up by the LC circuit, however it only has a small amplitude – 98.64mV. Amplification is necessary in order to make the signal amplitude large enough to be processed by the Arduino. To do this, an amplification circuit was constructed using operational-amplifiers.

The Op-Amp chip used was the Microchip MCP6004 (Fig. 12). It was selected for a number of reasons: it has a sufficiently high GBP for its intended purposes (1MHz) allowing each amplifier to operate with a maximum gain of around 11.

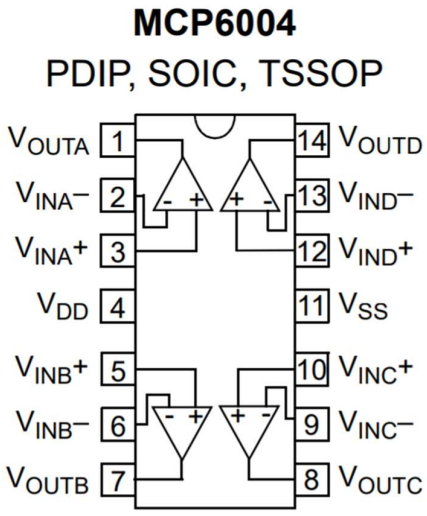


Figure 12 Microchip MCP6004 Op-Amp Source: Microchip.com

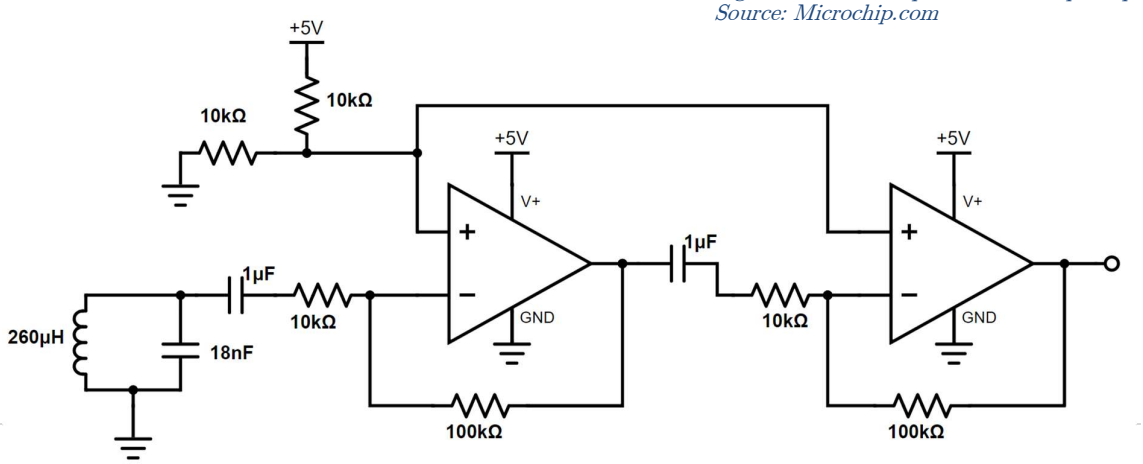


Figure 13 Amplification Stage of Radio Circuit

‘

The MCP6004 also supports rail-to-rail operation, which is necessary given the small operating range of 0-5V. The package also contains 4 Op-Amps, meaning that while two will be used for amplification, two more are left available for other applications, reducing the amount of space taken up elsewhere on the breadboard.

Furthermore, the MCP6004 is available at very low costs (£0.38), making it a highly economic solution.

The LC circuit and amplification stages are shown in Fig. 13. The amplification is carried out by two inverting amplifiers, each biased at 2.5V. The biasing is required as the AC signal operates around 0V, but the op-amp can only output in the range of 0-5V. By adding a DC bias, no signal is clipped by the amplifiers. A single potential divider configuration is used to achieve this DC biasing, and each stage is coupled with a capacitor to remove unwanted DC bias from the input signal.

The gain of each stage is determined by the equation: $Gain = \frac{R_{feedback}}{R_{input}} = \frac{100k\Omega}{10k\Omega} = 10$

The overall gain should therefore be approximately 100. The actual gain is shown in the screenshot of Fig. 14.

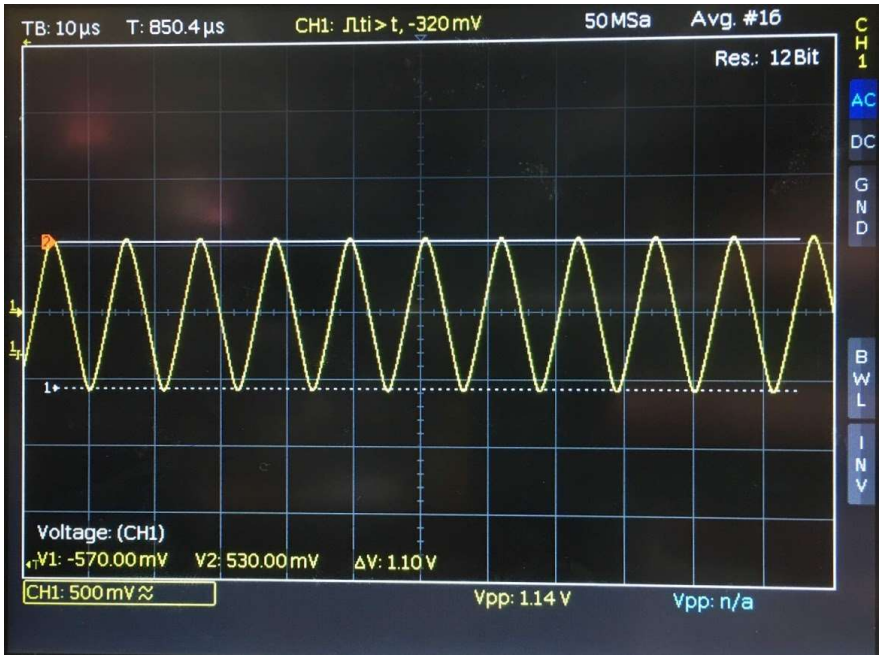


Figure 14 Screenshot of amplified radio signal

The measured gain is therefore 1.14/0.09864 = 11.5.

‘

The last thing to do after amplification is to convert the signal from analogue to digital. In order to do this, an envelope detector was constructed. The envelope detector first rectifies the signal, using a diode, before passing the signal through a low pass filter to remove the carrier frequency while allowing the much lower frequency ‘envelope’ to remain.

The diode exists to perform the rectification of the signal. A diode is used due to its asymmetric conductance. Negative signals are unable to progress past the diode, however positive signals (if larger than the threshold voltage) will make it through. This is demonstrated in the Figure 16.

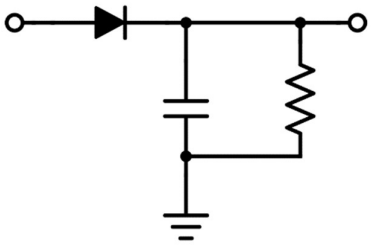


Figure 15 An Envelope Detector

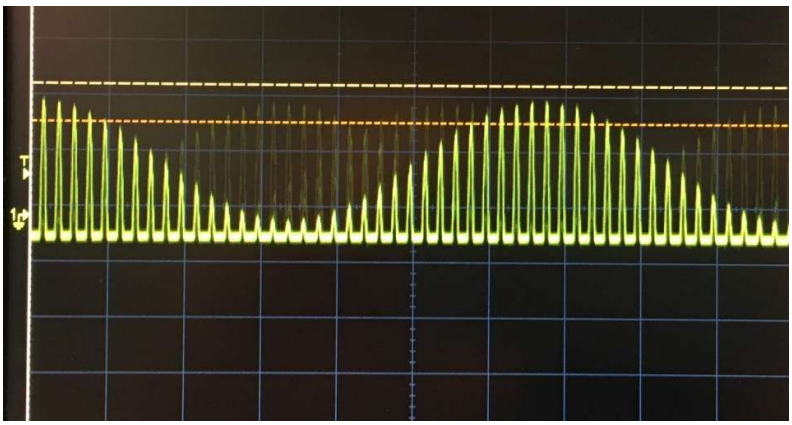


Figure 16 Rectified AC Signal

The operation of the envelope detector in removing the high frequency carrier frequency is outlined in the Figure 17 below:

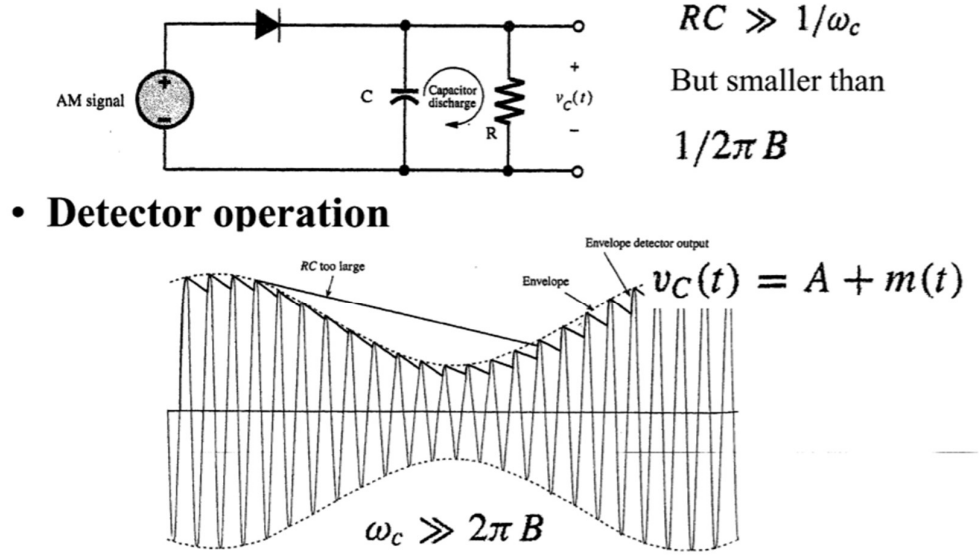


Figure 17 Operation of an envelope detector. Source: Prof. Ken Leung - Signals & Communications Lecture Slides

‘

As is shown in the diagram, it is important to choose a capacitor-resistor combination that produces an RC value in the range: $\frac{1}{\omega_C} \ll RC < \frac{1}{\omega_m}$

To achieve this, values of R = 10kΩ and C = 47nF were selected, producing an RC value of 0.00047s.

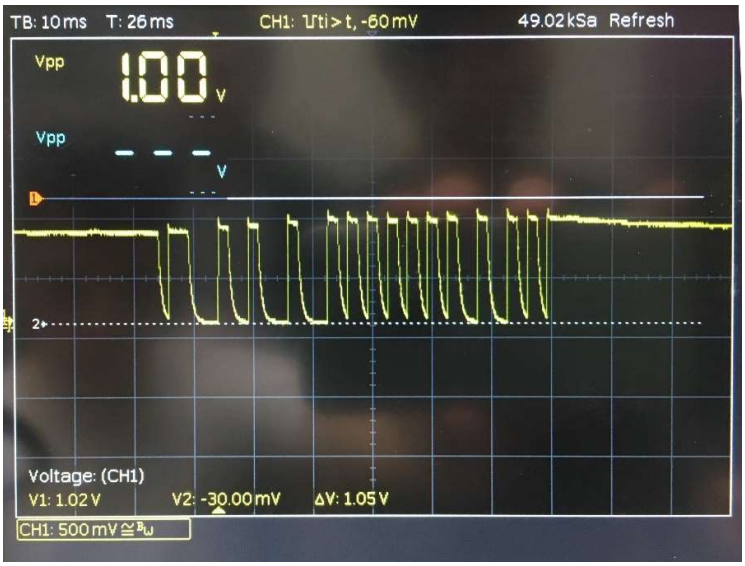


Figure 18 Demodulated Radio Signal

As seen from the scope in Fig. 18, the modulated signal has now been demodulated into a clear digital signal.

The circuit proved to work well even when the vertical distance between the antenna and the lizard was increased.

Further work may need to be carried out in order to make the signal even cleaner, perhaps the implementation of a Schmitt trigger as is done in the acoustic sensor.

Pending testing of the sensor with the Arduino, it will become clear whether the Arduino is able to decode the UART message contained.

‘

Infrared Sensor

The sensor chosen to detect infrared signals was the SFH300-3/4 phototransistor which was already provided and available from the EEBug experiment. The base terminal of the phototransistor is exposed and provides a base current when light energy is incident on the base. Connecting the phototransistor in series with a resistor and using the 5V and ground connections from the PCB, shown in Figure 19, allows for the output to vary as a DC value dependent on the intensity and wavelength of incident light. The infrared pulses would allow the value at the output to vary as an AC value. The chosen sensor has a range of detection between 400 nm to 1100 nm, shown in Figure 20 from the data sheet of the SFH300-3/4. The range of infrared wavelengths is greater than 700 nm. This means that natural light in the rover environment will affect the output of the circuit and needs to be filtered out to correctly determine the presence and frequency of infrared pulses.

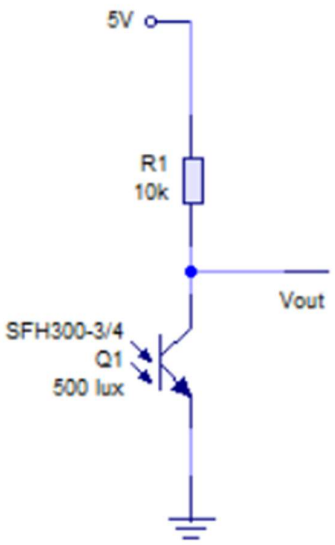


Figure 19 Phototransistor Circuit

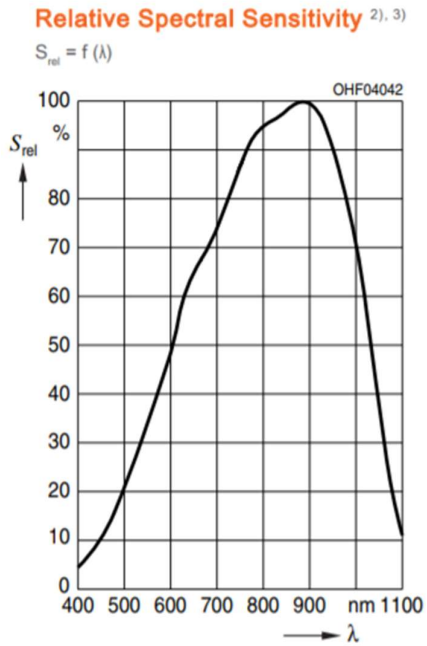


Figure 20 Relative spectral Sensitivity of SFH300-3/4

It was decided that using one circuit for both frequencies of infrared pulses was ideal as this would reduce the breadboard space and components needed. The simplest idea was to use a bandpass filter with a peak gain range containing the two frequencies of infrared signals. Theoretical testing of this determined it not be feasible as the circuit would require a minimum of 4 stages, increasing the cost and using much space on the breadboard. The solution which was decided on was to use a twin T notch filter. The circuit shown below in figure 21 is designed to have a resonant frequency of 468.9Hz. This is close to midway between the two frequencies

of infrared, 353Hz and 571Hz giving a mean of 462Hz. This will give significant amplification of both infrared frequencies, using standard resistor and capacitor values.

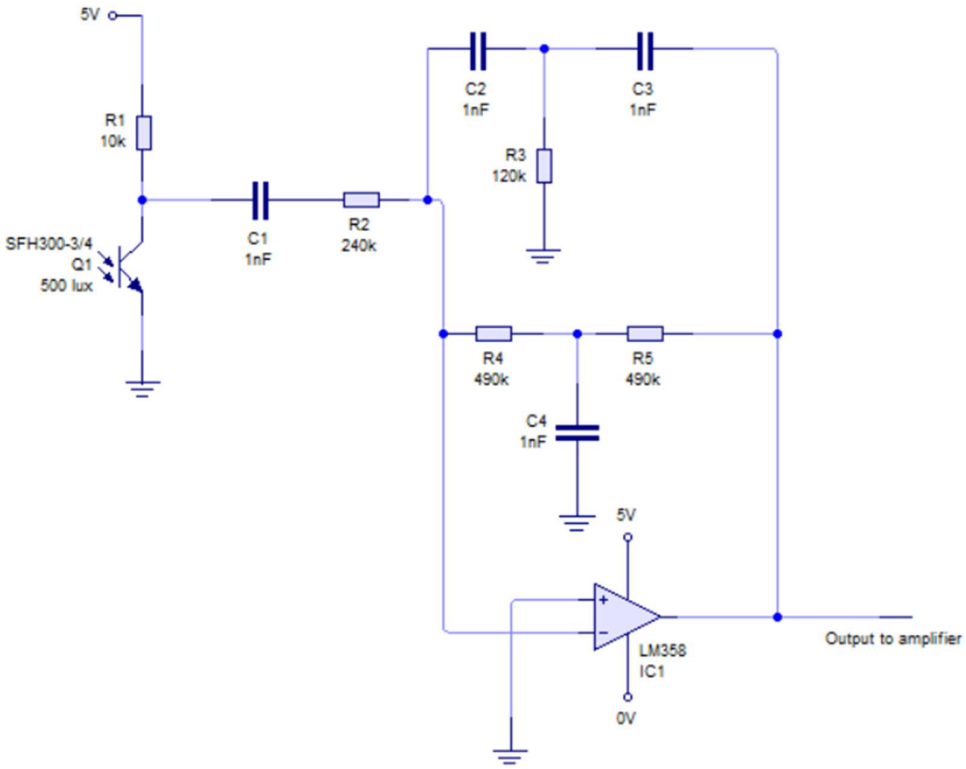


Figure 21 Twin T Notch Filter connected to phototransistor circuit

The magnitude of the output from this circuit is rather small if the phototransistor circuit is not directly next to the lizard and needs to be a passed through a non-inverting amplifier to be easily detected on the analogue input of the Arduino. This is done using the circuit in Figure 22, designed to have a voltage gain of x21. The complete sensor circuit is shown in Figure 23 and the outputs from each infrared frequency, at 2.5cm away from the lizard are shown in Figures 24 and 25.

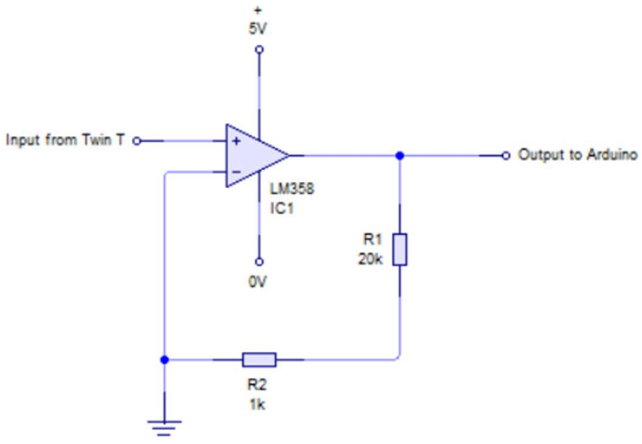


Figure 222 Non-inverting amplifier with x21 gain

‘

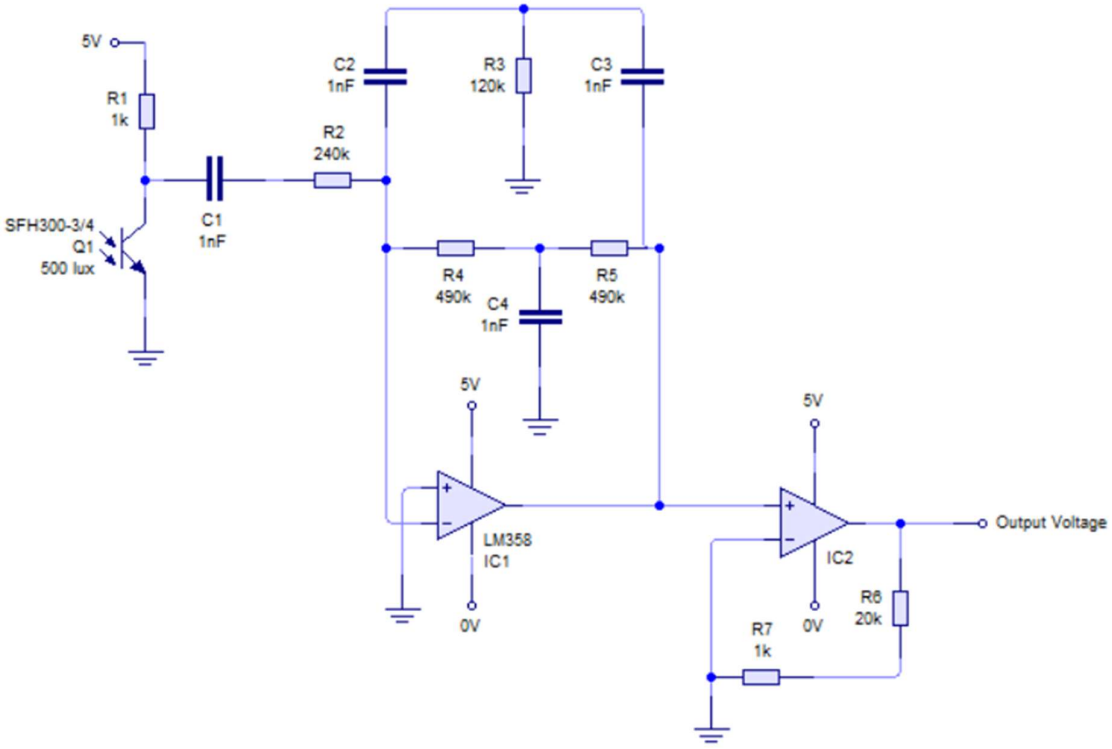


Figure 23 Complete Infrared Sensor Circuit

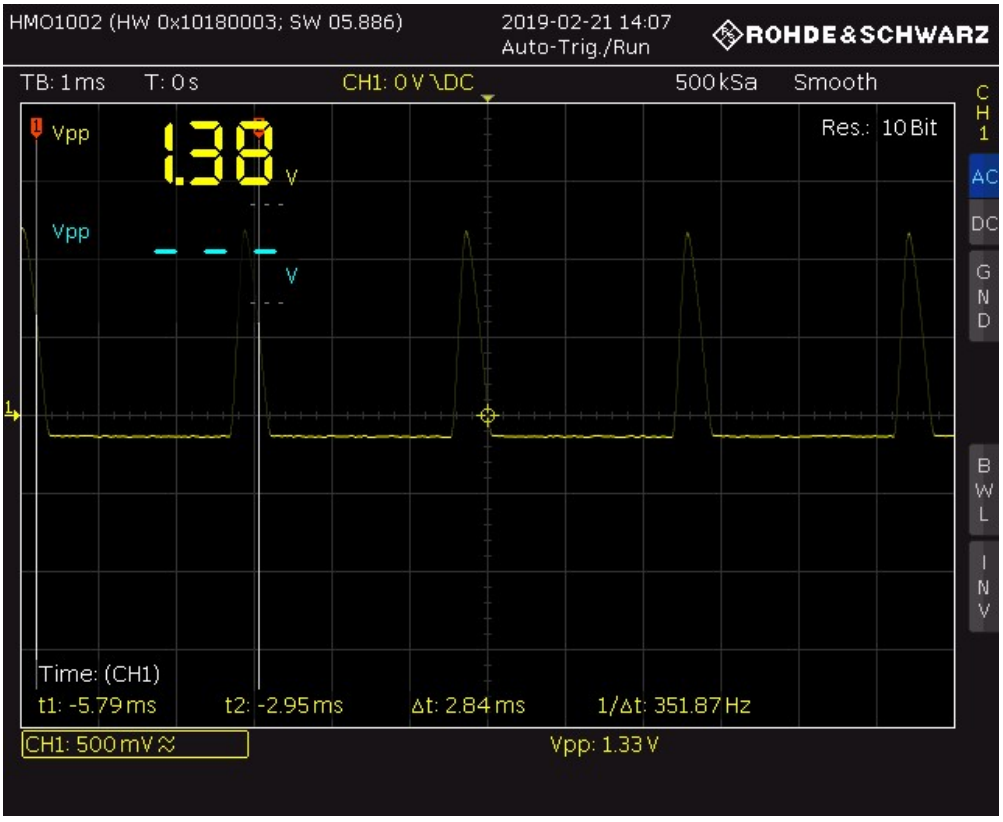


Figure 24 Output from 353Hz Signal

‘

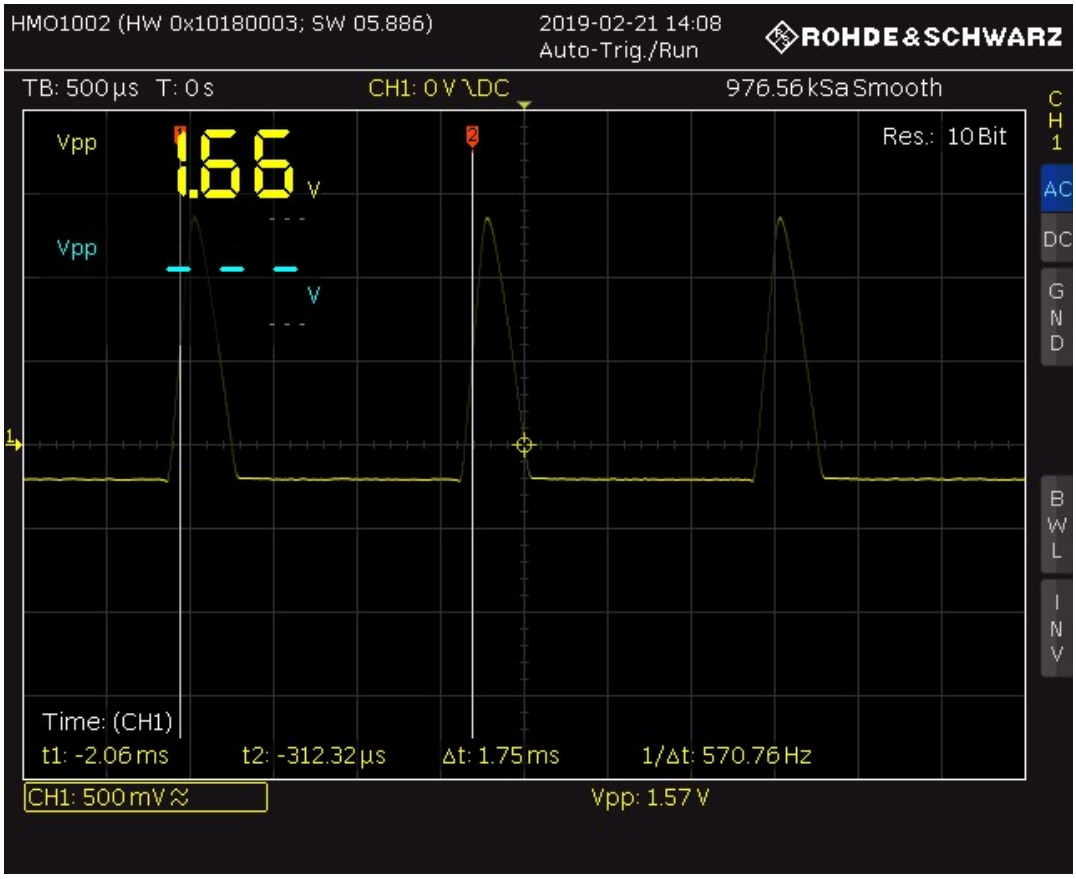


Figure 25 Output from 571Hz signal

Both frequencies of signal give very accurate outputs which can be easily distinguished and determined based on the frequency of positive voltage peaks.

‘

Magnetic Sensor

The magnetic field generated by two of the lizards is a static magnetic field, polarised either up or down. The group decided that a sensor which could detect the magnetic field while the rover was stationary was the most feasible solution. After researching different possibilities, the group opted to use a hall effect sensor to determine the presence of magnetic fields. The SS496A1 Hall Effect Sensor was used, as it can operate linearly from a 5V supply, and its output can be used to easily differentiate between up and down fields.

The SS496A1 has 3 connection pins, supply voltage, ground and output voltage. The

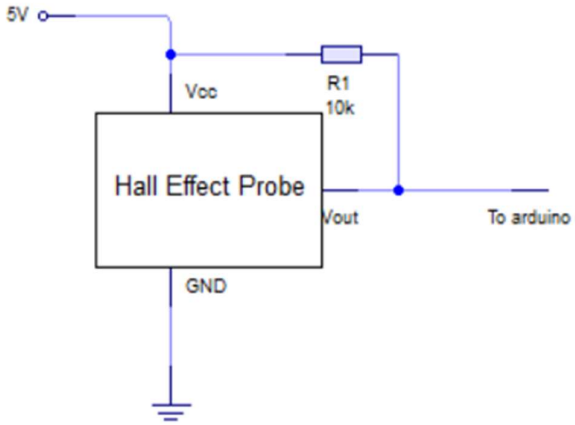


Figure 26 Magnetic sensor circuit

output voltage is determined by the strength and polarity of magnetic field detected by the sensor. The sensor circuit designed is shown in Figure 26. The voltage output is a DC voltage.

Connecting a 10k**Ω** resistor between the output and supply terminals of the sensor, allows the output voltage to be around 2.5V when no field is present. An ‘up’ magnetic field would increase the output towards 4.5V and a ‘down’ magnetic field

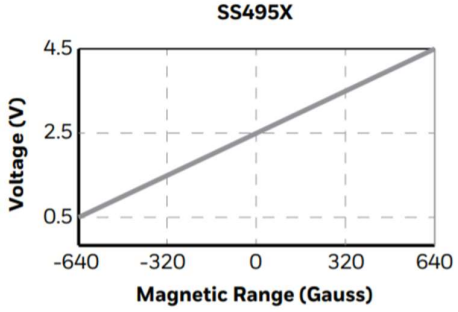


Figure 27 Output voltage when using a 5V supply, taken from data sheet of SS496A1

‘

would decrease the output towards 0.5V. This information is collected from the data sheet of the SS496A1 and is shown in Figure 27.

The circuit was tested to identify the output values for each detected field and determine the threshold voltages for the analogue inputs of the arduino, in order to correctly identify each field. When no field was present the output was at 2.49V. Then one end of the magnet was placed near to the sensor, this increased the voltage increased to 4.48V. The magnet was then turned around and the voltage dropped to 0.946V. These are shown in Figures 28, 29 and 30 respectively.

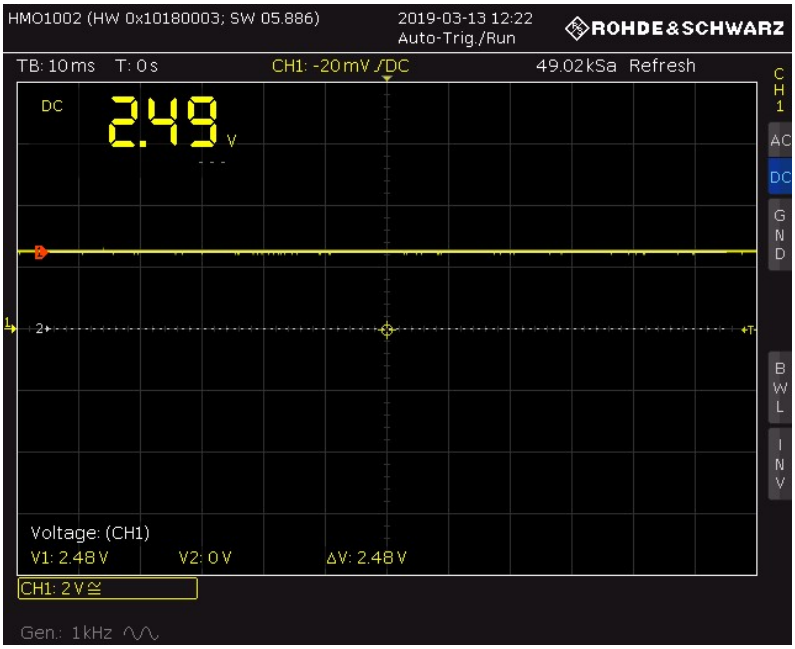


Figure 28 Output when no field is present

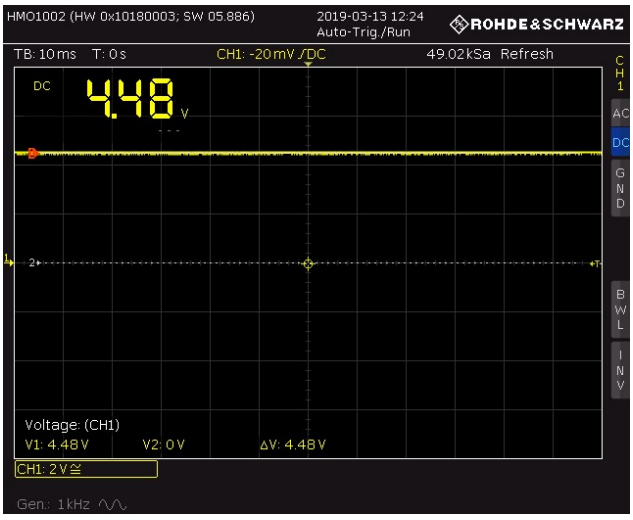


Figure 29 Output with up field present

‘

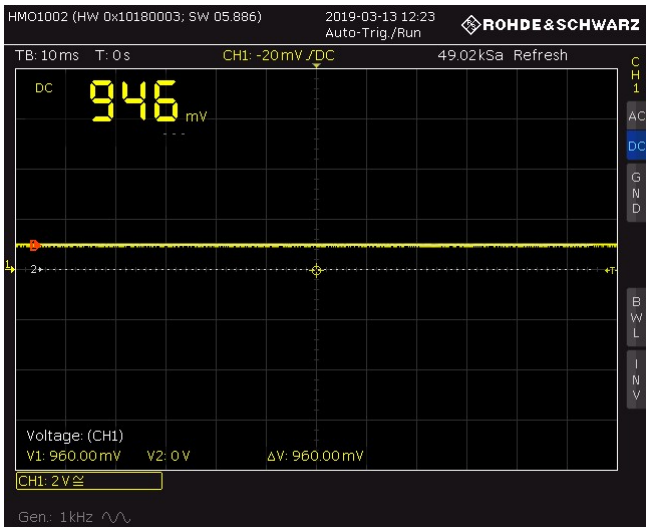


Figure 30 Output with down field present

The analogue input of the Arduino is used to determine the magnetic field present. However, the Arduino input has a maximum voltage of 3.3V. This means that the output from the magnetic sensor circuit has to be scaled down to be correctly read by the analogue input of the Arduino. Using a potential divider with a 3k**Ω** and 1.5k**Ω** resistors allows the range of the output to be 0.3V - 3V. With output threshold values of:

- 0.3V - 1.3V Down field
- 1.3V - 2.0V No field
- 2.0V - 3.0V Up field

This allows the analogue inputs of the Arduino to be used to easily differentiate between the signals and determine whether a magnetic field is present and if so, the polarity of the field. The complete circuit is shown in Figure 31.

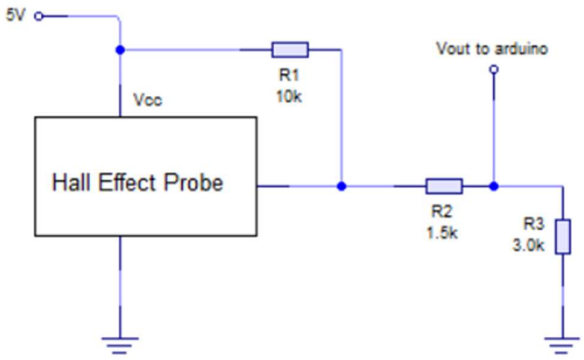


Figure 31 Complete magnetic sensor circuit.

‘

Acoustic Sensor

From the project brief, it can be ascertained that the species of lizard can be determined without measuring all the signals. For instance, based on whether a UART encoded radio signal or infrared pulses is detected, one can determine whether the lizard belongs to “Gaborus, Nucinkius, Durranis, Pereai” or “Cheungus, Yeatmana”. Furthermore, just by knowing the frequency of the infrared or radio waves, the direction of the magnetic field and the **presence** of acoustic waves, which are all at 40 kHz, the exact species can be determined.

Hence, the main objective of the acoustic sensor and its circuit is to detect the presence of the acoustic signal that is set to be 40 kHz instead of measuring the accurate value taken up by the sensor. This simplifies the overall design to accommodate other features for the rover.

The detection of acoustic signal of 40.0kHz emitted by two lizards, Gaborus and Yeatmana, is useful in differentiating and identifying those two lizards from the others.

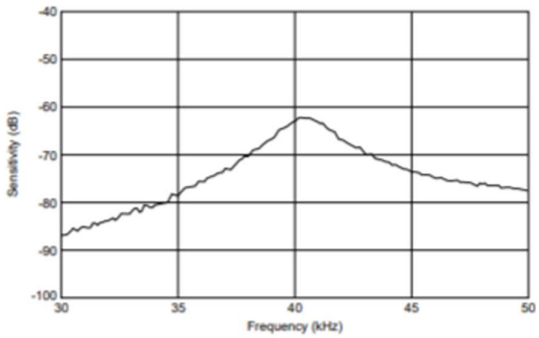


Figure 32 Sensitivity (dB) vs Frequency (kHz)

The main component used in the acoustic sensor is the RS Pro Ultrasonic Barrel Sensor. This ultrasonic sensor was chosen for several reasons. First of all, as discussed in the budget plan, the price is reasonable compared to other sensors. Furthermore, this sensor is most sensitive at 40kHz as shown in Figure 32, which is the target frequency for the signal emitted by the lizards. Moreover, the high sensitivity and penetrating power of the ultrasonic sensor means that the sensor can readily detect external acoustic signals with high accuracy, considering how the signal would not be strong when embedded in the lizard.

‘

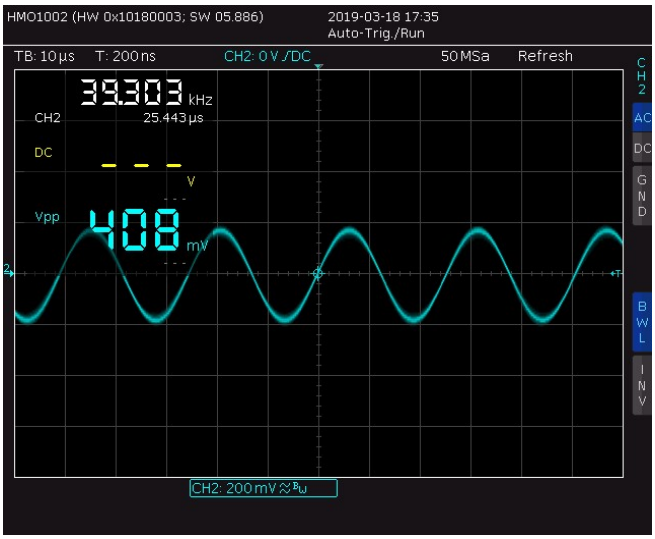


Figure 33

Initially, the RS Pro Ultrasonic Barrel Sensor was tested with a simple circuit connecting the sensor. As shown in figure 33, the acoustic signal was obtained from the oscilloscope probe when the acoustic sensor generator was placed near the sensor. However, the signal was too small to detect when the distance between the sensor and the lizard was increased. Therefore, an amplifying circuit has been constructed to amplify the signal.

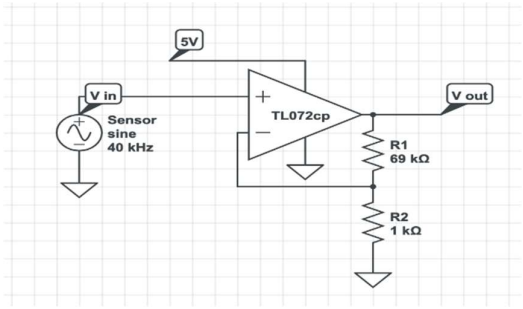


Figure 34 Amplifier implemented on the sensor

A non-inverting amplifier was constructed using the Op Amp-TL072CP. We have set the gain of the non-inverting amplifier as Gain = 1 + 69000Ω/1000Ω = 70 by using a 69k resistor and a 1k resistor. For the resistors employed in the circuit design, no extremely small or big values are chosen to lower the sensitivity to interference as much as possible. From the datasheet of the RS Pro ultrasonic sensor, it stated that it will detect acoustic signals up to 50 kHz, raising concerns in potential need of a filter to remove unwanted signals.

‘

While the signal to be processed is set at 40 kHz, a significant gain is required for the ease of processing by the Arduino program, which in this case is set at 70 as discussed. Hence, the required gain-bandwidth product is 2.8MHz. TL072CP is suitable for the non-inverting amplifier since the gain bandwidth required is 2.8MHz while the maximum gain bandwidth of TL072CP is 3 MHz. The amplifier is thus implemented as depicted in figure 34 above, while its other working conditions are suitable for the test environment, e.g. 200V/mV Gain, 0flC to +70flC working temperature, etc.

While the waves produced from the test results indicates that there is successful detection of the signals, it is best if the analogue waves can be made into easily readable and processible waves by the Arduino. Hence, a Schmitt Trigger is implemented as shown below in Figure 35 so that the positive feedback of the circuit design will adjust the output and make the wave oscillate between fixed values. While this leads to the produced waves being inverted, it is not a major problem for the Arduino to process. The test results are to be further investigated and compiled in the coming term.

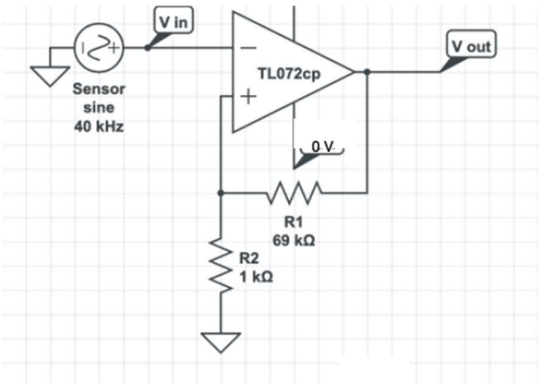


Figure 35 Schmitt Trigger

Considering the factor of the presence of ambient noise in the test environment, which the targeted frequency is presumably higher at 40 kHz, it was determined that it may be worthwhile to implement filters in the circuit, for instance, a 2nd order high pass filter that attenuates lower frequencies like the selective sallen-key filter. Upon testing, it was however found that it is not necessary to implement this set-up, as this factor is insignificant, while taking up extra space on the breadboard.

Hence, the use of the filter is deemed unnecessary as the main objective is to detect the presence of the wave to support the feedback from other sensors, which a more straightforward design as suggested already fulfils the objective and should suffice.

‘

Control and Interface Design

The rover is controlled via an Android application, which communicates with the Adafruit Metro M0 Express board via a WINC1500 Wi-Fi shield, over a WLAN (Wireless Local Area Network). The application was written with Java on Android Studio and tested on a Google Pixel 3 running Android 9.0, via ADB (Android Debug Bridge).

The application, as shown in figure 36 below, welcomes the user with a screen that allows them to choose their controls. The user can choose between “Button Controls” and “Slider Controls”.

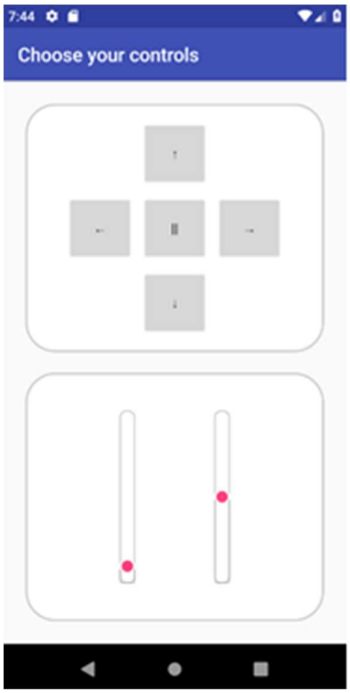


Figure 36 Control Selection

Figure 37.1 IP Configuration Screen

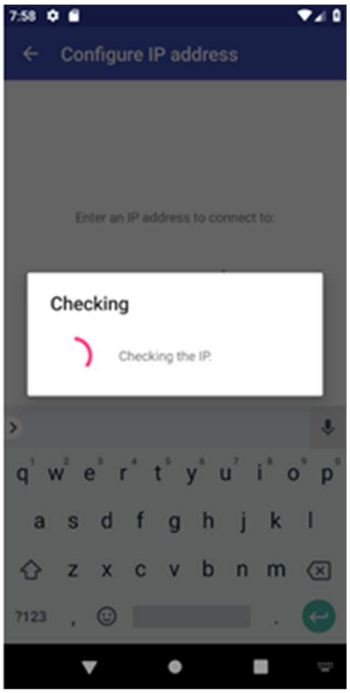


Figure 37.2 Verifying IP

When the user selects a control method, they are sent to an IP configuration screen, shown in figure 37.1 above. In this screen they can enter the local IP address assigned to their WINC1500 Wi-Fi shield’s MAC address and establish a connection between the app and Wi-Fi shield. When the “OK” button is clicked, the app scans the local network to find the IP address, and verifies it. If the address is valid and reachable, it then sends a connection request to the Wi-Fi shield. Afterwards, an InputStreamReader is created, which reads bytes from the source IP and decodes them into characters using a specified charset. It then appends the read input to a String, using a StringBuilder. This String is used to verify that the connection is up.

‘

Once the connection is up, the IP address is saved, and the user is redirected to the ButtonControls page (Figure 38). On this page, there are six buttons and a slider – four of the buttons are used to control the direction of the rover (Forwards, backwards, left and right). The rover keeps turning for as long as the left or right button is pressed, and reverts to going straight (backwards or forwards, depending on which direction it was going before the right or left buttons were pressed). The middle button can be thought of as a handbrake: it immediately reduces the speed to 0 when pressed. The speed slider controls the speed and has 10 stages, from 0 to 9. The “Reload IP” button reloads the IP address to be used by the app and displays it on the upper left corner of the screen.
When any direction button is pressed, or the speed bar changes its state, a GET request is sent to the Wi-Fi shield with the appropriate line ending. /F for forwards, /L for left, and so on. When the speed slider location changes, a line ending /H is sent, followed by the value of speed. Only ten stages were included for the speed, since this is precise enough and prevents a lag in changing speed (caused by too many GET requests sent if there are too many stages).
Screenshots of the code handling button inputs and the corresponding actions on the Metro board can be found in Appendix 4.
This screen is locked to portrait mode only, because it was found to be much easier use the buttons in this mode.

The other method of control that can be selected by the user in the “Choose your controls” screen is “Slider Controls”, that contains, in addition to the “Reload IP”

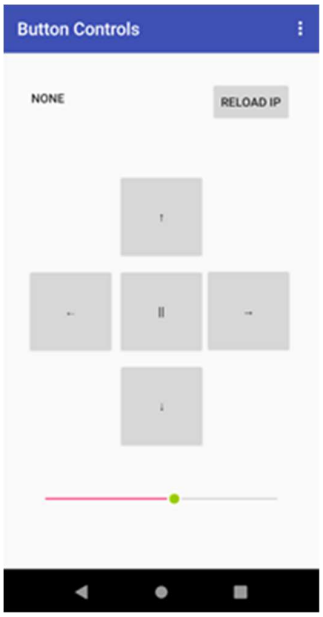


Figure 38 Button Control Screen

‘

button and IP text field, two sliders only; one to control the speed of the left motor and the other for the right, as can be seen in figure 39 below.

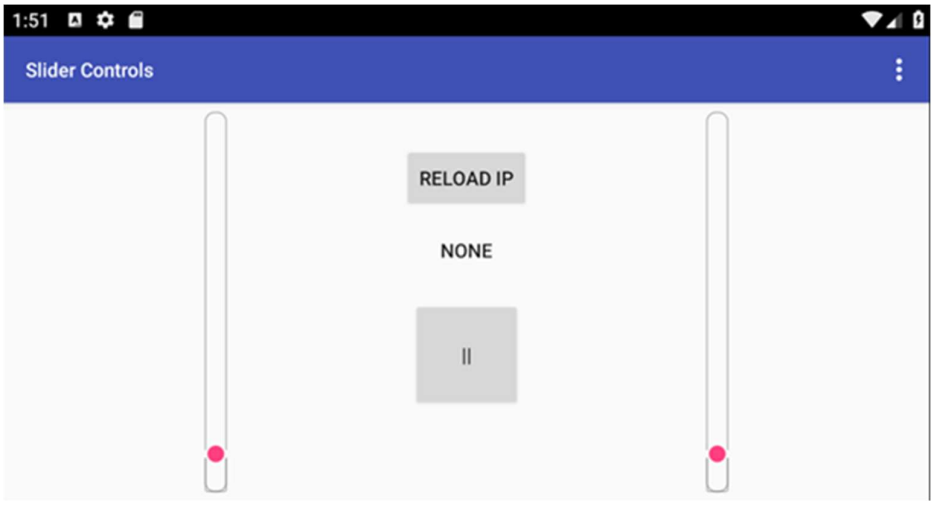


Figure 39 Slider Control Screen

This view is only available in landscape mode, since it was found to be easier to access the sliders comfortably and control the rover in this mode.
Similar to the button control screen, the initial selection of the control method sends the user to a “Configure IP” page, the “back” button on whose toolbar sends them to the screen depicted in figure 4.
Much like the single speed slider on the button control screen, the left and right motor sliders on this screen have 10 stages, the lowest of which being 0 and the highest being 9. The stop button reduces the speeds of both motors to 0.
Identical to the button control screen, pressing the triple dot menu on the toolbar takes the user to the IP configuration page, where they are able to change the IP address.

Issues with the design and future configurations

As of now, the app does not receive any data from the sensors on the Metro board. During the spring break, all Metro intelligence will be completed and responses from the Metro board will be sent to the app. The lack of intelligence also means that there is no means, at the moment, to control the sensors from the app. We have not added any buttons to send power to any of the sensors, because although all sensor circuits are complete and in working order, they are not connected to the Metro board and we have no way of making them communicate to the app.

‘

Intelligence

This section contains the algorithmic thinking and code behind the analysis of each signal and therefore ultimately the determination of the lizard under investigation.

Species	Property 1	Property 2
Gaborus	61kHz Radio Signal (#GAB)	40kHz Acoustic Signal
Nucinkius	61kHz Radio Signal (#NUC)	None
Durranis	89kHz Radio Signal (#DUR)	Magnetic Field Up
Pereai	89kHz Radio Signal (#PER)	Magnetic Field Down
Cheungus	Infrared Pulses (353Hz)	None
Yeatmana	Infrared Pulses (571Hz)	40kHz Acoustic Signal

Figure 40 Lizard Properties

Due to the properties of the lizards as given in the project brief and as presented in table form in Figure 40, there were automatically some logical conclusions that could be deduced in order to simplify the analysis and determination processes. These can be summarised as follows:

- The presence of an acoustic signal bypasses the need to differentiate between the two 61kHz radio signals. More practically, if a radio signal *and* an acoustic signal are present, then the only possible lizard with such characteristics is Gaborus.
- The same argument holds for the two different infrared signals.
- In a similar manner, the determination of the polarity of the magnetic field bypasses the need to differentiate between the two different 89kHz radio signals.

With these facts in mind, it was possible to start writing initial Arduino sketches for each of the four sensors with a higher aim of combining all of them together to create the overall sensor program.

Individual Sketches:

Magnetic:

As indicated in the sensors section, the magnetic sensor output is as follows:

0.3V – 1.3V down field

1.3V – 2.0V no field

2.0V – 3.0V up field

Therefore, an analogue input was going to be used, as there was more than one threshold value present between the different output states. Our board (Adafruit

‘

Metro M0 Express) uses 3.3V logic, and its analogue inputs operate with 12-bit precision. This means that the input values could only be in the range 0-3.3V, and that this range would be mapped by the board ADC to values 0-4095. This gives the following correspondences.
0.3V: (0.3/3.3) * 4096 = 372.36
1.3V: (1.3/3.3) * 4096 = 1613.57
2.0V: (2.0/3.3) * 4096 = 2482.42
3.0V: (3.0/3.3) * 4096 = 3723.63

Using these proportionalities, rough upper and lower limits for each range were determined, making it possible to write a sketch which would easily determine the sensor output state based on the value of the analogue input.

Magnetic

```
int analogPin = A1;
int measurement;
int polarity;

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

void loop() {

  measurement = analogRead(analogPin);                  //read the value of the input

  if((measurement>370) && (measurement<1610)) {            //implement the different
    polarity = -1;                                         //threshold values to
  }                                                        //determine the polarity
  else if((measurement>1615) && (measurement<2480)) {      //of the field
    polarity = 0;
  }
  else if((measurement>2485) && (measurement<3720)) {
    polarity = 1;
  }
```

Figure 41

Infrared:

As stated above, and assuming the conclusions drawn are accurate, the presence or not of an infrared signal should suffice for the determination of the investigated lizard. This resulted in an Arduino sketch which would use a single digital input and depending on its state (LOW/HIGH), the presence of an IR signal would be determined.

‘

IRanalog

```
int read1, read2;
unsigned long time1, time2;
int analogPin = A0;
double period;
double frequency = 0;
                                        //     0.5V
                                        //     ----  *  4096  ≈  620
                                        //     3.3V
void setup() {
  Serial.begin(9600);
}

void loop() {
  time2 = time1;                        //make time2 = previous time measurement

  read2 = read1;                        //take two consecutive measurements
  read1 = analogRead(analogPin);        //of the analog pin

  if((read2<620) && (read1>=620)) {     //every time the signal crosses 0.5V with a positive slope
    time1 = micros();                   //use micros() to keep a time measurement
  }

  period = time2 - time1;               //compute the period as the difference of the two times
  frequency = 1/period;                 //compute the frequency as the inverse of the period
}
```

Figure 42

However, it was decided that it was better to additionally develop an algorithm which could distinguish between the two signals, which meant being able to determine the signal's frequency. This was achieved by feeding the sensor output to an analogue input of the board. The comments in the sketch in the screenshot below

Radio

```
int input_pin_radio = 2;
int radio_presence = 0;

void setup() {
  pinMode(2, INPUT);                                //simply declare digital
}                                                   //pin 2 as an input

void loop() {
  radio_presence = digitalRead(input_pin_radio);    //read its value and store
}                                                   //it in a variable
```

Figure 43

indicate how the algorithm works to calculate the frequency. It is again important to remember that the metro board works with 3.3V logic and has 12-bit precision.

Radio:

Again, assuming our initial conclusions, we need only determine the presence of a radio signal, which, once more, requires a single digital input.

‘

Acoustic

```
int input_pin_acoustic = 3;
int acoustic_presence = 0;

void setup() {
  pinMode(3, INPUT);                                    //declare pin 3 as input
}

void loop() {
  acoustic_presence = digitalRead(input_pin_acoustic);  //read its digital value and
}                                                       //store it in a variable
```

Figure 44

Note: In 3.3V logic boards, voltage above 2.0V is considered HIGH, and any voltage below 1.0V is considered LOW.

Acoustic:

Since there are no different acoustic signals emitted, then again, a simple digital input sketch would suffice to determine the presence.

Overall Sensor Program:

Having written these initial sketches for every sensor individually, the next step was to combine them together in a single sketch. It should be noted that at this point it was suggested to use 3 of the board's GPIO pins as output pins to indicate the determined lizard. As there are 6 lizard species, then with 3 pins all permutations could be implemented. These can be summarised in Figure XX.

It should be mentioned that for the first version of the overall sensor program, a simple sketch was used for infrared that would only detect presence.

Condition 1	Condition 2	Lizard Species	Pin 8	Pin 9	Pin 10
Radio Presence	Acoustic Presence	Gaborus	LOW	LOW	LOW
Radio Presence	No Acoustic Presence	Nucinkius	LOW	LOW	HIGH
Radio Presence	Magnetic Up	Durranis	LOW	HIGH	LOW
Radio Presence	Magnetic Down	Pereai	LOW	HIGH	HIGH
Infrared Presence	No Acoustic Presence	Cheungus	HIGH	LOW	LOW
Infrared Presence	Acoustic Presence	Yeatmana	HIGH	LOW	HIGH

Figure 45 Summary of overall sensor program outputs

‘

Screenshots of the initial overall sensor program can be found below in Figures 46 and 47.

Combined

```
int input_pin_radio = 2;
int radio_presence = 0;

int input_pin_acoustic = 3;
int acoustic_presence = 0;

int input_pin_infrared = 4;
int infrared_presence = 0;

int analogPinMagnetic = A1;
int measurement;
int polarity;

void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  pinMode(8, OUTPUT);                                    //three output pins
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
}

void loop() {
  radio_presence = digitalRead(input_pin_radio);         //radio

  acoustic_presence = digitalRead(input_pin_acoustic);   //acoustic

  infrared_presence = digitalRead(input_pin_infrared);   //infrared

  measurement = analogRead(analogPinMagnetic);           //magnetic
  if((measurement>370) && (measurement<1610))
    {polarity = -1;}
  else if((measurement>1615) && (measurement<2480))
    {polarity = 0;}
  else if((measurement>2485) && (measurement<3720))
    {polarity = 1;}
```

Figure 46

Figure 46 above indicates the three input pins used as digital inputs, as well as the three output pins used to indicate the determined lizard. In the loop follows the combination of the individual sketches where each property is determined.

‘

```
if((radio_presence==1)&&(acoustic_presence==1)) {      //lizard
  digitalWrite(8, LOW);                                //determination
  digitalWrite(9, LOW);
  digitalWrite(10, LOW);
}

if((radio_presence==1)&&(acoustic_presence==0)) {
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  digitalWrite(10, HIGH);
}

if((radio_presence==1)&&(polarity==1)) {
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  digitalWrite(10, LOW);
}

if((radio_presence==1)&&(polarity==-1)) {
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  digitalWrite(10, HIGH);
}

if((infrared_presence==1)&&(acoustic_presence==0)) {
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  digitalWrite(10, LOW);
}

if((infrared_presence==1)&&(acoustic_presence==1)) {
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  digitalWrite(10, HIGH);
}

}
```

Figure 47

Figure 47 indicated the simple if statements which use the already-determined properties detected and regulate the output pin states accordingly.

Current testing focuses on making the above program more reliable. The first steps of this process up to now have been the following:

- Change the if statements into else if statements in order for the program to only determine a single lizard. During this process any else if statement containing polarity, the (at the time) only analogue input, was placed at the end of the else if sequence as analogue inputs are considered more unreliable considering factors such as constantly changing distance and thus input voltage.
- The infrared section was switched to use the calculated signal frequency instead of pure presence.
- Beginning of the development of an algorithm for the determination of the UART codes of the radio signals.

‘

The above steps, which are at the current time near complete implementation, have the aim of providing the team with the maximum number of analysis options available in order to choose the one with the best determination effectiveness.

Chassis Design

As progress has been made in the designing of circuits for each of the individual sensors, it has become obvious that space on the EERover is at a premium. If using the base chassis design from the EEBug, there would only be room for one breadboard. As such it became obvious that creating a new chassis design may be required.

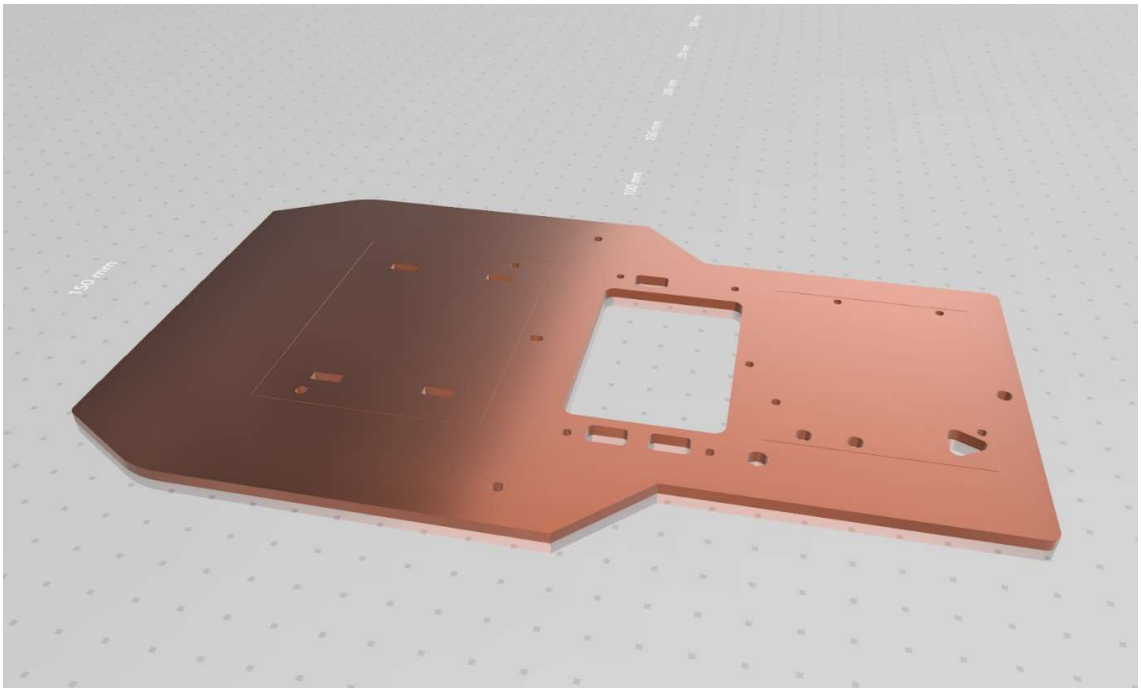


Figure 48 Fusion 360 Chassis Design

The above design, Figure 48, depicts a 3D model of a modified version of the EEBug chassis. It is wider to accommodate for an additional breadboard, enabling the EERover to more readily carry all the necessary sensors.

This design was created in Autodesk Fusion 360, and would be made through laser cutting acrylic.

It would have motors attached in the same way as the EEBug did, making the assembly easy.

Further ideas have been proposed for 3D printing larger wheels for the rover. Alternatively, rims could be laser cut from acrylic and rubber tires could be fitted to improve on the traction provided by the default wheel on the EEBug.

‘

Development Budget and Costings for the EERover

The initial plan on the use of the provided £50 budget was as follows:

- Prototyping Budget: £15
- Final Design: £20
- Contingency: £15

In the prototyping process, the budget is carefully monitored by the account manager, while requests on budget, i.e. for purchase of components, are agreed by the team. As prototyping began in February, by mostly referring to the initial budget plan, a sufficiently large budget is available for backup should any issues arise as follows

Component	Price (£)	Expenditure percentage of the prototype budget (%)
RS Pro Ultrasonic Sensor Barrel	7.34	48.9
TL072cp Opamp (2 ICs)	1.01	6.75
SS495 Hall Effect Sensor	2.24	14.9
LM358N Opamp	0.62	4.13
BZX85 Zener Diode	0.42	2.8
MCP6004 Opamp x 2	0.76	5.07
Total	12.39	82.6

For the prototype, the aim is to equip it with all the required functionality while lowering the cost. Hence, for the construction of the acoustic sensor, the *RS Pro* model was selected, as it is more affordable among models with similar specifications which typically costs around £7 to £15. The TL072CP Op Amp, which is the model in the lab, is chosen for its acceptable range of supply voltage, noise and low cost. As

‘

for the remaining components, they are selected with a similar principle, in total accounting for £11.63 of the prototyping budget, hence leaving room for modifications and optimization of the prototype in the next phase.

The objectives on the use of the remaining budget are as follows:

- Refinement of the new chassis
- Additional features for mobility
- Contingency budget for faulty components or in case of damage in testing

Project Management and Planning

Structure and Work Allocation

With the groundwork being done in the autumn term on the designing and planning of the EERover, steady progress has been made throughout the spring term on prototyping. To facilitate better division of labour with regards to each members' strengths and interests in the prototyping process, the management roles and technical parts have been adjusted and allocated accordingly. In terms of sensor design, it is split between four members - Jacob (Radio), Adrian (Acoustic), Justin (Acoustic), Rhys (Magnetic / Infrared), while the rest of the team - Arman and Vasilis, focused on the interface design, Arduino and the associated software designs.

With a clearer picture of the specific features to be implement after the client meeting and presentation, there has been a focus for each member on their specific technical role, this has mainly centred around the sensor design that is integral to the functionality of the rover.

Each circuit is tested independently with the provided Lizard in the lab throughout construction. This is to minimise the risk of error, whilst also enabling designs to change, so that an ideal signal output can be created for the Arduino to process.

For each deliverable there has been a similar approach to assigning work amongst members, with each member completing the section about their technical designs. Other non-technical tasks are then divided throughout the team, based on what each member is able to do.

Planning and Management

‘

Since the start of the project, there has been at least one group meeting per week, almost always on a Friday. During this meeting, general ideas about the rover are discussed, work is allocated, and any problems that may have arisen can also be discussed. During the rest of the week, WhatsApp is be the main channel of communication, often being used to schedule additional meetings between members. While the overall progress is overseen by the project manager, action points and minutes are made to ensure the progress of the project for the team.

Gantt Chart

While following the Gantt chart would be ideal, it is important to maintain flexibility at the same time, as unexpected delays may arise at any time. The following chart is an overview of the team's progress and timeline. Over the spring term, the major tasks have been broken down into smaller sub-sections for an easier assessment of the progress made, with ongoing failure analysis and improvement conducted.

Looking ahead, while the sensor designs are completed, and the Arduino code for mobility and intelligence is seeing great progress, they are to be further refined in the coming term in preparation for the demo. Apart from that, the main objective in the summer term for the team is to improve on the mobility and mechanical design aspect, as well as assembling the different sensor design and software together to

Figure 49 Gantt Chart

assess the rover's functionality when implemented as a whole. Once completed, less

‘

important elements of the design criteria may also be implemented, improving aesthetics for example.

References

Prof. Ken Leung – Introduction to Signals & Communications Lecture Slides

RS Components Ltd. - RS PRO Ultrasonic Proximity Sensors Datasheet

Microchip Technology Inc. - MCP6001/1R/1U/2/4 Datasheet

Osram Opto Semiconductors GmbH - SFH 300 Datasheet

Honeywell International Inc. - SS490 Series Linear Hall-Effect Sensor ICs Datasheet

‘

Appendices

Appendix 1: HTTP connection and IP configuration code

```
package com.crossobamon.rover;

import android.app.ProgressDialog;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class Setting extends AppCompatActivity {
   public ProgressDialog dialog; //The dialog, used when trying to get the
data from a IP.

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_setting);
       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
       setSupportActionBar(toolbar);
       setTitle("Configure IP address");

       getSupportActionBar().setDisplayHomeAsUpEnabled(true);
       final TextView input = (TextView) findViewById(R.id.ipTF);
       SharedPreferences                    preferences                    =
PreferenceManager.getDefaultSharedPreferences(this);
       String IP = preferences.getString("IP", "");
       input.setText(IP.replace("http://", "").replace("/", "")); //Fill in
the previous set IP in the label.

       //When the ok button is clicked, show the dialog and fire of a
request.
       Button ok = (Button) findViewById(R.id.okB);
       ok.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {

               dialog   =   ProgressDialog.show(Setting.this,   "Checking",
"Checking the IP.", false);
               String IP = input.getText().toString();
               IP = "http://" + IP + "/";
               new RequestTask().execute(IP);//Checking if the thing is up
           }
       });
```

‘

```
        TextView status = (TextView) findViewById(R.id.status);
        status.setText("Enter an IP address to connect to:");
    }

    //The request class.
    class RequestTask extends AsyncTask<String, String, String> {
        private String IP = "";//the variable to store the IP address to
store in the preferences.
        private boolean error = false;
        public static final String REQUEST_METHOD = "GET";
        public static final int READ_TIMEOUT = 15000;
        public static final int CONNECTION_TIMEOUT = 15000;

        @Override
        protected String doInBackground(String... uri) {

            IP = uri[0];    //Store the IP
            String responseString;
            String inputLine;

            try {
                //Create new URL object and open the connection channel
                URL url = new URL(uri[0]);
                HttpURLConnection    connection    =    (HttpURLConnection)
url.openConnection();

                connection.setRequestMethod(REQUEST_METHOD);
                connection.setReadTimeout(READ_TIMEOUT);
                connection.setConnectTimeout(CONNECTION_TIMEOUT);

                connection.connect();


                InputStreamReader         streamReader         =         new
InputStreamReader(connection.getInputStream());

                BufferedReader reader = new BufferedReader(streamReader);
                StringBuilder stringBuilder = new StringBuilder();

                while((inputLine = reader.readLine()) != null){
                    stringBuilder.append(inputLine);
                }

                reader.close();
                streamReader.close();

                responseString = stringBuilder.toString();
            }
            //Oops, something  went wrong.
            catch (IOException e) {
                e.printStackTrace();
                responseString = null;
            }

            return responseString;

        }
```

‘

```
        @Override
        protected void onPostExecute(String result) {

            dialog.dismiss();

            Toast.makeText(Setting.this, result, Toast.LENGTH_LONG).show();
            //If there isn't an exception.
            if(!error){
                    //Save the IP address.
                    SharedPreferences             preferences             =
PreferenceManager.getDefaultSharedPreferences(Setting.this);
                    SharedPreferences.Editor editor = preferences.edit();
                    editor.putString("IP", IP);
                    editor.apply();
                    Setting.this.finish();
            }
            else
                Toast.makeText(Setting.this,  "Oops,  something  went  wrong.
Error message: " + result,Toast.LENGTH_LONG).show();
        }
    }
}
```

Appendix 2: Button control code

```
package com.crossobamon.rover;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;

public class ButtonControls extends AppCompatActivity {
    public String ip;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_button_controls);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
```

‘

```
        //This gets the saved IP address, and puts it in the variable, if
not set, then it starts the Setting activity.
        SharedPreferences                    preferences                    =
PreferenceManager.getDefaultSharedPreferences(this);
        ip = preferences.getString("IP", "NONE");
        if (ip.equals("NONE")) {
            Intent intent = new Intent(this, Setting.class);
            startActivity(intent);
        } else {

            //requestAll();
            //This is when the app is started, you can already retrieve
data.
        }
        TextView textView = (TextView) findViewById(R.id.ipfield);
        textView.setText(ip);

        final Button reload = (Button) findViewById(R.id.reload);
        reload.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //When coming from the settings page, you need to press the
reload button.
                SharedPreferences                preferences                =
PreferenceManager.getDefaultSharedPreferences(ButtonControls.this);
                ip = preferences.getString("IP", "");
                //requestAll();

            }
        });
        final Button forwards = (Button) findViewById(R.id.forwards);
        forwards.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                new  ButtonControls.RequestTask().execute(ip  +  "F",  "1");
//Change the "GET" string to a string that is checked by the Arduino.
                //The "1" is for the correct textview, if you have more than
1,
                // you can set the correct destination of the output, or
what has to be done with it.
            }
        });
        final Button backwards = (Button) findViewById(R.id.backwards);
        backwards.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                new  ButtonControls.RequestTask().execute(ip  +  "B",  "1");
//Change the "GET" string to a string that is checked by the Arduino.
                //The "1" is for the correct textview, if you have more than
1,
                // you can set the correct destination of the output, or
what has to be done with it.
            }
        });
        final Button left = (Button) findViewById(R.id.left);
        left.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                new  ButtonControls.RequestTask().execute(ip  +  "L",  "1");
//Request the volts, put it in the other textview.
```

‘

```
                //The "1" is for the correct textview, if you have more than
1,
                // you can set the correct destination of the output, or
what has to be done with it.
            }
        });
        final Button right = (Button) findViewById(R.id.right);
        right.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                new  ButtonControls.RequestTask().execute(ip  +  "R",  "1");
//Request the volts, put it in the other textview.
                //The "1" is for the correct textview, if you have more than
1,
                // you can set the correct destination of the output, or
what has to be done with it.
            }
        });
        final Button stop = (Button) findViewById(R.id.stop);
        stop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                new  ButtonControls.RequestTask().execute(ip  +  "S",  "1");
//Request the volts, put it in the other textview.
                //The "1" is for the correct textview, if you have more than
1,
                // you can set the correct destination of the output, or
what has to be done with it.
            }
        });
        SeekBar speed = (SeekBar) findViewById(R.id.speed);
        speed.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {

            @Override
            public  void  onProgressChanged(SeekBar  speed,  int  progress,
boolean fromUser) {
                new   ButtonControls.RequestTask().execute(ip   +   "H"   +
String.valueOf(progress), "1");
            }

            @Override
            public void onStartTrackingTouch(SeekBar speed) {
                // TODO Auto-generated method stub
            }

            @Override
            public void onStopTrackingTouch(SeekBar speed) {
                // TODO Auto-generated method stub
            }
        });

    }

    @Override
    public  boolean  onCreateOptionsMenu(Menu  menu)  {  //some  blabla  for
creating the menu
        // Inflate the menu; this adds items to the action bar if it is
present.
```

‘

```
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override //Some blablabla for a pressed menu item.
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {//This starts the Setting class.
            Intent intent = new Intent(this, Setting.class);
            startActivity(intent);
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    //The actual task which requests the data from the Arduino.
//Can be fired with: new RequestTask().execute(String url, String MODE(1
for main label, rest you can change/add));
    class RequestTask extends AsyncTask<String, String, String> {
        private int MODE = 0; //1 == title of song , 2 == volume
        private boolean error = false;

        @Override
        protected String doInBackground(String... uri) {

            MODE = Integer.parseInt(uri[1]);    //Set the mode.

            String responseString;

            //Try to get the data from the LinkIt ONE. Do not change this
unless you know what this code does!
            try {
                URLConnection connection = new URL(uri[0]).openConnection();

                BufferedReader     br     =     new     BufferedReader(new
InputStreamReader((connection.getInputStream())));
                StringBuilder sb = new StringBuilder();
                String output;
                while ((output = br.readLine()) != null) {
                    sb.append(output);

                }
                responseString = sb.toString();

                //If something goes wrong.
            } catch (Exception e) {
                error = true;
                responseString = e.getLocalizedMessage();
            }

            return responseString;
```

‘

```
        }

        //After requesting the data.
        @Override
        protected void onPostExecute(String result) {
            super.onPostExecute(result);

            if (!error) {
                //Everything OK
                if (MODE == 0) {
                    //The MODE is something that can't be.
                    Toast.makeText(ButtonControls.this, "Starting the Async
went wrong", Toast.LENGTH_LONG).show();
                } else if (MODE == 1) {
                    TextView          textView          =          (TextView)
findViewById(R.id.ipfield);
                    textView.setText(ip);
                }

            } else {
                //A catch method caught an error.
                //Toast.makeText(MainActivity.this,  "Oops,  something  went
wrong.", Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

Appendix 3: Slider control code

```
package com.crossobamon.rover;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;

public class SliderControls extends AppCompatActivity {
    public String ip;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

‘

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_slider_controls);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        //This gets the saved IP address, and puts it in the variable, if
not set, then it starts the Setting activity.
        SharedPreferences                    preferences                    =
PreferenceManager.getDefaultSharedPreferences(this);
        ip = preferences.getString("IP", "NONE");
        if (ip.equals("NONE")) {
            Intent intent = new Intent(this, Setting.class);
            startActivity(intent);
        } else {

            //requestAll();
            //This  is  when  the  app  is  started,  you  can  already  retrieve
data.
        }
        TextView textView = (TextView) findViewById(R.id.ipfield);
        textView.setText(ip);

        final Button reload = (Button) findViewById(R.id.reload);
        reload.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //When coming from the settings page, you need to press the
reload button.
                SharedPreferences                preferences                =
PreferenceManager.getDefaultSharedPreferences(SliderControls.this);
                ip = preferences.getString("IP", "");
                //requestAll();
            }
        });

        SeekBar leftMotor = (SeekBar) findViewById(R.id.leftMotor);
        leftMotor.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {

            @Override
            public  void  onProgressChanged(SeekBar  speed,  int  progress,
boolean fromUser) {
                new   SliderControls.RequestTask().execute(ip   +   "J"   +
String.valueOf(progress), "1");
            }

            @Override
            public void onStartTrackingTouch(SeekBar speed) {
                // TODO Auto-generated method stub
            }

            @Override
            public void onStopTrackingTouch(SeekBar speed) {
                // TODO Auto-generated method stub
            }
        });

        SeekBar rightMotor = (SeekBar) findViewById(R.id.rightMotor);
        rightMotor.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
```

‘

```
            @Override
            public void onProgressChanged(SeekBar speed, int progress,
boolean fromUser) {
                new SliderControls.RequestTask().execute(ip + "K" +
String.valueOf(progress), "1");
            }

            @Override
            public void onStartTrackingTouch(SeekBar speed) {
                // TODO Auto-generated method stub
            }

            @Override
            public void onStopTrackingTouch(SeekBar speed) {
                // TODO Auto-generated method stub
            }
        });
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) { //some blabla for
creating the menu
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override //Some blablabla for a pressed menu item.
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {//This starts the Setting class.
            Intent intent = new Intent(this, Setting.class);
            startActivity(intent);
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    //The actual task which requests the data from the Arduino.
//Can be fired with: new RequestTask().execute(String url, String MODE(1
for main label, rest you can change/add));
    class RequestTask extends AsyncTask<String, String, String> {
        private int MODE = 0; //1 == title of song , 2 == volume
        private boolean error = false;

        @Override
        protected String doInBackground(String... uri) {

            MODE = Integer.parseInt(uri[1]);    //Set the mode.

            String responseString;
```

‘

```
            //Try to get the data from the LinkIt ONE. Do not change this
unless you know what this code does!
            try {
                URLConnection connection = new URL(uri[0]).openConnection();

                BufferedReader     br     =     new     BufferedReader(new
InputStreamReader((connection.getInputStream())));
                StringBuilder sb = new StringBuilder();
                String output;
                while ((output = br.readLine()) != null) {
                    sb.append(output);

                }
                responseString = sb.toString();

                //If something goes wrong.
            } catch (Exception e) {
                error = true;
                responseString = e.getLocalizedMessage();
            }

            return responseString;
        }

        //After requesting the data.
        @Override
        protected void onPostExecute(String result) {
            super.onPostExecute(result);

            if (!error) {
                //Everything OK
                if (MODE == 0) {
                    //The MODE is something that can't be.
                    Toast.makeText(SliderControls.this, "Starting the Async
went wrong", Toast.LENGTH_LONG).show();
                } else if (MODE == 1) {
                    TextView          textView          =          (TextView)
findViewById(R.id.ipfield);
                    textView.setText(ip);
                }

            } else {
                //A catch method caught an error.
                //Toast.makeText(MainActivity.this,  "Oops,  something  went
wrong.", Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

‘

Appendix 4: Metro board GET request handling code

```
#include <SPI.h>
#include <WiFi101.h>

bool on = false;

int period = 100;
int width;

char ssid[] = "ArmanSP4";
char pass[] = "sunnetcimahmut";
//char ssid[] = "EEERover";
//char pass[] = "exhibition";

int keyIndex = 0;                 // your network key Index number (needed
only for WEP)
IPAddress ip(192,168,0,22);     // unique ip address to access this Arduino
- MODIFY THIS

int status = WL_IDLE_STATUS;

WiFiServer server(80);

void setup() {
  //WiFi.config(ip);

  //Initialize serial and wait for port to open:
  Serial.begin(115200);

  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  // attempt to connect to WiFi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
```

‘

```
    // Connect to WPA/WPA2 network. Change this line if using open or WEP
network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
  server.begin();
  // you're connected now, so print out the status:
  printWiFiStatus();

  pinMode(1,OUTPUT);
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(6,OUTPUT);
  digitalWrite(3,1);
  digitalWrite(6,1);
}

void loop() {
  // listen for incoming clients
  WiFiClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    String currentLine = "";                    // make a String to hold
incoming data from the client
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("");
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
```

‘

```
          client.println("Connection: close");   // the connection will be
closed after completion of the response
          client.println("Refresh: 5");  // refresh the page automatically
every 5 sec
          break;
        }
        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
          currentLine = "";
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLine += c;      // add it to the end of the currentLine
          currentLineIsBlank = false;

          // Check to see if the client request was "GET /H" or "GET /L":
          if (currentLine.endsWith("GET /F")) {
            digitalWrite(1,1);
            digitalWrite(2,1);
          }
          if (currentLine.endsWith("GET /B")) {
            digitalWrite(1,0);
            digitalWrite(2,0);
          }
          if (currentLine.endsWith("GET /L")) {
            digitalWrite(1,0);
            digitalWrite(2,1);
          }
          if (currentLine.endsWith("GET /R")) {
            digitalWrite(1,1);
            digitalWrite(2,0);
          }
          if (currentLine.endsWith("GET /S")) {
            if(digitalRead(5) == 1) {
              digitalWrite(3,0);
              digitalWrite(6,0);
            } else {
              digitalWrite(3,1);
              digitalWrite(6,1);
            }
```

‘

```
          }
            if(currentLine.endsWith("GET /H0")) {
              analogWrite(3,0);
            }
            if(currentLine.endsWith("GET /H1")) {
              analogWrite(3,28);
            }
            if(currentLine.endsWith("GET /H2")) {
              analogWrite(3,57);
            }
            if(currentLine.endsWith("GET /H3")) {
              analogWrite(3,85);
            }
            if(currentLine.endsWith("GET /H4")) {
              analogWrite(3,113);
            }
            if(currentLine.endsWith("GET /H5")) {
              analogWrite(3,142);
            }
            if(currentLine.endsWith("GET /H6")) {
              analogWrite(3,170);
            }
            if(currentLine.endsWith("GET /H7")) {
              analogWrite(3,198);
            }
            if(currentLine.endsWith("GET /H8")) {
              analogWrite(3,227);
            }
            if(currentLine.endsWith("GET /H9")) {
              analogWrite(3,255);
            }
        }
      }
    }
    // give the client time to receive the data
    delay(1);

    // close the connection:
    client.stop();
    Serial.println("client disconnected");
  }
```

```
‘

}

void printWiFiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

‘

Appendix 5 – Previous PDS

The ticks indicate the point has been implemented so far, the crosses mean that the point will not be implemented

1. Performance

- Variable speed (Top Speed ~ 0.5 ms^{-1}?) ✓
- 4 wheel drive (or tank tracks) if feasible ×
- Continuously measuring for signals while power is provided ×
- Electrically powered by DC battery source ✓
- Necessary sensors for:
    - Radio signals ✓
    - Acoustic signals ✓
    - Infrared Signals ✓
    - Magnetic Fields ✓
- Able to filter out background noise while measuring signals. ✓
- Using the information collected from the sensors, we should be able to successfully identify each type of lizard. ✓

2. Environment

- Environment not fully known
- “The demo environment will have a floor that is mostly smooth with defects less than 2mm high.”
- Some obstacles must be avoided entirely (e.g. trees and plants), while others with a height of up to 15mm can be crossed to reduce travel distance (e.g. roots and creepers).
- Some areas may be inaccessible to rovers weighing more than 750g. All features of the demo environment will available for testing during the Summer Term”
- Possibility of a river which may have to be crossed

4. Maintenance

- Has to be able to withstand test of individual components. ✓
- Regular maintenance is desirable but not necessary
- Must be delivered if the tests fail
- All team members required to understand how to maintain their components and other components to a lesser extent if other members fail to show up. ✓

‘

- Rover must be modular for easy removal and maintenance of parts - each part must be individually removable. ✓
- No specialised tools for maintenance. ✓

5. Target Product Cost

- Total budget below £50. ✓
- Prototype budget below £15. ✓
- Final product budget £25. However, prototype parts will be included (Not complete)
- £10 left over for maintenance and additional parts. ✓

10. Manufacturing Facilities

- Limited by facilities in the lab
- No new facility needed. ✓
- No need to outsource. ✓
- Ideally, components self-designed instead of pre-assembled purchases. ✓
- 3D Printing and Laser cutting will be available. ✓

11. Size

- Not final, however current estimates for dimensions are:
    - Length: 25 cm
    - Width: 20 cm
    - Height: 15 cm
- Wheels large enough to provide enough traction to maneuver over 2 mm high defects with ease
- Able to cross 15 mm obstacles
- Chassis ideally sits 4 cm above ground

12. Weight

- Maximum weight: 750 grams
- Ideal weight range: 400 to 550 grams
- Centre of mass in the middle for a balanced vehicle
- Heavy enough to maintain traction when crossing river (and on all other surfaces)

17. Ergonomics

- Only mean of interaction by the user is through the remote control, i.e. laptop or any other external form of control. ✓ (Android app is made)
- Straightforward and we will review on this aspect on a consistent basis, most importantly upon completing each stage
- Consider use in actual environment

19. Quality and Reliability

‘

- Failure Analysis will be done in detail for the prototype to identify any possible parts that may go wrong . ✓ (Constant failure analysis being conducted)
- Reliability is essential for the commercialization purpose, e.g. for rigorous scientific research on lizards
- Part of the budget is reserved for improving the design ✓ (Most of budget is left for improvements)

23. Testing

- Testing for each sensors independently. ✓
- Testing for the whole operation of the EERover at the end. (Not conducted yet)
- Testing location similar to assessment location. ×
- Ability to distinguish similar signals is of paramount importance. ✓
- Effectiveness of mobility solution.

24. Safety

- No exposed wires
- No sharp edges

25. Company Constraints

- All components must be sourced through the EE supplier ✓
- Θ50 budget ✓
- Resolution of sensors dictated by budget ✓
- Manufacturing tools are limited (eg. we only have laser cutters and 3D printers, we don’t have PCB fabrication tools)
- Materials within the budget cannot use expensive materials such as gold. ✓
- Time is limited as we all have additional work to carry out, and it may be hard to organise full team meetings.