D4 Chinchilla - Team Report

Hugo McNally - hm6g17@soton.ac.uk
Matt Crossley - mac1g17@soton.ac.uk
Matthew Johns - mrj1g17@soton.ac.uk
Mark Warnants - mpw1g17@soton.ac.uk
Francis Wharf - few1g17@soton.ac.uk
William Webb - wcw1g17@soton.ac.uk
Tom Lam - yfl1u17@soton.ac.uk
Kexin Pan - kp1y18@soton.ac.uk

March 8, 2019

1 Challenge Solution Statement

The product was conceived to help deaf people navigate day-to-day conversations a little easier. Currently, deaf people must follow conversations through the use of lip reading and following contextual cues. This is made more difficult when many people talk at once, or dont get their attention first when talking. This solution enables deaf people to follow conversations more easily by visualising the sounds coming towards the listener from around the room, in an easy-to-understand way. It also provides a web interface to make interacting with the device easy, by providing a way to view the current sounds with high accuracy as well as the history of received sounds. It is simple to set up, and any firmware updates would be downloaded through the web interface.

This product is a table-top device, enclosed in its own case, with a sleek, white design. It has a diffused RGB LED ring on top to indicate the angles, amplitudes and frequencies of incoming sounds. It receives its inputs through four microphones on the top of the device which are sampled and sent to the system's Raspberry Pi by an ARM microcontroller. Cross-correlation and a fast-Fourier Transform are carried out on the Raspberry Pi to extract all the necessary information from the incoming signals. A web interface would be hosted by the Raspberry Pi which provides detailed read-outs of all the information gathered, and provides a secure file upload for firmware updates. There is a hardware calibration button in the centre of the device which, when pressed, triggers the collection of a ten second sound sample of the background noise. This is used to determine the level of noise which the device should subsequently ignore.

The system should be run on a 5V ± 400 mV battery to avoid any adverse effects on the operation of the system, particularly the microphones, pre-amps and analogue-to-digital converters (ADCs). The output of the microphone and pre-amp stages should be between 0V and 3.3V. After going through the ADC stage, it should transmit to the microcontroller in real-time at 50 kHz. This information should then be buffered on the microcontroller, and sent to the back-end on the Raspberry Pi after the buffer is filled up. This transfer should occur at no less than 120 kb/s. Analysis will then be carried out in the back-end and the angle of the sound should be output to an accuracy of 9°. An amplitude would also be provided as would frequency information. The resolution of the back-end outputs should be 8-bits at a frequency of 100 Hz.

2 System Design

The purpose of the system was to obtain audio from microphones, extract the angle of incidence of sounds, and display it on the hardware, and also a web interface. In order to make this happen, data must be sampled from an analogue microphone, and processed in a digital form which can be displayed to the user by some kind of computer. This naturally splits the project into a digital and analogue component.

For the analogue component of the system, instrumentation amplifiers were initially chosen to amplify the signal from the microphones, but audio pre-amps were later chosen instead. This is because the microphone output was not differential, as there was no negative rail.

Four microphones were used, as although a minimum of three are required to deduce the angle of a sound, four simplify the mathematics. The extra microphone also allows for increased accuracy. Using six or more was considered, but these ideas were rejected due to the extra hardware required.

It was also decided to build some analogue filters to remove mains and any other unwanted noise. The filtered audio can then be read by ADCs and sent to the digital components of the system over SPI.

For the digital component of the system, it was decided to use a Raspberry Pi Model 3, since every member of the team already had one, which was useful for development and testing. This presents some issues however. A Pi is well suited to performing the signal processing portion of the system, as it has a quad-core CPU with multipliers. However, unless an RTOS is installed, which would be awkward for also running a web server, a Raspberry Pi cannot sample audio from four separate ADCs at once with a consistent sampling rate. It was therefore decided to handle reading the ADCs on another piece of hardware.

An FPGA was considered for this role, as in theory it could very rapidly sample from a set of ADCs, and if one with multipliers was chosen, could even rapidly perform some of the signal processing.

However, this was rejected in favour of an ARM micro-controller, as a member of the team already had one to begin development on, and it would cost considerably less to find the required performance. Specifically an ARM Development Board was chosen, as it could easily interface with the Pi over USB to transfer both data and flash new firmware.

It would be impractical to process data from all four microphones on the Raspberry Pi in real-time, due to the huge amount of computing power required, so the decision was taken to buffer the ADCs continually for a period of time using the micro-controller. The micro-controller can then transfer the raw ADC readings to the Raspberry Pi over USB serial.

As a user interface, an LED ring was chosen, as it is easy to interpret. It is also aesthetically pleasing and cheap when compared to a display. A ring of pre-made addressable LED modules was chosen, as a bespoke solution would have been unnecessarily complex. The Pi can communicate with the LED ring using the one-wire protocol.

A web interface was also chosen, as the Pi can easily host a web-server, and it could be used to upload more firmware to the system, and provide the user more detailed information. Finally, a button was also chosen to the Pi, and used to trigger calibration. The back-end on the Pi could be shut down, or told to calibrate through a simple downward interface exposed as a FIFO.

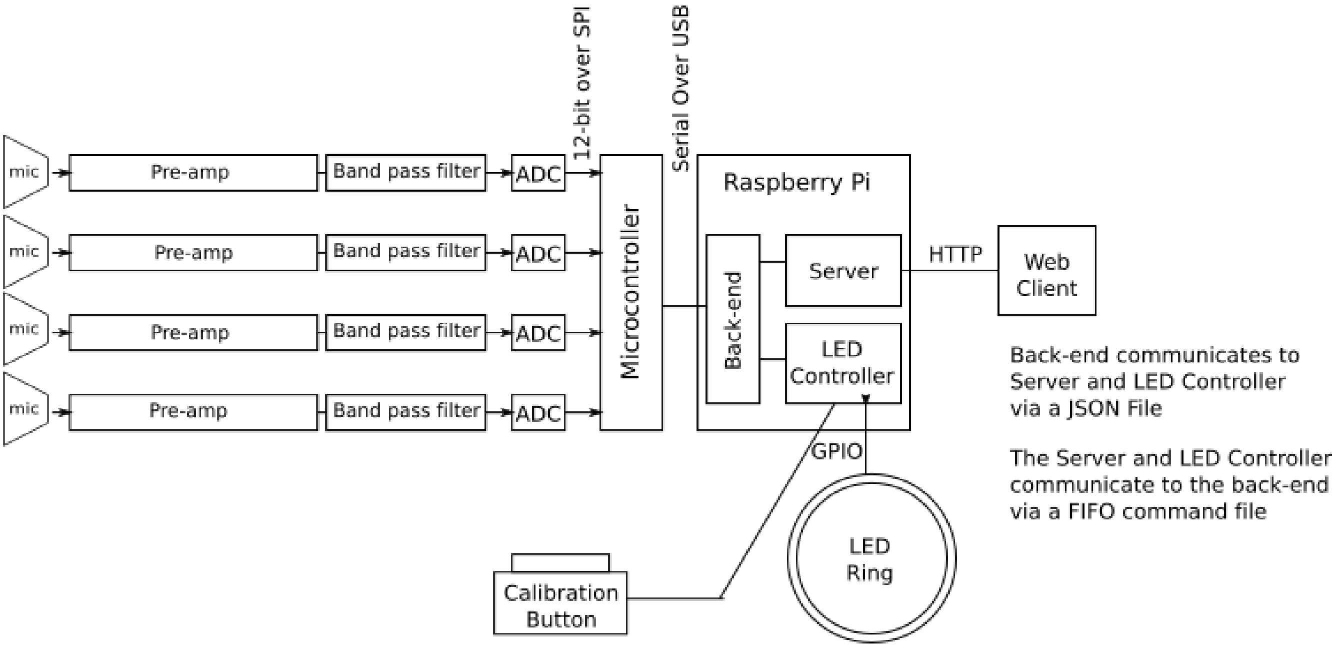


Figure 1: block diagram

3 Design Evaluation

3.1 Difficulty of Specification Attempted

The specification of this product was relatively complex. Despite the goal being relatively straightforward, the way that this was attempted, using multiple microphones and calculating phase differences of sound signals, required advanced and precise signal processing, with little room for timing error, as a high sampling rate was necessary to analyse audio.

3.2 Quality of the Electronic Design

The design relied heavily on embedded software, and as a result a lot of the electronics comprised of embedded development boards. There was realistically no alternative to this in such a shortage of time. The anti-aliasing filters were second-order, meaning the roll-off was only 40dB/decade. Given the very narrow range of frequencies audio occupies, this was not sharp enough to be very effective. A higher order filter was needed to remove the higher frequencies. The ADCs were chosen as 13-bit differential amplifiers, intended to give better common-mode rejection. However, single-ended ADCs would probably have sufficed, as would a lower resolution, given only eight bits were used.

3.3 Ease of Use

The design was incredibly easy to use as it was designed with elderly people in mind, as they are the largest demographic for hearing problems. There was only a single on-off push button for power, and a single large button on the top for calibration. Audio data was displayed on the LED ring on the top in an intuitive way. The case was slightly larger than ideal when considering portability, but it was not heavy enough to be cumbersome. The internals were accessed incredibly easily by rotating the lid a small amount to release it.

3.4 Creativity and Innovation of the Designed Product

This is a product that we believe has not yet been brought to market, and is therefore very innovative. The methods of solving the problem are not revolutionary, but putting them together in this specific product aimed at improving the lives of people with hearing problems is a new concept.

3.5 Aesthetics

The case was designed to give it a clean and minimalist look, and the LEDs were covered by diffuser plastic help the lights blend together in a smooth, attractive manner. This had the result of our enclosure being one of the best (if not the best!) looking out of all the teams. The aesthetics were improved further using LED animations on startup and calibration.

3.6 Cost

Nothing in the device was made of specialist components, and thus everything was relatively cheap. The major source of cost was the use of the MCU and Raspberry Pi boards, which would not be used in a production model. They were also both more powerful than was required for the project, but as exact performance requirements were not know at the time of design, they were bought to allow some leeway. Using a plastic case makes the design cheaper to mass produce.

3.7 Reliability

The design had a calibration feature implemented in software, which theoretically should allow it to function reliably in most social environments. Situations in which it may not be reliable include high levels of vibration of the surface on which it is placed, or if there is a constant, loud background noise such as heavy machinery. It would not be expected to work outside unless there was little wind. Tapping the case may reduce reliability unless padding was introduced between the case and the microphones. The case is robust enough to protect the interior electronics during transport and handling. There is nothing on the outside of the case likely to be broken off.

4 Quality Factor

4.1 Costing

Appendix B contains a breakdown of the costing of the unit. This not only includes the breakdown of costs associated with producing a single unit but also the costs incurred with manufacturing the product over a period of time. The enables the cost of setup fees and development costs to be amortised over a batch of units, in this case 1000 units, to better estimate a recommended retail price. A number of assumptions were made, and are stated in Appendix B that certain work was done to reduce the price of the unit based on the manufacturing methods that would be used for a production run of that size. 1000 units was chosen as it represents a large production run for this product. This is considering that it is in the accessibility market where it is purchased more through necessity by a small niche as opposed to being actively market to the general public as a helpful product. The recommended retail price here was estimated to be 405.77. This results in profit of 112.77 per unit sold.

4.2 Marketing

As mentioned, the product only appeals to environments where there are a number of people sitting in static locations. This means the product appeals largely to conferences in a business setting where a hearing impaired person is involved. This limits the target audience quite substantially and the best route for marketing would most probably be through existing companies that already specialise in accessibility based products.

4.3 Conformance Marking

CE compliance requires the product to visible be marked somewhere on the outer casing with the official CE logo once conformed. The obvious place for this is the bottom of the unit. The product would be required to undergo a round of testing at a verified test house, TUV for example, to ensure it meets requirements for a number of standards such as radiated, conducted and induced emissions. Since the product uses wifi for communication, it would have to be tested under the 'intentional radiator' category which may incur additional cost or testing over what is in the cost breakdown as in Appendix B. Quotations would be necessary to verify this.

5 Final Product

First of all, to compare our specification with what we have achieved, the back-end successfully passed all the tests using the simulated data. It was shown to work for multiple angles with frequencies up to 2000Hz. For the back-end, it was specified that it needed to calculate the angle with the tolerance ±9°. It exceeds this and has an accuracy of 0.01° for a single sine wave. Also, the microphone can receive the spoken voice and the pre-amp can output the voltage between 0 to 3.3V from it. Moreover, the ADC to MCU communication can work and samples the frequency larger than 50kHz. Furthermore, we specify that the buffer communicates with back-end at over 120kb/s. It again exceeds this and can able to communicate at 172kb/s. As well as this, the WebUI is able to display the amplitude and angle of the incoming sound. The device is fully constructed and we are able to put all the components into the 3D-printed case. In terms of the LED ring, we can use it to represent the signal direction, the frequency and the amplitude by using test data.

In terms of which we did not achieve, the WebUI cannot update at a rate greater than 60fps. Secondly, the final device cannot reject artificial ambient noise and therefore cannot indicate the correct direction of the signal in a noisy room.

With the final product, theoretically once an object emitted a sound the four microphones inside the device will amplify it, and the ADCs will convert the sound into a set of real values within a range from 0 to 255. Then it will pass to a microcontroller which controls the data sent to the Raspberry Pi. Once the Raspberry Pi receives the sample values from the microcontroller, it will compute the program compiled in it. In addition, it finds the direction of the sound by using cross-correlation and the delay between two signals in order to calculate the angle of the sound that it comes from. The frequency of the sound is calculated using the Fourier Transform. Finally, the device will be able to show the direction of the sound by turn on the LED light and the frequency of the sound which is represented by the color of the LED.

Finally, for the further extensions of our device, we would like to add filters to remove frequencies higher than 12kHz which are not common in audio. Also, we want to use the frequency to differentiate different sounds from the microphones and improve the speed and accuracy of LED output. Lastly, we want the device able to sense the position of sounds in absolute space, either distance or magnitude of the vertical angle.

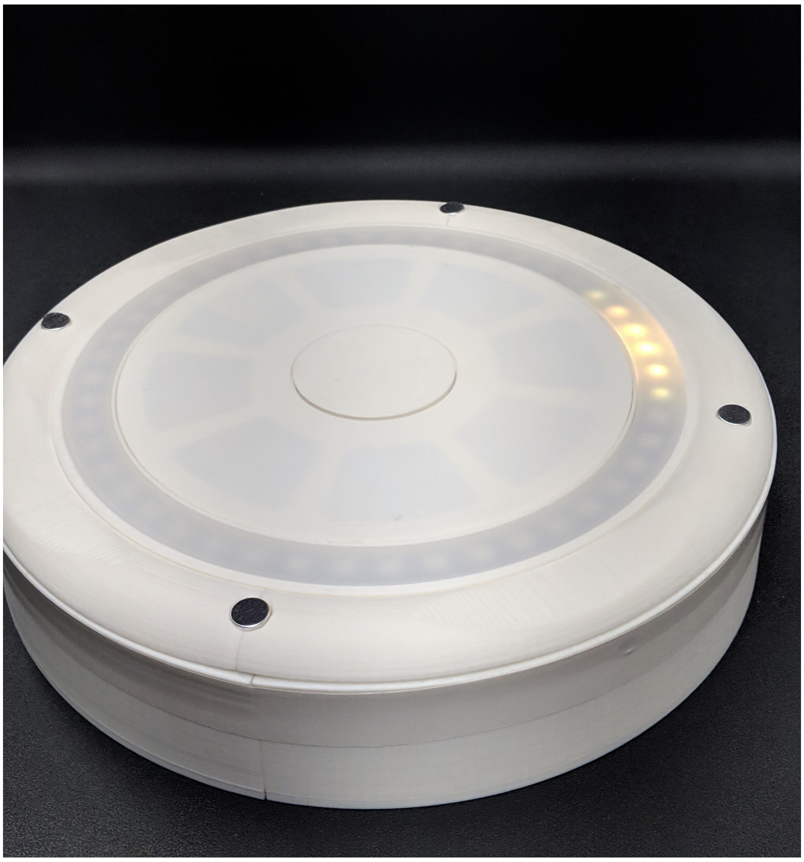
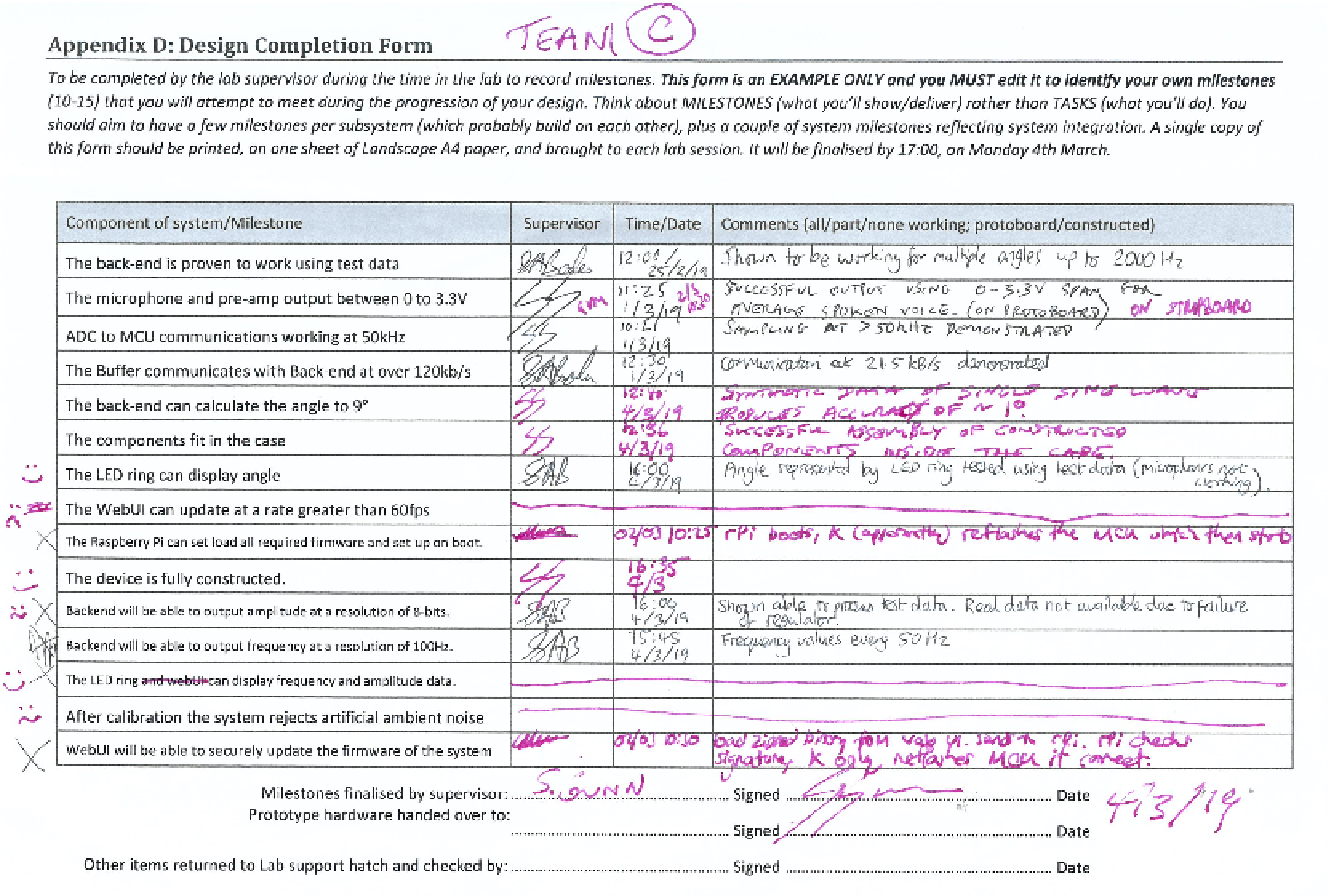


Figure 2: Photo of the final product with LED light

A Design Completion Form



Appendix D: Design Completion Form

TEAM C

To be completed by the lab supervisor during the time in the lab to record milestones. ***This form is an EXAMPLE ONLY and you MUST edit it to identify your own milestones*** *(10-15) that you will* ***attempt to meet*** *during the progression of your design. Think about MILESTONES (what you'll show/deliver) rather than TASKS (what you'll do). You should aim to have a few milestones per subsystem (which probably build on each other), plus a couple of system milestones reflecting system integration. A single copy of this form should be printed, on one sheet of Landscape A4 paper, and brought to each lab session. It will be finalised by 17:00, on Monday 4th March.*

Component of system/Milestone	Supervisor	Time/Date	Comments (all/part/none working; protoboard/constructed)
The back-end is proven to work using test data		12:00 / 25/2/19	Shown to be working for multiple angles up to 2000 Hz
The microphone and pre-amp output between 0 to 3.3V	GM	11:25 1/2/19 2/3 10:30	SUCCESSFUL OUTPUT USING 0-3.3V SPAN FOR AVERAGE SPOKEN VOICE. (ON PROTOBOARD) ON STRIPBOARD
ADC to MCU communications working at 50kHz		10:21 1/3/19	SAMPLING AT > 50kHz DEMONSTRATED
The Buffer communicates with Back end at over 120kb/s		12:30 1/3/19	Communication at 21.5 kB/s demonstrated
The back-end can calculate the angle to 9°		12:40 4/3/19	SYNTHETIC DATA OF SINGLE SINE WAVE PRODUCES ACCURACY OF ~ 1°.
The components fit in the case		12:36 4/3/19	SUCCESSFUL ASSEMBLY OF CONSTRUCTED COMPONENTS INSIDE THE CASE.
The LED ring can display angle		16:00 6/3/19	Angle represented by LED ring tested using test data (microphones not working).
The WebUI can update at a rate greater than 60fps			
The Raspberry Pi can set load all required firmware and set up on boot.		02/03 10:25	rPi boots, & (apparently) reflashes the MCU which then starts
The device is fully constructed.		16:35 4/3	
Backend will be able to output amplitude at a resolution of 8-bits.		16:00 4/3/19	Shown able to process test data. Real data not available due to failure of regulator.
Backend will be able to output frequency at a resolution of 100Hz.		15:45 4/3/19	Frequency values every 50 Hz
The LED ring ~~and webUI~~ can display frequency and amplitude data.			
After calibration the system rejects artificial ambient noise			
WebUI will be able to securely update the firmware of the system		04/03 10:30	Load signed binary from web UI. Send to rPi. rPi checks signature, & only reflashes MCU if correct.

Milestones finalised by supervisor: S. GUNN Signed .... Date 4/3/19

Prototype hardware handed over to: .... Signed .... Date

Other items returned to Lab support hatch and checked by: .... Signed .... Date

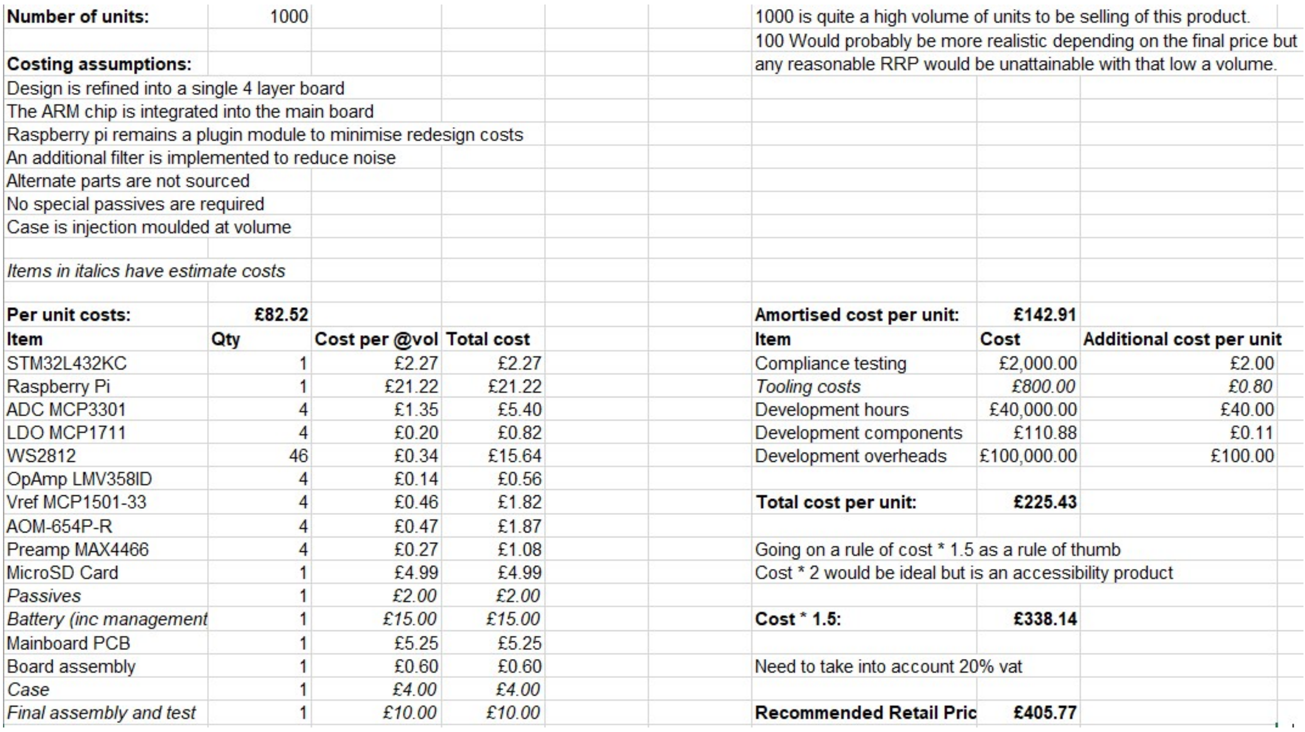
B Project Completion Form

(see imported PDF on pages below)

Project Completion Form

Cost Estimates

Please give detailed calculations and estimates of the overall cost of your actual design below. Take care to include person-hour estimates for your software, board production and debugging, as well as your components and consumables. You should also estimate the production cost of your final unit (you may assume a large quantity are to be produced), the market price and determine how many need to be sold to be profitable. Account for any differences between the actual values and the values given in your original project proposal form.



Number of units: 1000

1000 is quite a high volume of units to be selling of this product.
100 Would probably be more realistic depending on the final price but any reasonable RRP would be unattainable with that low a volume.

Costing assumptions:
Design is refined into a single 4 layer board
The ARM chip is integrated into the main board
Raspberry pi remains a plugin module to minimise redesign costs
An additional filter is implemented to reduce noise
Alternate parts are not sourced
No special passives are required
Case is injection moulded at volume

Items in italics have estimate costs

Per unit costs: **£82.52**

Item	**Qty**	**Cost per @vol**	**Total cost**
STM32L432KC	1	£2.27	£2.27
Raspberry Pi	1	£21.22	£21.22
ADC MCP3301	4	£1.35	£5.40
LDO MCP1711	4	£0.20	£0.82
WS2812	46	£0.34	£15.64
OpAmp LMV358ID	4	£0.14	£0.56
Vref MCP1501-33	4	£0.46	£1.82
AOM-654P-R	4	£0.47	£1.87
Preamp MAX4466	4	£0.27	£1.08
MicroSD Card	1	£4.99	£4.99
Passives	1	*£2.00*	*£2.00*
Battery (inc management	1	*£15.00*	*£15.00*
Mainboard PCB	1	£5.25	£5.25
Board assembly	1	£0.60	£0.60
Case	1	*£4.00*	*£4.00*
Final assembly and test	1	*£10.00*	*£10.00*

Amortised cost per unit: **£142.91**

Item	**Cost**	**Additional cost per unit**
Compliance testing	£2,000.00	£2.00
Tooling costs	*£800.00*	*£0.80*
Development hours	£40,000.00	£40.00
Development components	£110.88	£0.11
Development overheads	£100,000.00	£100.00

Total cost per unit: **£225.43**

Going on a rule of cost * 1.5 as a rule of thumb
Cost * 2 would be ideal but is an accessibility product

Cost * 1.5: **£338.14**

Need to take into account 20% vat

Recommended Retail Pric **£405.77**

Design Changes

Briefly summarise any design changes your team had to make to the original design proposal, in order to get your system to work. Do not go into vast detail, as it is anticipated that this will be done by the individuals responsible for these components of the design in the formal report.

Amplifier

An instrumentation amplifier was not chosen to act as the preamp as it is primarily a differential type amplifier. Since the signal from the microphone capsule was with referenced to ground to begin with, no gains were made with the added complexity of an instrumentation amplifier. Filters also needed to be implemented before or in the preamp to eliminate noise while it is still a small signal, and the architecture for the instrumentation amplifier would make it harder to design such filters.

Filters

It makes sense to adapt the preamp to only amplify the signal frequencies in the first place, as opposed to amplifying everything and then filtering afterwards. The preamp design filter, however, consists of only a second-order high-pass and first-order low-pass. The rolloff for the low-pass is too slow and an additional filter would have increased this to a second- or third-order rolloff, leading to fewer higher frequencies being samples. These higher frequencies had proved problematic.

Actual Project Activities

Please list the activities that took place during your laboratory time, and indicate when they occurred, and who did them. The 'Initials'column must specify only one person. If two people worked on the same subsystem or task, you should list this as two separate activities, and be clear about what each individual contributed.

Activity	Initials	Fri am	Fri pm	Mon am	Mon pm	Tue	Wed	Thu	Fri am	Fri pm	Mon am	Mon pm
Backend: Output interface	FW	X										
Backend: Generating test data with script	FW	X										
Backend: Tweaking signal processing	FW		X	X	X	X	X	X	X			
Backend: Hardware grunt work	FW								X	X		
Backend: Interface file management	FW						X					
Backend: Peak finding code	FW			X								
Backend: Debugging and configuration	FW								X	X	X	X
Buffer: Add serial implementation code	MJ	X	X									
Buffer: Test data transfer speed on own board	MJ			X								
Buffer: ADC behaviour with own board	MJ				X							
Buffer: Diagnose and fix timing glitch	MJ					X	X	X				
Buffer: Write and test logic test program	MJ							X				
Buffer: Test and tune sampling speed on new board with ADCS	MJ								X			
Construct and test ADC board	MJ									X		

Construct and test complete system	MJ										X	X
LED ring code (angle and amplitude)	MPW	X	X									
LED code tested and debugged	MPW			X								
Calibration button code	MPW				X							
Case design	MPW					X	X	X	X	X		
LED ring code written, tested and debugged (frequency)	MPW										X	
Calibration button code tested and debugged	MPW										X	
Helped with some Pi configuration	MPW											X
Raspberry Pi: Web file upload	MC	X										
Raspberry Pi: FFT webUI developed with generated data	MC			X								
Raspberry Pi: Web back-end file handling and install scripts	MC				X							
Raspberry Pi: Major install script debugging and workarounds	MC					X						
Raspberry Pi: File signing and permissions workarounds	MC						X	X				
Raspberry Pi: Working file system from upload to install (debugging)	MC							X	X			
Raspberry Pi: Additional functionality (Pi restart and calibration)	MC									X		
Raspberry Pi: Web server testing, debugging and tidying	MC										X	X
Raspberry Pi: Full image backup	MC										X	

Raspberry Pi: Apply simulated real-time data to FFT using JQuery	MC											X
Hardware: Finalise layout of hardware within the case	WW									X	X	X
Hardware: Finalise the schematic	WW		X	X								
Hardware: Testing design	WW		X	X	X	X	X	X	X	X	X	X
Hardware: Testing final board	WW										X	X
Hardware: LED ring construction	WW							X	X			
Hardware: Case design and printing	WW							X	X	X	X	
Hardware: Any final construction and assembly	WW									X	X	X
Raspberry Pi: Implement the DFT	YFL	X	X									
Raspberry Pi: Implement the cross-correlation function	YFL	X	X									
Raspberry Pi: Testing DFT and CC code	YFL	X	X									
Raspberry Pi: Tweaking and debugging signal processing	YFL			X	X	X	X		X	X		
Hardware construction	YFL							X	X			
Assist debugging of buffer	YFL						X					
Assist hardware debugging	YFL										X	X
Researching use of microphone and amplifier chip	KP	X										
Analogue: Construct test board	KP		X	X	X							
Analogue: Test design	KP				X	X						

Analogue: Change filter cutoff frequency	KP						X	X				
Analogue: Construct final filter boards	KP							X	X			
Analogue: Test final boards	KP								X	X	X	
Analogue: Helping with soldering for final product	KP										X	X

Discrepancy in Project Activities

Comment on any major differences between the planned and actual project activities.

William Webb: Construction happened late due to parts arriving later than expected, and noise in signal that turned out to not be a problem when working in labs due to better equipment.

Matthew Johns: It was decided to send a precompiled binary to the Pi to flash onto the buffer MCU, so the mbed CLI tools did not need to be set up on the Pi to compile firmware updates. The timing glitch that appeared on Monday was critical and needed fixing. Testing was late due to other subsystems being behind.

Francis Wharf: Getting file management interfaces working took longer than expected, but making the test data script took less time than expected.

Matt Crossley: Software on the Pi took longer due to issues with Linux permissions. PHP was hard to debug due to being a PHP novice. Due to construction for the product being delayed, it was not possible to program real-time graphs and displays until the very end of project, due to poor communication between the web side and back-end.

Yiu Fai Lam (Tom): No major differences in plans

Mark Warnants: The calibration button code took longer than expected due to being new to Python, as did the case design. Testing was delayed due to construction being behind.

Kexin Pan: Using a battery meant there was no negative power rail, so the active filter had to be changed into a passive filter. The required cutoff frequency was reduced midway through form 15kHz to 12kHz so this took more time.

Assessment of Effort

The table below will be used as an indication how team marks should be allocated across the team.

Name	Signature	% of effort
Hugo McNally		12.5
Francis Wharf		12.5
Matthew Johns		12.5
Kexin Pan	Kexin Pan	12.5
Mark Warnants		12.5
William Webb		12.5
Matt Crossley		12.5
Yiu Fai Lam	Lam	12.5

C Circuit Diagrams

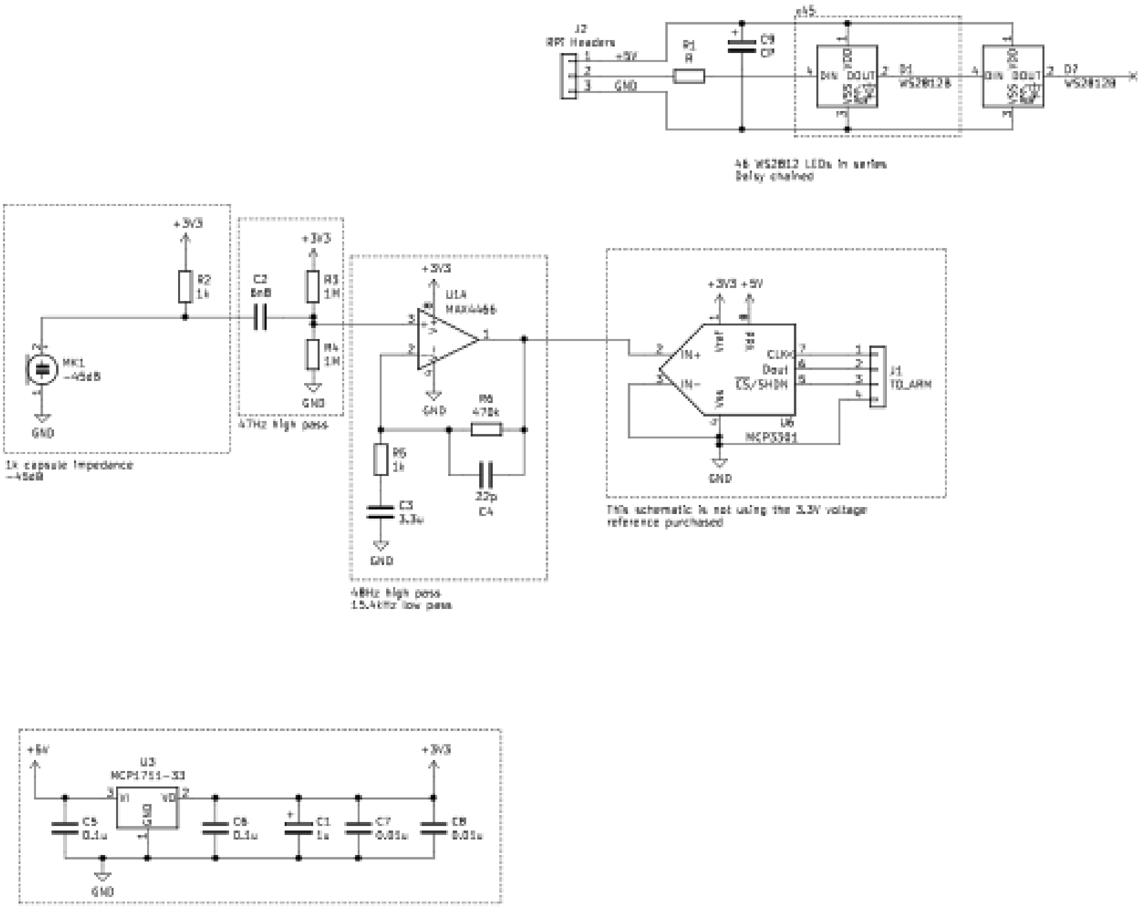


Figure 3: Schematic

D Software Listings

D.1 Buffer

arm_main.cpp

```
// Code that runs on the Arm MCU buffer. Reads data from ADCs using SPI, then
// formats it into a serial packet and sends it to the back-end over serial.
// Maintains sampling rate of >50kHz.
// Written by Matthew Johns (mrj1g17@soton.ac.uk)
#include "mbed.h"

// Constants for pin numbers
#define ADC0_PIN A0
#define ADC1_PIN A1
#define ADC2_PIN A2
#define ADC3_PIN A3
#define CS_PIN D5
#define CLK_PIN D6

```

```
#define BUFFER_SIZE 1024
#define DATA_BITS 9
#define NUM_MICS 4
#define CLK_DELAY 8
#define BAUD 460800
#define START_BYTE 0xFF
DigitalOut cs(CS_PIN);
DigitalOut clk(CLK_PIN);
Serial serial(USBTX, USBRX);

char serial_buffer[NUM_MICS*BUFFER_SIZE];
uint16_t current_sample[NUM_MICS];
uint16_t samples_buffer[BUFFER_SIZE*NUM_MICS];
uint16_t top = 0;
uint8_t stall=0;

DigitalIn a0(ADC0_PIN);
DigitalIn a1(ADC1_PIN);
DigitalIn a2(ADC2_PIN);
DigitalIn a3(ADC3_PIN);

// Uses a bit-bashing method similar to SPI to read the values from all four
// ADCs. Then the values are manipulated into the correct format to be
// represented by a single int variable each.// These can then be sent over
// serial.
void read_samples()
{
    // Pulse clock once to get ADC sample going
    cs = 0;
    clk = 1;
    clk = 0;

    for(uint8_t i = 0; i < DATA_BITS; i++)
    {
        clk = 1;

    // Introduce delay to maintain square clock pulse and keep SPI clock below
    // 1MHz. #pragmas are to try and stop it being optimised out by compiler
#pragma GCC push_options
#pragma GCC optimze ("no-unroll-loops")
        for(uint8_t d=0; d <= CLK_DELAY; d++) {stall=d; __asm volatile("NOP");}
#pragma GCC pop_options

        clk = 0;

        current_sample[0] += a0 << (DATA_BITS-(i+1));
        current_sample[1] += a1 << (DATA_BITS-(i+1));
        current_sample[2] += a2 << (DATA_BITS-(i+1));
        current_sample[3] += a3 << (DATA_BITS-(i+1));

        // Above logic produces enough of a delay to not need an extra one
    }
    // The MSB is a sign bit, and should always be 0. If it isn't, the bit may
    // have been corrupted and the sample should be set to 0.
    for(uint8_t i = 0; i < NUM_MICS; i++)
    {
        if(current_sample[i] >> (DATA_BITS-1))
            current_sample[i] = 0;
    }

    cs = 1;
}
```

```

// Takes data in samples_buffer and makes it suitable for transmission, then
// sends it
void send_serial()
{
    serial.putc(START_BYTE);
    for(uint16_t i = 0; i < BUFFER_SIZE*NUM_MICS; i++)
    {
        // Truncating sample to so the serial only sends the necessary 8 bits
        serial_buffer[i] = (uint8_t)(samples_buffer[i]);

        // Start byte is 0xff (255). If the sample == 255 it must be made 254
        // to avoid confusion. It's a small error so shouldn't cause issues
        if(serial_buffer[i] == 255)
            serial_buffer[i] = 254;

        // Sending the value of the sample
        serial.putc(serial_buffer[i]);
    }

    // Don't need to reset the samples buffer, as it will be overwritten. Just
    // say the top is the first element and new samples will be stored there
    top = 0;
}

int main()
{
    // Initial setup to keep the ADCs happy
    cs = 1;
    clk = 0;

    serial.baud(BAUD);

    // mbed OS scheduler thread suspected to be messing with timings. Make
    // everything critical except serial tranmissions to ensure the sampling
    // rate is maintained
    CriticalSectionLock::enable();

    for(;;)
    {
        read_samples();

        // Adding the current sample to the buffer once it is retrieved
        for(uint8_t i = 0; i < NUM_MICS; i++)
        {
            samples_buffer[top] = current_sample[i];
            top++;
        }

        // Checks to see if the buffer is full. If so, sends serial.
        // Have to disable CriticalSectionLock, as serial uses interrupts which
        // cannot work when locked (crashes OS). Locked straight after though
        if(top == BUFFER_SIZE*4)
        {
            CriticalSectionLock::disable();
            send_serial();
            CriticalSectionLock::enable();
        }
    }

    return 0;
}
```

logic_test_program.cpp

```cpp
// Simple command line program to test the bit-shifting logic of the buffer
// code. Data on the ADC inputs is simulated using the a0-a3 arrays. Prints
// outputted char at the same time as it would send over serial.
// Structure is generally very similar to MCU code to be more comparable. More
// information on code function can be found in comments of arm_main.cpp.
// Written by Matthew Johns (mrj1g17@soton.ac.uk)
#include <iostream>
using namespace std;

#define BUFFER_SIZE 1
#define DATA_BITS 9
#define NUM_MICS 4

uint8_t serial_buffer[NUM_MICS*BUFFER_SIZE];
uint16_t current_sample[NUM_MICS];
uint16_t samples_buffer[BUFFER_SIZE*NUM_MICS];
uint16_t top = 0;

// These test the important cases:
uint8_t a0[DATA_BITS] = {0,0,0,1,0,0,1,1,1}; // Standard number
uint8_t a1[DATA_BITS] = {1,1,1,0,1,1,0,1,0}; // Negative reading
uint8_t a2[DATA_BITS] = {0,1,1,1,1,1,1,1,1}; // 255 value
uint8_t a3[DATA_BITS] = {0,0,0,0,0,0,0,0,0}; // 0 value


void sample()
{
    for(uint8_t i = 0; i < DATA_BITS; i++)
    {
        current_sample[0] += a0[i] << (DATA_BITS-(i+1));
        current_sample[1] += a1[i] << (DATA_BITS-(i+1));
        current_sample[2] += a2[i] << (DATA_BITS-(i+1));
        current_sample[3] += a3[i] << (DATA_BITS-(i+1));
    }

    for(uint8_t i = 0; i < NUM_MICS; i++)
    {
        if(current_sample[i] >> (DATA_BITS-1))
            current_sample[i] = 0;
    }
}

void serial()
{
    for(uint16_t i = 0; i < BUFFER_SIZE*NUM_MICS; i++)
    {
        // cout << "samples_buffer: " << samples_buffer[i] << endl;
        serial_buffer[i] = (uint8_t)(samples_buffer[i]);

        if(serial_buffer[i] == 255)
            serial_buffer[i] = 254;

        // Have to cast serial_buffer[] else it tries to print like a char.
        // (Gives nonsense/unhelpful output)
        cout << "Sample␣" << i << ":␣" << (int)serial_buffer[i] << endl;

    }
    top = 0;
}
```

```

int main()
{
    sample();

    for(uint8_t i = 0; i < NUM_MICS; i++)
    {
        // cout << "top: " << top << endl;
        samples_buffer[top] = current_sample[i];
        top++;
    }

    if(top == BUFFER_SIZE*4)
        serial();

    return 0;
}
```

D.2 Signal Processing

conf.h

```
#define CONF_ROOT "/tmp/"

#define CONF_INPUT "/dev/ttyACM0"
#define CONF_CTL CONF_ROOT "chinchilla-backend-ctl"
#define CONF_SOUND CONF_ROOT "chinchilla-sounds"
#define CONF_FFT CONF_ROOT "chinchilla-fft"
```

main.c

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include "sample.h"
#include "xcorr.h"
#include "errno.h"
#include "string.h"
#include "conf.h"

/* Make and return a stream pointed to the backend control file */
FILE *ctl_file(void)
{
    FILE *f;

    if (mkfifo(CONF_CTL, 0666) == -1)
    {
        if (errno != EEXIST)
            printf("Error,␣cannot␣make␣backend-ctl␣fifo:␣%s\n", strerror(errno));
    }

    f = fopen(CONF_CTL, "r");

    return f;
}

/* Cleanup temporary files and fifos once I close */
void clean_files(void)
{
    FILE *f;

    /* Delete the control file */
    unlink(CONF_CTL);

    /* Just empty the CONF_SOUND file */
    f = fopen(CONF_SOUND, "w");
    fwrite("", 1, 0, f);
}

void main(void)
{
    int running;
    FILE *ctlf;
    xcorr_manager_s manager;
    /* Make child threads */
    xcorr_manager_init(&manager);

    running = 1;
    /* Open a control file input */
    ctlf = ctl_file();

    while (running)
    {
        char line[16];
        char *chr, *end;
        chr = &line[0];
        end = &line[sizeof(line) - 1];
        memset(line, 0, sizeof(line));

        /* Read a line from the control file */
        while (chr < end)
        {
            int cint;
            cint = fgetc(ctlf);

            if (cint == -1)
            {
                /* Clear any errors so we don't get stuck re-reading */
                clearerr(ctlf);
                usleep(100000);
                break;
            }

            *(chr++) = (unsigned char)cint;
        }

        /* If there is a stop command, stop running */
        if (memcmp("stop", line, 4) == 0)
            running = 0;

        /* Run a calibration routine if needed */
        if (memcmp("calibrate", line, 4) == 0)
        {
            manager.calibrating = 1;
            printf("CALIBRATING\n");
            sleep(5);
            printf("DONE\n");
            manager.calibrating = 0;
        }

    }
    fclose(ctlf);

    /* Kill our child thread(s) */
    xcorr_manager_kill(&manager);

    /* Cleanup */
    clean_files();
}
```

sample.h

```
#if !defined(SAMPLE_H)
# define SAMPLE_H
# include <stdio.h>

# define SAMPLE_SIZE 1024
# define XCORR_LEN 151
# define NUM_MICS 4
# define NUM_XCORR (NUM_MICS - 1)
# define MAX_PEAKS 4
# define SAMPLE_RATE 60000

// This is big, so avoid storing it on stack memory as much as possible :)
typedef struct packet packet_s;

struct packet
{
    int data[NUM_MICS][SAMPLE_SIZE];
    int xcorr[NUM_XCORR][XCORR_LEN];
};

int sample_packet_recv(packet_s *pkt, FILE *stream);

int sample_match_peaks(packet_s *pkt);

#endif
```

sample.c

```
#include "sample.h"
#include "sound.h"
#include <string.h>
#include <errno.h>
#include <unistd.h>

/* Use select to wait until a stream is readable. *
 * There is a 1 second timeout on this function. *
 * It returns 1 if the stream has become readable, *
 * and 0 otherwise. */
int wait_for_file(FILE *stream)
{
    int fn;
    /* Timeout */
    struct timeval tout = { .tv_sec = 0, .tv_usec = 1000000 };
    fd_set waitfor;
    fn = fileno(stream);

    /* Set the appropriate bits in the fd_set */
    FD_ZERO(&waitfor);
    FD_SET(fn, &waitfor);

    /* Wait for the fd */
    if (select(fn + 1, &waitfor, NULL, NULL, &tout) == 1)
    {
        return 1;
    }

    return 0;
}

int sample_packet_recv(packet_s *pkt, FILE *stream)
{
    int c, n;
    size_t micnum, samplenum;
    micnum = 0;
    samplenum = 0;
    n = 0;

    // If there's clearly bullshit, run away
    while ((++n) < (100 * SAMPLE_SIZE))
    {
        // This is an experimental optimization, kill it if you want <3 - francis
        if (!wait_for_file(stream))
        {
            puts("Timed␣out␣waiting␣for␣input");
            return -1;
        }

        /* Get the next character */
        c = fgetc(stream);

        if (feof(stream))
        {
            /* Clear EOF or we'll continually read EOF chars */
            clearerr(stream);
            return -1;
        }
        else if (c == EOF)
        {
            /* Other errors */
            clearerr(stream);
            printf("Error␣reading:␣%s\n", strerror(errno));
            return -1;
        }

        /* The starting character */
        if (c == 0xff)
        {
            /* It is expected as the first character */
            if (micnum == 0 && samplenum == 0)
                continue;
            /* But not in other positions */
            else
            {
                printf("Unexpected␣0xff\n");
                return -1;
            }
        }

        /* Read the data point */
        pkt->data[micnum][samplenum] = (int)c;
        micnum += 1;

        /* If we're done with a group of four mic readings, *
         * increment the sample position. */
        if (micnum == NUM_MICS)
        {
            micnum = 0;
            samplenum += 1;
```

```c
        }

        /* If we're out of samples to read, we're done with *
         * a packet! */
        if (samplenum == SAMPLE_SIZE)
        {
            return 0;
        }
    }

    printf("No␣0xff␣byte\n");
    return -1;
}

int sample_match_peaks(packet_s *pkt)
{
    sound_s sound;
    /* Vectors to store peaks and their amplitudes in */
    double peaks[NUM_XCORR][MAX_PEAKS]; /* They are stored as times in seconds here */
    int peakv[NUM_XCORR][MAX_PEAKS]; /* Amplitudes are stored here */
    int numpeaks[NUM_XCORR];
    int peak, xc;

    /* For each cross correlation */
    for (xc = 0; xc < NUM_XCORR; ++xc)
    {
        numpeaks[xc] = 0;
        peak = -1;
        while (numpeaks[xc] < MAX_PEAKS)
        {
            /* Get the next peak */
            peak = xcorr_next_peak(pkt->xcorr[xc], peak);
            if (peak == -1)
                break;

            /* Convert the peak offset to a time delta */
            double dt = peak;
            dt -= XCORR_LEN / 2;
            dt /= SAMPLE_RATE;

            /* Set the peak position and value */
            peaks[xc][numpeaks[xc]] = dt;
            peakv[xc][numpeaks[xc]] = pkt->xcorr[xc][peak];

            numpeaks[xc] += 1;
        }
    }

    /* Match the sets of peaks to sounds */
    sound_match_peaks(&sound,
        peaks[0], numpeaks[0], peakv[0],
        peaks[1], numpeaks[1], peakv[1],
        peaks[2], numpeaks[2], peakv[2]
    );
}
```

sound.h

```c
#if !defined(SOUND_H)
# define SOUND_H
# include "conf.h"
# include "sample.h"
# include "xcorr.h"
# include <math.h>
# include <stdint.h>
# include <sys/time.h>
# include <stdbool.h>

typedef struct sound sound_s;

struct sound
{
    double angle;
    double amplitude;
    double dt[NUM_XCORR];
};

/* 1 o----o 2
 * | |
 * 3 o----o 4
 *
 * |--> x
 * y v
 *
 * dt[0] is xcorr of 1 and 2,
 * dt[1] is xcorr of 1 and 3,
 * dt[2] is xcorr of 1 and 4
 */

#define SOUND_DT_X1(s) (s->dt[0])
#define SOUND_DT_X2(s) (s->dt[2] - s->dt[1])
#define SOUND_DT_Y1(s) (s->dt[1])
#define SOUND_DT_Y2(s) (s->dt[2] - s->dt[0])


/* Get the average delay of the sound in the y direction as it passes *
 * the mics. */
static inline float get_sound_dy(sound_s *sound)
{
    return (SOUND_DT_Y1(sound) + SOUND_DT_Y2(sound)) / 2.0;
}

/* Get the average delay of the sound in the x direction as it passes *
 * the mics. */
static inline float get_sound_dx(sound_s *sound)
{
    return (SOUND_DT_X1(sound) + SOUND_DT_X2(sound)) / 2.0;
}

/* Get the error in the sound. This is how far the sound deviates *
 * from the expected uniform x velocity and uniform y velocity. *
 * Large values mean either the sound is close, or that this is *
 * not a sound. */
static inline float get_sound_error(sound_s *sound)
{
    double x1, x2, xerr;
    double y1, y2, yerr;

    x1 = SOUND_DT_X1(sound);
    x2 = SOUND_DT_X2(sound);
    y1 = SOUND_DT_Y1(sound);
    y2 = SOUND_DT_Y2(sound);

    xerr = fabs((x1 - x2));
    yerr = fabs((y1 - y2));

    return xerr + yerr;
}
/* Get the angle of the sound from -pi to +pi */
static inline float get_sound_angle(sound_s *sound)
{
    return atan2(get_sound_dy(sound), get_sound_dx(sound));
}

/* Estimate the speed of the sound in m/s */
static inline float get_sound_speed(sound_s *sound)
{
    /* The distance between the pairs of mics */
    float mic_dist = 0.2;
    return mic_dist/sqrt(pow(get_sound_dx(sound), 2) + pow(get_sound_dy(sound), 2));
}

void sound_print(sound_s *sound, FILE *stream);

bool sound_verify(sound_s *sound);

bool sound_init(sound_s *sound, double dt0, double dt1, double dt2, int v );

bool sound_match_peaks(
    sound_s *sound,
    double *dt0, int ndt0, int *v0,
    double *dt1, int ndt1, int *v1,
    double *dt2, int ndt2, int *v2);

#endif
```

sound.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/select.h>
#include "sound.h"

/* Get the current time since the epoch */
static uint64_t get_time_ms()
{
    uint64_t rtn;
    struct timeval tv;

    gettimeofday(&tv, NULL);

    rtn = 1000 * tv.tv_sec;
    rtn += tv.tv_usec / 1000;

    return rtn;
}

/* Dump the JSON representing a sound to a file */
void sound_print(sound_s *sound, FILE *stream)
{
    static int id = 1;
    int nchrs = 1024;
    char buf[nchrs];
    char *ptr = &buf[0];
    char *end = &buf[nchrs];
```

```c
    uint64_t time;

    ptr += snprintf(ptr, end - ptr, "{\"id\":␣%d,␣", id++);
    ptr += snprintf(ptr, end - ptr, "\"angle\":␣%f,␣", sound->angle);
    ptr += snprintf(ptr, end - ptr, "\"amplitude\":␣%f,␣", sound->amplitude);
    ptr += snprintf(ptr, end - ptr, "\"freq\":␣null,␣");
    ptr += snprintf(ptr, end - ptr, "\"speed\":␣%f,␣", get_sound_speed(sound));
    ptr += snprintf(ptr, end - ptr, "\"error\":␣%f,␣", get_sound_error(sound));
    ptr += snprintf(ptr, end - ptr, "\"time\":␣%ld␣}\n", get_time_ms());

    if (ptr >= end)
        return;

    fwrite(buf, 1, ptr - buf, stream);
}

/* Truncate a long file down to size when it gets too long */
FILE *sound_trim_file(const char *fname)
{
    FILE *filein;
    FILE *fileout;

    const int maxsize = 4096, trimsize = 1024;

    char file[trimsize];
    struct stat status;
    char *end, *iter;

    /* If the file doesn't exist, return null */
    if (access(fname, F_OK))
        return NULL;

    /* If stat doesn't run, return null */
    if (stat(fname, &status))
        return NULL;

    /* If the file isn't long enough to truncate, return null */
    if (status.st_size <= maxsize)
        return NULL;

    /* Get the file to truncate and seek to the earliest byte *
     * that might be preserved. */
    filein = fopen(fname, "r");
    fseek(filein, status.st_size - trimsize -1, SEEK_SET);
    /* Read the remainder of the file to a buffer */
    fread(file, 1, trimsize, filein);
    fclose(filein);

    /* Iterate along the buffer until the first newline *
     * (we need to truncate along newlines, which is why *
     * we write back the final 1024 bytes, at the first *
     * newline. */
    iter = &file[0];
    end = &file[trimsize - 1];
    while (iter < end)
    {
        if (*(++iter) == '\n')
        {
            fileout = fopen(fname, "w");
            fwrite(iter + 1, 1, iter - end, fileout);
            return fileout;
        }
    }

    return NULL;
}

/* Open the file where we write sounds */
FILE *sound_get_file(void)
{
    FILE *rtn;

    rtn = sound_trim_file(CONF_SOUND);
    if (!rtn)
        return fopen(CONF_SOUND, "a");

    return rtn;
}

/* Verify whether a sound could exist. This is used to *
 * ignore sounds which aren't legitimate. */
bool sound_verify(sound_s *sound)
{
    double speed = get_sound_speed(sound);
    double error = get_sound_error(sound);

    return (error < 0.2e-3) && (speed > 300.0) && (speed < 450.0);
}

/* Initialize a sound from a set of delta times between microphones *
 * if the sound is verified, true is returned. Otherwise, false. */
bool sound_init(sound_s *sound, double dt0, double dt1, double dt2, int v)
{
    sound->dt[0] = dt0;
    sound->dt[1] = dt1;
    sound->dt[2] = dt2;

    if (!sound_verify(sound))
        return false;

    sound->angle = get_sound_angle(sound);
    sound->amplitude = v;

    return true;
}

bool sound_match_peaks(
    sound_s *sound,
    double *dt0, int ndt0, int *v0,
    double *dt1, int ndt1, int *v1,
    double *dt2, int ndt2, int *v2)
{
    int i0, i1, i2;
    FILE *f;

    f = sound_get_file();

    for (i0 = 0; i0 < ndt0; ++i0)
    for (i1 = 0; i1 < ndt1; ++i1)
    for (i2 = 0; i2 < ndt2; ++i2)
    {
        sound_s sound;
        if (sound_init(
            &sound,
            dt0[i0], dt1[i1], dt2[i2],
            v0[i0] + v1[i1] + v2[i2]))
        {
            sound_print(&sound, f);
        }
    }

    fclose(f);
}
```

xcorr.h

```
#if !defined(XCORR_H)
# define XCORR_H
# include <pthread.h>
# include "sample.h"
# include "conf.h"

#define PEAK_X_THRESHOLD 20
#define PEAK_Y_THRESHOLD 0

typedef struct xcorr_job xcorr_job_s;
typedef struct xcorr_manager xcorr_manager_s;

struct xcorr_job
{
    pthread_t thread;

    int running;
    pthread_cond_t launch;
    pthread_mutex_t launch_mtx;
    pthread_cond_t done;
    pthread_mutex_t done_mtx;

    int *a, *b;
    int *res;
};

struct xcorr_manager
{
    int running;
    pthread_t thread;
    int calibrating;
    int calibratingstarted;

    int calib[NUM_XCORR][XCORR_LEN];
    int ncalib;
    xcorr_job_s workers[NUM_XCORR];
    packet_s *packet;
};

void xcorr_manager_init(xcorr_manager_s *manager);
void xcorr_manager_kill(xcorr_manager_s *manager);
int xcorr_next_peak(int *vals, int prev);

#endif
```

xcorr.c

```
#include <string.h>
#include "fft/wrap.h"
#include "xcorr.h"
#include "sample.h"
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <math.h>

/* This is the file where our program spends most of its time *
 * it is where all the management of threads takes place */

/* The threading model is simple, three child threads are launched *
 * by one manager thread, and associated with a pair of microphones *
 * to crosscorrelate. Condition locks are used by all the threads *
 * to wait for their manager to signal a go, and then are used by *
 * the manager to wait for each thread to finish. The manager *
 * meanwhile does an FFT of the data. */
static void xcorr_job_init(xcorr_job_s *job, int *a, int *b, int *res);
static void xcorr_job_kill(xcorr_job_s *job);
static void *xcorr_job_main(void *arg);
static void xcorr_job_wait(xcorr_job_s *job);

static void *xcorr_manager_main(void *arg);

/* Wait for a job to finish */
static void xcorr_job_wait(xcorr_job_s *job)
{
    pthread_cond_wait(&(job->done), &(job->done_mtx));
}

/* Initialize a job */
static void xcorr_job_init(xcorr_job_s *job, int *a, int *b, int *res)
{
    job->a = a;
    job->b = b;
    job->res = res;
    job->running = 1;

    /* Start the pair of condition locks */
    pthread_mutex_init(&(job->launch_mtx), NULL);
    pthread_cond_init(&(job->launch), NULL);
    pthread_mutex_init(&(job->done_mtx), NULL);
    pthread_cond_init(&(job->done), NULL);

    /* Lock the done mutex before setting the thread, that way *
     * we can wait for it to send a done condition when it is *
     * initialized. */
    pthread_mutex_lock(&(job->done_mtx));

    pthread_create(&(job->thread), NULL, xcorr_job_main, job);

    xcorr_job_wait(job);
}

#define MAX(a, b) ((a > b) ? a : b)
#define MIN(a, b) ((a < b) ? a : b)

/* Normalize a sample to be zero average */
static void xcorr_norm(int *a)
{
    int ind;
    double avg;
    avg = 0.0;
    for (ind = 0; ind < SAMPLE_SIZE; ++ind)
    {
        avg += a[ind];
    }

    avg /= SAMPLE_SIZE;

    for (ind = 0; ind < SAMPLE_SIZE; ++ind)
    {
        a[ind] -= avg;
    }
}

/* Do a cross-correlation */
static void xcorr(int *a, int *b, int *res)
{
    int offset, offind, ind;

    /* To keep the cross correlation flat, we do each offset with the *
     * same number of samples. */
    for (offind = 0; offind < XCORR_LEN; ++offind)
    {
        int sum;
        sum = 0;
        offset = offind - (XCORR_LEN / 2);

        for (ind = (XCORR_LEN / 2) - offset; ind < SAMPLE_SIZE - (XCORR_LEN / 2) - offset; ++ind)
        {
            sum += a[ind] * b[ind + offset];
        }
        res[offind] = sum;
    }
}

/* Launch a job */
static void xcorr_job_launch(xcorr_job_s *job)
{
    pthread_mutex_lock(&(job->launch_mtx));
    pthread_cond_signal(&(job->launch));
    pthread_mutex_unlock(&(job->launch_mtx));
}

/* Kill a job */
static void xcorr_job_kill(xcorr_job_s *job)
{
    job->running = 0;
    /* It must be launched first, so that it isn't blocked on its condition */
    xcorr_job_launch(job);
    pthread_join(job->thread, NULL);
}

static void *xcorr_job_main(void *arg)
{
    xcorr_job_s *job;
    job = arg;

    /* Start by locking the launch mutex */
    pthread_mutex_lock(&(job->launch_mtx));

    /* Signal we are initialized. We must do this after locking launch, *
     * to avoid race conditions! */
    pthread_mutex_lock(&(job->done_mtx));
    pthread_cond_signal(&(job->done));
    pthread_mutex_unlock(&(job->done_mtx));

    while (job->running)
    {
        /* Wait to be launched */
        pthread_cond_wait(&(job->launch), &(job->launch_mtx));

        /* If we're no longer alive, die */
        if (!job->running)
            break;

        /* Do our job */
        xcorr(job->a, job->b, job->res);

        /* Signal that we are done now! */
        pthread_mutex_lock(&(job->done_mtx));
        pthread_cond_signal(&(job->done));
        pthread_mutex_unlock(&(job->done_mtx));
    }
    puts("DONE");
    return NULL;
}

/* Initialize the manager thread */
void xcorr_manager_init(xcorr_manager_s *job)
{
    job->running = 1;
    job->packet = malloc(sizeof(packet_s));

    pthread_create(&(job->thread), NULL, xcorr_manager_main, job);
}

/* Kill the manager thread */
void xcorr_manager_kill(xcorr_manager_s *job)
{
    job->running = 0;

    pthread_join(job->thread, NULL);
}

/* Calculate an fft and send it to file */
void dft_to_file(int *in)
{
    int i;
    FILE *stream;

    double reals[DFT_OUT_LEN];
    double imags[DFT_OUT_LEN];

    dft_wrap(in, reals, imags);

    stream = fopen(CONF_FFT, "w");
    fprintf(stream, "{\"fft\":␣{\n");

    for (i = 0; i < DFT_OUT_LEN; ++i)
    {
        if (i) fprintf(stream, ",\n");
        fprintf(stream, "␣␣␣␣%.2f:␣%.2f",
            (i + 1) * (DFT_MAX_FREQ/DFT_OUT_LEN),
            sqrt(reals[i] * reals[i] + imags[i] * imags[i])
        );
    }

    fprintf(stream, "\n}}\n");
    fclose(stream);
}

/* The main thread for the manager thread */
static void *xcorr_manager_main(void *arg)
{
    packet_s *pkt;
    FILE *f;
    xcorr_manager_s *job;
    xcorr_job_s *workers;
    int njob;

    job = arg;
    workers = job->workers;
    pkt = job->packet;

    /* This is set when the last xcorr was part of a calibration */
    job->calibratingstarted = 0;
    /* This is set by the main thread */
    job->calibrating = 0;

    memset(job->calib, 0, sizeof(job->calib));
    job->ncalib = 1;

    // When this is working fully, we don't need to mkfifo!
    //mkfifo("/tmp/chinchilla-serial", 0666);
    f = fopen(, "r");

    /* Initialize the xcorrelation workers */
    for (njob = 0; njob < NUM_XCORR; ++njob)
        xcorr_job_init(
            &(workers[njob]),
            pkt->data[0], pkt->data[1 + njob], pkt->xcorr[njob]
        );

    while (job->running)
    {
        int ind;

        if (sample_packet_recv(pkt, f) != 0)
        {
            usleep(100000);
            continue;
        }

        for (ind = 0; ind < NUM_MICS; ++ind)
            xcorr_norm(pkt->data[ind]);

        for (njob = 0; njob < NUM_XCORR; ++njob)
            xcorr_job_launch(&(workers[njob]));

        dft_to_file(pkt->data[0]);

        for (njob = 0; njob < NUM_XCORR; ++njob)
            xcorr_job_wait(&(workers[njob]));

        // This is the case where calibrating has just been started.
        if (job->calibrating && !job->calibratingstarted)
        {
            puts("Starting␣calibration");
            job->calibratingstarted = 1;
            job->ncalib = 0;
            memset(job->calib, 0, sizeof(job->calib));
            // This the ongoing case is run
        }

        // This is the case where calibrating is ongoing
        if (job->calibrating && job->calibratingstarted)
        {
            int xc;
            for (xc = 0; xc < NUM_XCORR; ++xc)
            {
                int ind;
                for (ind = 0; ind < XCORR_LEN; ++ind)
                {
                    job->calib[xc][ind] += pkt->xcorr[xc][ind];
                }
            }
            job->ncalib += 1;
        }
        // This is the case where calibrating has just stopped
        else if (!job->calibrating && job->calibratingstarted)
        {
            if (job->ncalib)
            {
                puts("Ending␣calibration");
                int xc;
                for (xc = 0; xc < NUM_XCORR; ++xc)
                {
                    int ind;
                    for (ind = 0; ind < XCORR_LEN; ++ind)
                    {
                        job->calib[xc][ind] /= job->ncalib;
                    }
                }
                printf("%d␣NCALIB\n", job->ncalib);
            }
            else
            {
                puts("Empty␣calibration␣:(␣I␣was␣told␣to␣calibrate␣but␣got␣no␣data");
            }
            job->calibratingstarted = 0;
            // Then the normal case is run
        }

        // This is the normal case
        if (!job->calibrating)
        {
            int xc;
            for (xc = 0; xc < NUM_XCORR; ++xc)
            {
                int ind;
                for (ind = 0; ind < XCORR_LEN; ++ind)
                {
                    printf("%d␣%d␣%d␣%d\n", xc, ind, job->calib[xc][ind],
pkt->xcorr[xc][ind]);
                    pkt->xcorr[xc][ind] -= job->calib[xc][ind];
                }
            }
            sample_match_peaks(pkt);
        }
```

```
    }

    for (njob = 0; njob < NUM_XCORR; ++njob)
        xcorr_job_kill(&(workers[njob]));

    fclose(f);

    return NULL;
}

int xcorr_next_peak(int *vals, int prev)
{
    int peak, off;
    if (prev != -1)
        peak = prev + PEAK_X_THRESHOLD;
    else
        peak = PEAK_X_THRESHOLD;

    while (peak < XCORR_LEN - PEAK_X_THRESHOLD)
    {
        // Iterate forward and see if the current peak is
        // a maximum forwards
        for (off = 0; off < PEAK_X_THRESHOLD; ++off)
        {
            if (vals[peak] < vals[peak + off])
                break;
        }

        // If there is no larger peak forwards
        if (off == PEAK_X_THRESHOLD)
        {
            // Iterate backwards and see if the peak is the maximum
            // looking backwards
            for (off = 0; off > -PEAK_X_THRESHOLD; --off)
            {
                if (vals[peak] < vals[peak + off])
                    break;
            }

            // If it is, and it is over the Y threshold, it is a peak
            if (off == -PEAK_X_THRESHOLD && vals[peak] > PEAK_Y_THRESHOLD)
                return peak;
            else
                peak += PEAK_X_THRESHOLD;
        }
        else
        {
            peak += off;
        }
    }
    return -1;
}
```

DFT.h

```
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>

using namespace std;

vector<double> c_dft_re(const vector<double> &dec_in); //compute DTF real part
```

```

vector<double> c_dft_im(const vector<double> &dec_in); //compute DTF imaginary part

vector<double> i_dft(const vector<double> &re_freq, const vector<double> &im_freq); //compute inverse DTF
```

DFT.cpp

```
#include "DFT.h"

using namespace std;
constexpr double PI = 3.14159265358979323846;

vector<double> c_dft_re(const vector<double> &dec_in) //compute Discrete Fourier Transform, real part only, by using the vector that have been passed
{
    vector<double> re_freq_temp; //temp vector

    for (int i = 0; i < dec_in.size(); i++)
    {
        re_freq_temp.push_back(0); //allocate memory
    }

    //compute Discrete Fourier Transform
    for (int i = 0; i < dec_in.size(); i++)
    {
        for (int j = 0; j < dec_in.size(); j++)
        {
            re_freq_temp[i] += dec_in[j] * cos( (2 * PI*i*j) / dec_in.size());
        }

        cout << setprecision(6) << "re_freq_temp␣" << i << "is:␣" << re_freq_temp[i] << endl; //cout for display data and checking
    }

    cout << endl;

    return re_freq_temp; //return DFT_re
}

vector<double> c_dft_im(const vector<double> &dec_in) //compute Discrete Fourier Transform, imaginary part only, by using the vector that have been passed
{
    vector<double> im_freq_temp; //temp vector

    for (int i = 0; i < dec_in.size(); i++)
    {
        im_freq_temp.push_back(0); //allocate memory
    }

    //compute Discrete Fourier Transform
    for (int i = 0; i < dec_in.size(); i++)
    {
        for (int j = 0; j < dec_in.size(); j++)
        {
            im_freq_temp[i] += -dec_in[j] * sin( (2 * PI*i*j) / dec_in.size());
        }

        cout << setprecision(6) << "im_freq_temp␣" << i << "is:␣" << im_freq_temp[i] << "j" << endl; //cout for display data and checking
    }

    cout << endl;
```

```

        return im_freq_temp; //return DFT_im
}

vector<double> i_dft(const vector<double> &re_freq,const vector<double> &im_freq) //compute inverse Discrete Fourier Transform, by using the vector of DFT_re & DFT_im that have been passed
{

        int vec_size;

        //store the biggest size of vector
        if (re_freq.size() >= im_freq.size())
        {
                vec_size = re_freq.size();
        }
        else
        {
                vec_size = im_freq.size();
        }

        //temp vector
        vector<double> re_freq_temp;
        vector<double> im_freq_temp;
        vector<double> i_dft_temp;

        re_freq_temp = re_freq;
        im_freq_temp = im_freq;

        //allocate memory
        for (int i = 0; i < vec_size; i++)
        {
                i_dft_temp.push_back(0);
        }

        //compute inverse Discrete Fourier Transform
        for (int j = 0; j < vec_size; j++)
        {
                for (int i = 0; i < vec_size; i++)
                {
                        i_dft_temp[j] += re_freq_temp[i] * cos( (2 * PI*i*j) / vec_size);
                        i_dft_temp[j] += -im_freq_temp[i] * sin( (2 * PI*i*j) / vec_size);
                }

                i_dft_temp[j] /= vec_size;

                cout << setprecision(6) << "i_dft_temp␣" << j << "␣is:␣" << i_dft_temp[j] << endl; //cout for display data and checking
        }
        cout << endl;

        return i_dft_temp; //return the result of inverse DFT
}
```

backend_code.cpp

```

#include "DFT.h"
#include "x_corr.h"
#include "dw_iface.h"

using namespace std;

```

```
int main()
{
        int sample;

        //test data
        vector<double> mic_1 = { 47, 115, 87, 128, 38, 210, 35, 127, 63, 165, 61, 255, 245,
            144, 23, 80, 50, 17, 143, 156, 198, 39, 107, 82, 223, 105, 94, 199, 84, 226 };
        vector<double> mic_2 = { 115, 87, 128, 38, 210, 35, 127, 63, 165, 61, 255, 245, 144,
            23, 80, 50, 17, 143, 156, 198, 39, 107, 82, 223, 105, 94, 199, 84, 226, 132 };
        vector<double> mic_3 = { 47, 115, 87, 128, 38, 210, 35, 127, 63, 165, 61, 255, 245,
            144, 23, 80, 50, 17, 143, 156, 198, 39, 107, 82, 223, 105, 94, 199, 84, 226 };
        vector<double> mic_4 = { 63, 165, 61, 255, 245, 144, 23, 80, 50, 17, 143, 156, 198, 39,
             107, 82, 223, 105, 94, 199, 84, 226, 27, 55, 106, 111, 210, 92, 179, 243 };

        //init vector
        vector<double> dec_str_1;
        vector<double> dec_str_2;

        vector<double> dft_str_1_re;
        vector<double> dft_str_1_im;
        vector<double> dft_str_2_re;
        vector<double> dft_str_2_im;

        vector<double> idft_str_1;
        vector<double> idft_str_2;

        vector<double> x_corr_f;
        vector<double> x_corr_s;

        //init delay
        int delay;

        //select the data to compute
        int mic_no1, mic_no2;

        mic_no1 = select_mic();

        switch (mic_no1)
        {
        case 1: dec_str_1 = mic_1;
                break;
        case 2: dec_str_1 = mic_2;
                break;
        case 3: dec_str_1 = mic_3;
                break;
        case 4: dec_str_1 = mic_4;
                break;
        }

        mic_no2 = select_mic();

        switch (mic_no2)
        {
        case 1: dec_str_2 = mic_1;
                break;
        case 2: dec_str_2 = mic_2;
                break;
        case 3: dec_str_2 = mic_3;
                break;
        case 4: dec_str_2 = mic_4;
                break;
        }
```

```cpp


        //not in used (cross_correlation using convolution)
        //x_corr_s = x_corr(dec_str_1, dec_str_2);


        //compute DFT
        dft_str_1_re = c_dft_re(dec_str_1);
        dft_str_1_im = c_dft_im(dec_str_1);

        dft_str_2_re = c_dft_re(dec_str_2);
        dft_str_2_im = c_dft_im(dec_str_2);

        if (dft_str_1_re.size() >= dft_str_1_im.size())
        {
                sample = dft_str_1_re.size();
        }
        else
        {
                sample = dft_str_1_im.size();
        }

        //cout for display data and checking
        for (int i = 0; i < sample; i++)
        {
                if (dft_str_1_im[i] < 0)
                {
                        cout << setprecision(6) << "dft_str␣" << i << "␣is:␣" << dft_str_1_re[i] << dft_str_1_im[i] << "i" << endl;
                }
                else
                {
                        cout << setprecision(6) << "dft_str␣" << i << "␣is:␣" << dft_str_1_re[i] << "+" << dft_str_1_im[i] << "i" << endl;
                }
        }

        cout << endl;

        //inverse DFT
        idft_str_1 = i_dft(dft_str_1_re, dft_str_1_im);
        idft_str_2 = i_dft(dft_str_2_re, dft_str_2_im);

        //cross_correlation (using DFT)
        x_corr_f = x_corr_dft(dec_str_1, dec_str_2);

        //calculate the power of signal in dB
        cal_amplitude(dec_str_1);

        //find delay of 2 data
        delay = delay_dft_func(x_corr_f, dec_str_1, dec_str_2);

        system("pause");
        return 0;
}
```

dw_iface.cpp

```cpp
#include "dw_iface.h"

//select which microphone data to compute
int select_mic()
{
        bool check = 0;
        int mic_no = 0;
        while (check == 0)
        {
                cout << "Please␣enter␣mic␣number␣to␣compute:␣" << endl;
                cin >> mic_no;

                //check the input is valid or not, because only 4 microphones available
                if (mic_no <= 0 || mic_no >= 5)
                {
                        cout << "error:␣selected␣number␣out␣of␣range␣(range:1␣to␣4)" << endl << endl;
                }
                else
                {
                        check = 1;
                }
        }
        return mic_no; //return selected microphone number
}

//calculate the power of signal in dB
double cal_amplitude(const vector<double> &data)
{
        vector<double> data_temp = data;

        double data_avg = 0;
        double sum_sqre = 0;
        double amplitude = 0;

        //calculate the average of data
        for (int i = 0; i < data_temp.size(); i++)
        {
                data_avg += data_temp[i];
        }

        data_avg /= data_temp.size();

        //amplitude = the sum of data[i]^2 - average of data
        for (int i = 0; i < data_temp.size(); i++)
        {
                amplitude += (pow(data_temp[i], 2) - data_avg);
        }

        //calculate in dB
        amplitude = 10 * log(amplitude);

        cout << "signal␣power(in␣dB)␣is:␣" << amplitude << endl << endl;

        return amplitude;
}
```

D.3 Web

index.php

```
<!DOCTYPE html>
<html>
    <head>
        <title>D4 UI</title>
        <link rel="stylesheet" href="style.css">
        <?php
```

```
            $dataJSON = file_get_contents("/tmp/chinchilla-fft");
            $dataArray = json_decode($dataJSON,true);
            print_r($dataArray);
            echo $dataArray[0];
        ?>
        <script src="radar.js"></script>
        <script src="log.js"></script>
        <script type="text/javascript" src="canvasjs.min.js"></script>
        <script type="text/javascript">
            window.onload = function()
            {
                var chart = new CanvasJS.Chart("chartContainer", {
                    interactivityEnabled: true,
                    title: {
                        text: "Amplitude␣Response"
                    },
                    axisX: {
                        logarithmic: true,
                        title: "Frequency␣(Hz)",
                        minimum: 1,
                        maximum: 10000
                    },
                    axisY: {
                        title: "Magnitude␣(dB)"
                    },
                    data: [
                        {
                            type: "line",
                            dataPoints: [
                                {x: 1, y: 10},
                                {x: 10, y: 1}
                            ]
                        }
                    ]
                });
                chart.render();
                var radar = new Radar(document.getElementById("ui-radar"));
                radar.init(200);

                window.setInterval(function()
                {
                    radar.blip(Math.random() * 2 * Math.PI, Math.random(), Math.random());
                }, 1000);
            }
        </script>
    </head>
    <body>
        <?php ini_set('display_errors', 'On'); error_reporting(E_ALL | E_STRICT); ?>
            <!-- <?php phpinfo();?> -->
        <?php
            if($_SERVER['REQUEST_METHOD'] == "POST" and isset($_POST['restart']))
            {
                restartPi();
            } else if($_SERVER['REQUEST_METHOD'] == "POST" and isset($_POST['calibrate']))
            {
                calibratePi();
            }
            function restartPi()
            {
                $filePath = fopen("chinchilla-reset", "w");
                //echo $filePath;
                //if(!$filePath) {echo "File Open failed";}
                //echo "Writing";
                fwrite($filePath, "reset\n");
                //echo "Closing";
                fclose($filePath);
            }
            function calibratePi() {
                $filePath = fopen("chinchilla-reset", "w");
                fwrite($filePath,"calibrate\n");
                fclose($filePath);
            }
        ?>
        <form action="upload.php" method="post" enctype="multipart/form-data">
                Select firmware to upload:
                <input type="file" name="fileToUpload" id="fileToUpload">
                <input type="submit" value="Upload␣Firmware" name="submit">
        </form>
    <form action="index.php" method="post">
        <input type="submit" name="restart" value="Restart␣Pi" />
        <input type="submit" name="calibrate" value="Calibrate␣Device" />
    </form>
    <div id="chartContainer" style="height:␣200px;␣width:␣100%;"></div>
        <div id="ui">
            <div id="ui-radar" class="radar">
            </div>
            <div id="ui-log" class="log">
                <table id="ui-log-table" class="log-table">
                    <thead>
                        <tr>
                            <th>Angle</th>
                            <th>Amplitude</th>
                            <th>Speed</th>
                            <th>Error</th>
                        </tr>
                    </thead>
                    <tbody>
                    </tbody>
                </table>
            </div>
        </div>
    </body>
</html>
```

upload.php

```
<?php
ini_set(’display_errors’, ’On’);
error_reporting(E_ALL | E_STRICT);

$target_dir = "uploads/";
$target_file = $target_dir . "firmware.zip.gpg";

$goodFile = 1;

$fileType = strtolower(pathinfo($_FILES["fileToUpload"]["name"],PATHINFO_EXTENSION));

if($_FILES["fileToUpload"]["size"] > 1000000) {
        echo nl2br("File␣too␣large,␣must␣be␣<1MB.␣\n");
        $goodFile = 0;
}

if($fileType != "gpg") {
        echo nl2br("Incorrect␣file␣type,␣please␣upload␣signed␣.zip.gpg␣only.␣\n" );
        $goodFile = 0;
}

if($goodFile == 0) {
        echo nl2br("File␣not␣uploaded.␣\n␣Redirecting...");
}else{
        if(move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],$target_file)) {
                echo nl2br("The␣file␣" . basename($_FILES["fileToUpload"]["name"]). "␣has␣been␣uploaded.␣\n␣Redirecting...");
                        //exec("fwExtract/installationScripts/install");
        echo "Opening";
        $filePath = fopen("chinchilla-reset","w");
        echo $filePath;
        //if(!$filePath) {echo "File Open failed";}
        echo "Writing";
        fwrite($filePath,"install\n");
        echo "Closing";
        fclose($filePath);

        } else {
                echo nl2br("Sorry,␣error␣uploading␣file,␣please␣try␣again.␣\n␣Redirecting...");
        }
}
header('refresh:5;␣url=index.php');
die();
?>
```

install-daemon.sh

```
#!/bin/bash

fname="/var/www/html/chinchilla-reset"
logfile="/tmp/chinchilla-log"
ctlfile="/tmp/chinchilla-backend-ctl"

[[ -p $fname ]] || mkfifo $fname
[[ -f $logfile ]] && rm $logfile

shutdown()
{
    echo SHUTTING DOWN!
    [[ -p $ctlfile ]] && echo stop > $ctlfile
    # Here, put code to stop all current processes
}

start()
{
    echo STARTING!
    sleep 10
    # Serial channel setup
    stty -F /dev/ttyACM0 406:0:18b4:8a30:3:1c:7f:15:4:2:64:0:11:13:1a:0:12:f:17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
    /var/www/backend > /tmp/chinchilla-backend-log &
    # Here, put code to start a new set of processes
}

install()
{
    echo INSTALLING!
    # Here, put code to verify an install file and install it
    lxterminal -e echo Hello!
    rm /var/www/html/uploads/firmware.zip
    echo $?
    gpg -o /var/www/html/uploads/firmware.zip -d /var/www/html/uploads/firmware.zip.gpg
    result=$?
    echo $result
    if [[ $result -eq "0" ]]; then
            echo Unzipping!
            rm /var/www/html/fwExtract/*
        unzip /var/www/html/uploads/firmware.zip -d /var/www/html/fwExtract 2>file
        echo Extracting!
        instdir=$(echo /media/pi/NODE_L432KC* | cut -d "␣" -f 1)
        echo "$instdir"
        [[ -d "$instdir" ]] && cp /var/www/html/fwExtract/*.bin "$instdir"
## cp /var/www/html/fwExtract/*.bin /media/pi/NODE_L432KC
# find /var/www/html/fwExtract -iname ’*.bin’ -exec cp {} /media/pi/NODE_L432K* \;
    else
        echo Incorrectly signed file!
    fi
}

reset()
{
    echo RESTARTING PI!
    reboot
}

calibrate()
{
    echo Initialising calibration!
    [[ -p $ctlfile ]] && echo calibrate > $ctlfile
}

sleep 5
install
sleep 10
start
while true; do
    if read -r line < $fname; then
        echo $line
        case $line in
        restart)
            shutdown
            start
            ;;
        stop)
            shutdown
            ;;
        start)
            start
            ;;
        install)
            shutdown
            install
            start
            ;;
        reset)
            shutdown
            reset
            ;;
        calibrate)
            calibrate
            ;;
        esac
    else
        echo Sleepy
        sleep 1
    fi 2>&1 >> $logfile
done
```

log.js

```
function Log(elem)
{
    this.elem = elem;
    this.body = elem.getElementsByTagName("tbody")[0];

    this.pop_row = function()
    {
        this.body.removeChild(this.body.firstChild);
    }

    this.push_row = function(data)
    {
        var row = document.createElement("tr");
        for (str of data)
        {
            var textnode = document.createTextNode(str);
            var cell = document.createElement("td");
            cell.appendChild(textnode);
            row.appendChild(cell);
        }
        this.body.appendChild(row);
    }
}
```

radar.js

```
/* Add the styles to an element to make it a circular shape */
function circle_style(elem, radius)
{
    elem.style.borderRadius = radius.toString() + "px";
    elem.style.width = (radius * 2).toString()+"px";
    elem.style.height = (radius * 2).toString()+"px";
}

/* A radar element of the UI *
 * - elem is the DOM node representing the radar element */
function Radar(elem)
{
    this.elem = elem;

    /* Initialize the radar element *
     * - radius is the radius of the radar in pixels */
    this.init = function(radius)
    {
        this.radius = radius
        circle_style(elem, radius);

        for (var angle=0; angle < Math.PI - 0.01; angle += Math.PI / 6)
        {
            this.add_radial(angle);
        }

        for (var radius = 0.1; radius < 1; radius += 0.2)
        {
            this.add_circular(radius);
        }
    }

    /* Make a blip appear on the radar *
     * - angle is the angular position of the blip in radians *
     * - radius is the distance from the origin of the blip (0.0 to 1.0) *
     * - size is the size of the blip (0.0 to 1.0) */
    this.blip = function(angle, radius, size)
    {
        var blip = new Blip(this);
        blip.init(angle, radius, size);
    }

    /* Add a radial line to the radar display *
     * - the angle of the line (rads) */
    this.add_radial = function(angle)
    {
        var elem = document.createElement("div");
        elem.className = "radial";

        elem.style.top = this.radius.toString() + "px";
        elem.style.width = (2 * this.radius.toString()) + "px";
        elem.style.transform = "rotate(" + angle.toString() + "rad)";

        this.elem.appendChild(elem);
    }

    /* Add a circular line to the radar display *
     * - the radius to add the line to (0 to 1) */
    this.add_circular = function(radius)
    {
        var elem = document.createElement("div");
        elem.className = "circular";

        circle_style(elem, radius * this.radius);
        elem.style.top = ((1 - radius) * this.radius).toString() + "px";
        elem.style.left = ((1 - radius) * this.radius).toString() + "px";

        this.elem.appendChild(elem);
    }
}

/* A blip on the radar *
 * - radar is the Radar() where we want the blip */
function Blip(radar)
{
    this.radar = radar;

    /* Initialize the blip *
     * - angle is the angular position of the blip in radians *
     * - radius is the radial position of the blip (0 to 1) *
     * - size is the size of the blip (0 to 1) */
    this.init = function(angle, radius, size)
    {
        this.elem = document.createElement("div");
        this.elem.className = "blip";
        this.radius = this.radar.radius * size / 25;

        circle_style(this.elem, this.radius);
        this.elem.style.left = this.get_xpos(angle, radius, size).toString() + "px";
        this.elem.style.top = this.get_ypos(angle, radius, size).toString() + "px";

        this.radar.elem.appendChild(this.elem);
        var self = this;

        window.setTimeout(function () { self.fade() }, 1000);
        window.setTimeout(function () { self.kill() }, 4000);

        this.elem.style.display = "block";
    }

    /* Get the offset of the blip from its parent element in pixels */
    this.get_xpos = function(angle, radius, size)
    {
        var centre = Math.cos(angle) * radius * this.radar.radius;
        return (centre - this.radius) + this.radar.radius;
    }

    /* Get the offset of the blip from its parent element in pixels */
    this.get_ypos = function(angle, radius, size)
    {
        var centre = - Math.sin(angle) * radius * this.radar.radius;
        return (centre - this.radius) + this.radar.radius;
    }

    /* Cause this blip to start fading away to nothing */
    this.fade = function()
    {
        this.elem.style.transform = "scale(0)";
    }

    /* Cause this blip to stop existing */
    this.kill = function()
    {
        this.radar.elem.removeChild(this.elem);
    }
}
```

requestor.js

```
function Requestor()
{
    this.radar = new Radar(document.getElementById("ui-radar"));
    this.log = new Log(document.getElementById("ui-log"));
    this.lastid = 0;

    this.add_row = function(data)
    {
        if (data.id <= this.lastid)
        {
            return;
        }

        this.lastid = data.id;
        this.push_row([data.angle, data.amplitude, data.speed, data.error])
    }

    this.add_blip = function(data)
    {
        this.radar.blip(data.angle, 0.5, data.amplitude);
    }

    this.on_sounds = function()
    {
        var text = this.req.responseText;
        var lines = this.text.split("\n");

        for (line of lines)
        {
            var data = JSON.parse(line);
            this.add_row(data);
            this.add_blip(data);
        }
    }

    this.request_sounds = function()
    {
        this.req = new XMLHttpRequest();
        this.req.open("GET", "/chinchilla-sounds");
        this.req.onreadystatechange = this.on_sounds;
    }
}
```

style.css

```
.radar {
    background-color: #335;
    position: relative;
}

.blip {
    background-color: #ff7;
    transition: transform 2s;
    position: absolute;
    display: none;
}

/* The radial lines on the radar */
.radar > .radial {
    background-color: #fff;
    height: 1px;
    position: absolute;
}

/* The circular lines on the radar */
.radar > .circular {
    border: 1px solid #fff;
    position: absolute;
    background-color: #335;
}

/* The first circle has a solid background to block the middle, but the *
 * others are all transparent */
.radar > .circular ~ .circular { background-color: transparent; }

.log {
        display: inline-block;
        overflow: auto;
        height: 400px;
}

.log-table th {
        padding: 10px;
        background-color: #335;
        color: white;
        font-style: bold;
}

.log-table td {
        background-color: #559;
        color: white;
        padding: 2px;
        border: 1px solid #335;
}

.log-row {
}
```

D.4 LED Control

led_ctl.py

```python
# Based on example by Adafruit and using Adafruit libraries

# imports
import time
import board
import neopixel
import json
import digitalio
import ast
from math import pi

# Definitions
FAN_OUT = 3
NUM_PIXELS = 46

# _____________________________________________________________________________
# Setup
#
# Setup LEDs for Adafruit library
# Setup button and declare as pull up input
# Setup memory for remembering id, time and button state between loops
# _____________________________________________________________________________

# LED setup
pixel_pin = board.D18
ORDER = neopixel.GRB
pixels = neopixel.NeoPixel(pixel_pin, NUM_PIXELS, brightness=1,
                           auto_write=False, pixel_order=ORDER)

# Button setup
button = digitalio.DigitalInOut(board.D23)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

# Declare memory between angles
last_id = 0
last_ms = 0

# _____________________________________________________________________________
# Loading animation on LEDs
# _____________________________________________________________________________
def load_screen():

    # Clear all LEDs
    for n in range(0,NUM_PIXELS):
        pixels[n] = ((0, 0, 0))
    pixels.show()

    # Make first and last LED white
    pixels[0] = ((255, 255, 255))
    pixels[NUM_PIXELS - 1] = ((255, 255, 255))

    # Display LED values calculated
    pixels.show()

    # Send white LED lit up round ring
    for n in range(1, int(NUM_PIXELS / 2)):
        time.sleep(0.1)

        pixels[n - 1] = ((0, 0, 0))
        pixels[n] = ((255, 255, 255))

        pixels[NUM_PIXELS - n] = ((0, 0, 0))
        pixels[NUM_PIXELS -1 - n] = ((255, 255, 255))

        # Display LED values calculated
        pixels.show()

    # Clear all LEDs
    for n in range(0,NUM_PIXELS):
        pixels[n] = ((0, 0, 0))
    pixels.show()

    # Light up all LEDs in sequence with colours that range from orange to purple
    for n in range(int(NUM_PIXELS / 2) - 1, -1, -1):
        time.sleep(0.1)
        pixels[n] = ((round(n * 255 / (int(NUM_PIXELS / 2) - 1)), n, round(255 - (n * 255 / (int(NUM_PIXELS / 2) - 1))) ))
        pixels[NUM_PIXELS -1 - n] = ((round(n * 255 / (int(NUM_PIXELS / 2) - 1)), n, round(255 - (n * 255 / (int(NUM_PIXELS / 2) - 1))) ))

        # Display LED values calculated
        pixels.show()

    time.sleep(1)

    # Clear all LEDs
    for n in range(0,NUM_PIXELS):
        pixels[n] = ((0, 0, 0))
    pixels.show()

# -----------------------------------------------------------------------------
# Calibration animation
# -----------------------------------------------------------------------------
def calibrate():
    # Clear all LEDs
    for n in range(0,NUM_PIXELS):
        pixels[n] = ((0, 0, 0))
    pixels.show()

    # Make first LED white
    pixels[0] = ((255, 255, 255))

    # Display LED values calculated
    pixels.show()

    # Send white LED lit up round ring
    for n in range(1, NUM_PIXELS):
        time.sleep(0.1)

        pixels[n - 1] = ((0, 0, 0))
        pixels[n] = ((255, 255, 255))

        # Display LED values calculated
        pixels.show()

    # Clear all LEDs
    for n in range(0,NUM_PIXELS):
        pixels[n] = ((0, 0, 0))
    pixels.show()

# ---------------------------------------------------------------------------
# Sign function
#
# This outputs only the sign of a number ignoring the value
# Possible outputs are 1 and -1
# ---------------------------------------------------------------------------
def sign(num):
    if num <= 0:
        return -1
    else:
        return 1

# ---------------------------------------------------------------------------
# LED ring code
#
# Calculates LED RGB values based on angle and amplitude
# Fades out as it goes round the ring up to the fanout value
# ---------------------------------------------------------------------------
def led_ring(angle, amplitude, freq):

    # Translate angle to LED number
    ring_pos = round((angle * NUM_PIXELS) / (2 * pi))

    # Offset from Led number found above
    ring_offset = ((angle * NUM_PIXELS) / (2 * pi)) - ring_pos

    # For loop stepping through FAN_OUT
    for n in range(1 - FAN_OUT, 1 + FAN_OUT):

        # Calculate LED index based on angle
        index = ring_pos + (n * sign(ring_offset)) + (NUM_PIXELS *
                sign(-ring_pos - (n * sign(ring_offset))))

        # Extract current value of LED in question
        pixel = list(pixels[index])

        # Calculate RGB values for LEDs incorporating:
        # past value, amplitude and angle

        pixels[index] = (( max(0, min(255, pixel[0] + round(amplitude * 255 *
                         max(0, ((freq / 500) - 1)) * (FAN_OUT - abs(n) +
                         (abs(ring_offset) * sign(n)) ) / FAN_OUT ))),

                         max(0, min(255, pixel[1] + round(amplitude * 255 *
                         (1 - abs((freq / 500) - 1)) * (FAN_OUT - abs(n) +
                         (abs(ring_offset) * sign(n)) ) / FAN_OUT ))),

                         max(0, min(255, pixel[2] + round(amplitude * 255 *
                         max(0, ((-freq / 500) + 1)) * (FAN_OUT - abs(n) +
                         (abs(ring_offset) * sign(n)) ) / FAN_OUT )) )))

    # Display LED values calculated
    pixels.show()
```

```

# ------------------------------------------------------------------------------
# Main Code
#
# Call setup
# Imports angle and amplitude value from json file, and triggers LED ring code
# Fades the LED values by 1/3 each loop
# Reads button value
# Sends a calibrate command if short button press
# Sends a reset command if long button press
# ------------------------------------------------------------------------------

load_screen()

while True:

    # Fade all LEDs out by 1/3 each time
    for i in range(0, NUM_PIXELS):

        # Convert from tuple to list for current LED
        pixel = list(pixels[i])
        if pixel[0] > 0:
            pixel[0] -= max(1, round(pixel[0]/3))
        if pixel[1] > 0:
            pixel[1] -= max(1, round(pixel[1]/3))
        if pixel[2] > 0:
            pixel[2] -= max(1, round(pixel[2]/3))

        # Convert back from list to tuple for current LED
        pixels[i] = tuple(pixel)

        # Display LED values calculated
        pixels.show()

    # Open Json file and read id, angle and amplitude
    with open('/tmp/chinchilla-sounds', 'r') as json_file:
        for line in json_file.readlines():
            object = json.loads(line)
            id = object['id']
            angle = object['angle']
            amplitude = object['amplitude'] / 2000000.0

    # Import FFT data
    try:
        fft = ast.literal_eval(open('/tmp/chinchilla-fft', 'r').read())
    except:
        fft = None

    # Check for frequency with largest amplitude and use that
    if fft != None:
        largest_amp = 0
        for currfreq, ampl in fft['fft'].items():
            if ampl > largest_amp:
                freq = currfreq
                largest_amp = ampl

    # Or default frequency of 500
    else:
        freq = 500

    # Check id has increased (don't repeat same sound)
    if id > last_id:
```

```

    # Call LED ring code
    led_ring(angle, amplitude, freq)

    # Update last id memory
    last_id = id

  # Get current time in milliseconds
  now_ms = int(time.time() * 1000)

  # Get current button value
  button_value = not button.value

  # Open file, write command and close if button press is greater than 50ms (debounce)
  f = open('/tmp/backend-ctl', 'w')
  if (not button_value):
    last_ms = now_ms
  if (button_value and now_ms > last_ms + 50):
    f.write('calibrate')
  f.close()

  # Trigger calibration animation
  if (button_value and now_ms > last_ms + 50):
    calibrate()
```

E Project Meeting Agenda and Minutes

E.1 15:00 16/02/2019

Attendees: Francis, Hugo, Mark, Matt J, Tom
Place: Level 3 Zepler labs

- Met and discussed possible project options
- Ruled out basic sensors as those would be too simple
- Decided project would have a web UI
- Deciding between il matto and display, or LED ring to give output
- Discussed power requirements and amplifiers
- Decided it would detect sound using four microphones
- Decided it would have a 3D printed case
- Would use microcontroller for data collection
- Discussed Raspberry Pi or FPGA for data collection
- Split code into five block sections
- Split up project into block sections
- Made GitHub for code and docs

E.2 10:00 18/02/2019

Attendees: all
Place: Level 2 Zepler Labs

- Discussed idea from last meeting with everyone
- Confirmed everybody was OK with basic idea
- Specified details for requirements

- Detailed processing methods
- Given everybody access to GitHub and explained usage
- Assigned individual leads for each block

E.3 11:50 19/02/2019

Attendees: All + Tim Forcer
Place: Level 2 Zepler Labs

- Discussed project idea and specific details
- Were advised on the importance of specifying interfaces
- Were advised that we should make test data so that each block can be tested independently of each other
- Told about the format of the project and advised about individual roles

E.4 15:00 19/02/2019

Attendees: All
Place: Level 3 Zepler Labs

- Discussed plan in detail in response to Tim Forcer's comments
- Established schematics for some components
- Confirmed which parts are needed
- Found parts from RS Components and Onecall
- Created parts list

E.5 15:00 21/02/2019

Attendees: Hugo, Mark, Will
Place: Level 3 Zepler Labs

- Discussed design summary
- Discussed individual roles
- Discussed individual module specifications
- Discussed costing
- Discussed construction methods
- Discussed risk assessment
- Discussed project proposal form