Frequency-based event detection

- Topic: Embedded Realtime signal processing -

Project Report
ED6-6-F2020

Aalborg University
Electronics and Computer Engineering

Copyright © Aalborg University 2019

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Frequency-based event detection

Theme:
Embedded real-time signal processing

Project Period:
November 2019 - March 2020

Project Group:
ED6-6-F2020

Participant(s):
Paulius Riauka

Supervisor(s):
Daniel Ortiz-Arroyo

Copies: 1

Page Numbers: 45

Date of Completion:
August 4, 2020

Abstract:

The report consists of analysis and implementation of a frequency-based clap detector. The project testing and development was performed using two setups:

- microphone from laptop Dell3543 and MATLAB
- visual studio code and Atmel SAM3X8E embedded in Arduino Due

The first chapter analyses a problem similar to the one this project is trying to solve and raises a hypothesis. The second chapter analyses the solution for the problem. The third chapter implements the solution and presents its properties The fourth chapter implements the previously shown solution in an embedded environment. The fifth chapter reviews the progress and discusses the perspectives.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

Preface

Paulius Riauka
<priauk16@student.aau.dk>

Chapter 1

Introduction

1.1 Frequency based event detection

Frequency based event detection is a widely applied tool that has been used for centuries. The use of it spread even more after the start digital computation era.
At the center of frequency based event detection lies the Fourier transform: a method to convert time series of samples into series of frequency. This method has given us tools to analyse the phase and magnitude of a signal's frequency. Observing these new properties has enabled us to develop new tools as well improve the old ones. A great example of a well-improved toolis a seismoscope. Back then seismoscopes were dependent on amplitude of the signal far more than the frequency while modern seismometers can make great calculated guess with regards to what type of signal was detected by the frequency of it on top of amplitude analysis [2]. Furthermore, frequency based event detection plays an important role in many innovative technology fields: it's a part of what helps neural networks comprehend the human speech. It's also the main tool for neural networks to extract information about various disorders as well as emotions from our brain using electroencephalography. The field of application for frequency based event detection will most likely keep expanding to fields such as power grid disturbance localisation [8] etc.

1.2 Problem analysis

For the last three decades multiple governments situated in zones of high seismic activity have invested in early earthquake warning systems. These systems can locate the earthquake's epicenter, find its intensity and inform the local communities on average about one minute before the earthquake.

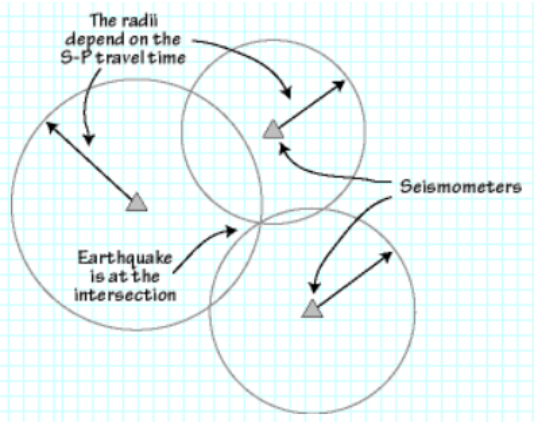

Figure 1.1: A common method to locate earthquake's epicenter [1]

The reason these systems are able to forecast these events are pressure waves also known as primary waves (P-waves). P-waves travel at speeds varying between 1 and 14km/s depending on the soil. They are faster than the surface waves (S-waves), that travel at speeds varying from 1 to 8km/s, and sometimes can be felt, but in most cases vibrations of P-waves come through unnoticed by our senses.
As normally earthquake epicenters are located between the earth's surface and depths up to 800km, in many cases the alert reaches the target population within less than a minute until the arrival of S-wave [1]. Since the time given to prepare is so short, communities that have access to this technology often perform alarm tests.
The accuracy rate of these systems was initially quite inconsistent: it could not recognize P-waves from microearthquakes (earthquakes that have a magnitude of less than 2 on Richter scale) nor were they programmed to register multiple earthquakes at a time. Nowadays these problems are recognized and solved, what has laid a foundation for a consistently highly accurate system. For example, in Japan, the accuracy rate of these systems has increased from 75% in 2007 (due to aftershocks in 2011 the accuracy for fiscal year dropped to 26%) to 85% by 2015.
The problem of microearthquake intervention was solved by ignoring the waves of certain frequencies knowing that microearthquake caused seismograph oscillations have higher frequency than P-waves (figure 1.2).

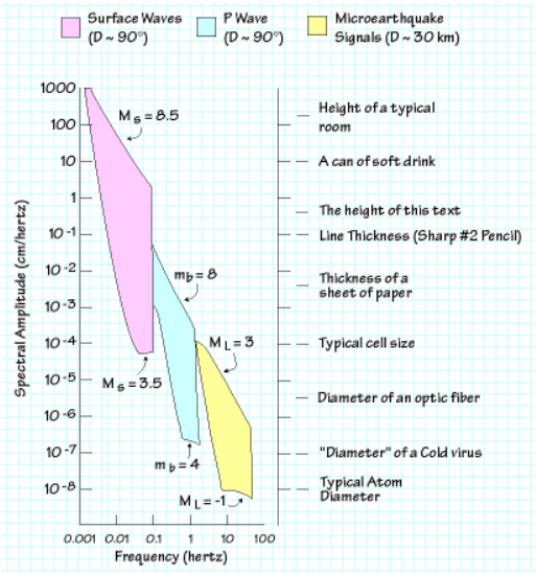

Figure 1.2: Ranges of earthquake signals. Letter D represents distance. Large distances over earth's surface are represented in degrees. 1 degree = 111.19km [2]

1.3 Clap triggers available on the market

Clap based triggers became a trend in 1980's. It became a part of American pop culture due to high level of convenience it introduced in people's daily lives: these switches, connected to wall outlets, create the ability to toggle any electric outlet-powered device in the house without using any additional tools. Furthermore, some of them are programmable: different number of claps can toggle different devices. Unfortunately, these devices had a massive downside: they could be false triggered by not only loud speech and music, but also animal induced noises, steps etc.. In case of people's absence, a turned-on light is an inconvenience. However a blow dryer or other heat inducing devices could become a fire hazard.
The cause of this issue is the working principle of clappers: these devices are triggered, when the amplitude of a signal obtained through a built-in microphone overcomes a threshold. The patented clapper device was programmed to search for a patterns of impulses to increase the accuracy rate [7].
Nowadays the clappers that were made back then are still available on the market. The market oriented around sound induced triggers has turned towards voice controlled switches that surpass the clappers in functionality yet majority of them are at best as good regarding trigger success rate. Voice of a casually speaking person does not normally exceed 7kHz while a clap sound occupies a far wider frequency range. Having this in mind a hypothesis can be made that a clap event could be recognized from event of speech by looking at the performing Fourier analysis and comparing the amplitude spectres of the two signals' frequencies. Therefore, im-

plementation of frequency based event detection should increase the accuracy rate of a clapper trigger device.

1.4 Project delimitation

This project focuses on pattern / amplitude detection in Fourier transform of the signal. It applies the methods used in fields such as seismology for a simpler purpose:to detect a hand clap among other sounds, such as speech. Although the focus of this project is event detection based on frequency, changes in amplitude will not be ignored: it will be used as a trigger for input detection.

Project report consists of documentation regarding MATLAB implementation, design of the hardware system and implementation of the system using a microcontroller towards developing a clap trigger.

1.5 Tools

The following list contains software tools used for development of this project:

- Arduino IDE
- Matlab R2018a
- KiCad
- Visual Studio code

Chapter 2

Fourier transform

2.1 Fourier analysis

2.1.1 Fourier transform

In order to have a full understanding and control of the signal processing performed by the processor, it was decided to develop a Fourier transform function from scratch instead of using a dedicated library.
Fourier transform is a method to decompose a time-based pattern into a pattern of frequencies that it consists of. Continuous Fourier transform is represented by formula 2.1

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2i\pi ft} dt \tag{2.1}$$

The discrete equivalent of it is represented as formula 2.2

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-2i\pi kn/N} \tag{2.2}$$

In the equation above, X(k) represents amount of frequency k is the signal, N - number of samples, n - sample number (0 to N-1), x(n) - nth sample, k- frequency (0 to N-1 Hz). Exponent in equation 2.2 can be be rewritten to 2.3 using Euler's formula (equation 2.4).

$$X(k) = \sum_{n=0}^{N-1} x(n)(cos(-2\pi kn/N) + isin(-2\pi kn/N)) \tag{2.3}$$

$$e^{xi} = cos(x) + i * sin(x) \tag{2.4}$$

In order to implement this equation in MATLAB, k values are placed in a row vector while n values are placed in a column vector as represented in equations 2.5

and 2.6.

$$n = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ N-1 \end{pmatrix} \tag{2.5}$$

$$k = \begin{pmatrix} 0 & 1 & \cdots & N-1 \end{pmatrix} \tag{2.6}$$

This way exponent is converted into a NxN matrix of values:

$$A_{n,k} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{0,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,1} & \cdots & a_{N-1,N-1} \end{pmatrix} \tag{2.7}$$

where

$$a_{n,k} = cos(-2\pi kn/N) + i * sin(-2\pi kn/N) \tag{2.8}$$

Then

$$X(k) = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix} * A_{n,k} \tag{2.9}$$

However, this method is considered slow due to multiplication of matrices containing complex values with fractions that have many values after past decimal point.

2.1.2 Fast Fourier transform

Fast Fourier transform tackles the issue of long computation time. One of the most popular methods of fast Fourier transform implementation is Cooley-Tukey algorithm. by reducing the size of matrices that are used for multiplication. This is achieved using recursive subdivision. and introducing a new variable:

$$W_N^k = e^{-2i\pi k/N} \tag{2.10}$$

Then the algorithm can be represented using a butterfly diagram:

From the illustration above, two equations can be derived:

$$X(0) = x(0) + W_2^0 * x(1) \tag{2.11}$$

$$X(1) = x(1) + W_2^1 * x(0) \tag{2.12}$$

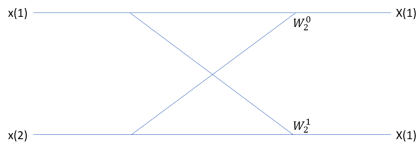

Figure 2.1: Butterfly diagram of Fast Fourier transform for signal consisting of 2 samples using Cooley-Tukey algorithm. Source: [5]

What is more, the second half of array of W is always identical to the first half, but negative (the case for array of two values is expressed in equation 2.13)

$$W_2^0 = -W_2^1 \tag{2.13}$$

Given a longer signal, simplifies the operations by dividing the signal into two even arrays until there are only two values in an array. The a discrete Fourier transform is performed with the two values and is taken back into the previous array it was derived from to repeat the action. Butterfly diagram of the entire algorithm can be found in figure 2.2

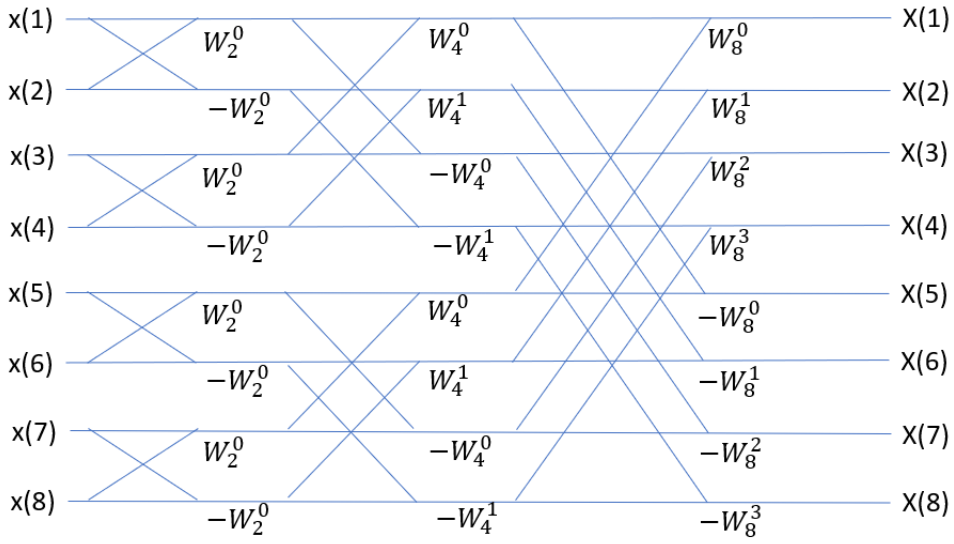

Figure 2.2: Butterfly diagram of Fast Fourier transform for signal consisting of 8 samples using Cooley-Tukey algorithm. Source: [5]

However, this algorithm demands a signal length to be a power of two. This problem can be solved by either zero-padding the input or rounding it up to a suitable value.In the case for this project, in MATLAB implementation section the sample length was set to 1/20th of a second for a signal sampled at 44100Hz. This is equal to 2205 samples, however as the second from the top illustration in figure

3.3 indicates, the signal seems to have settled after around 2000 samples, therefore it is decided to round the signal length to be 2048.

2.2 Output formatting

The Fourier transform is not comfortable to interpret in the form that it is obtained at initially (figure 2.3).

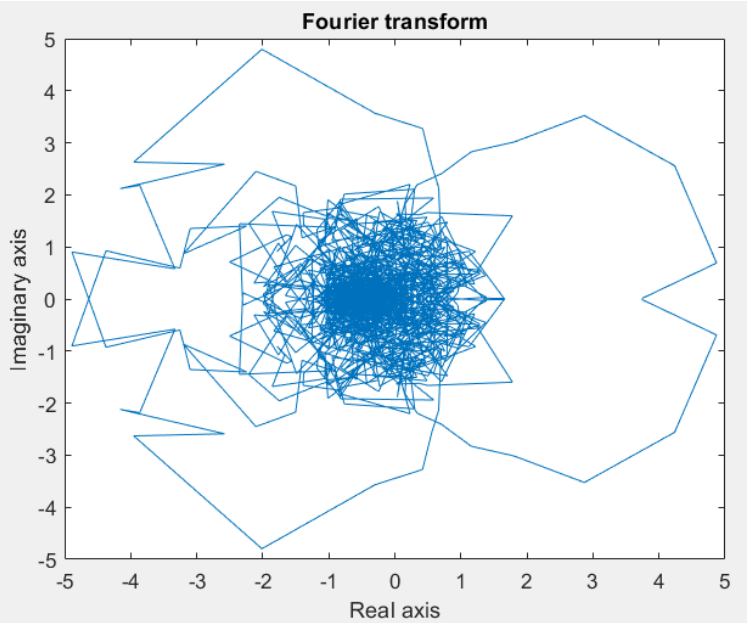

Figure 2.3: Fourier transform of a clap signal

Therefore it is aimed to create a plot that would indicate which frequencies dominate the signal. This can be achieved by plotting amplitude spectrum. Amplitude spectrum can be obtained by plotting absolute value of signal's Fourier transform divided by number of samples (figure 2.4) 2.3).

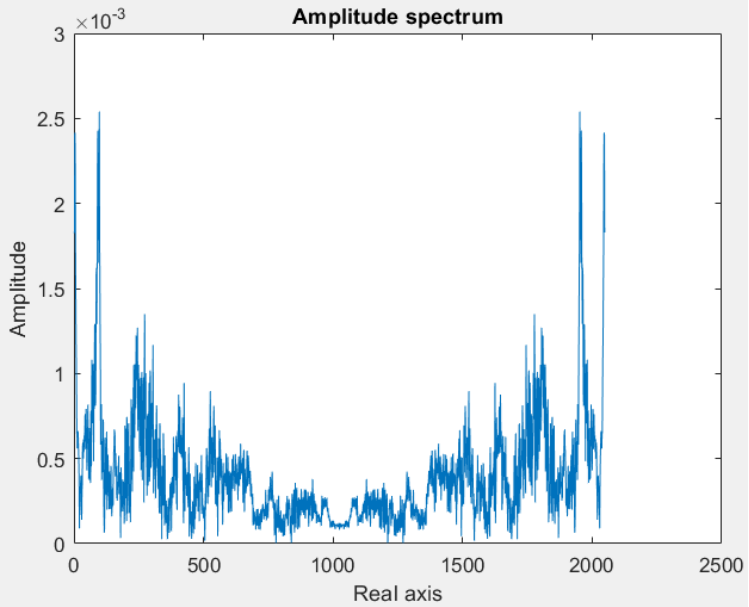

Figure 2.4: Amplitude spectrum of a clap signal

Since a complex Fourier transform formula is used for to perform calculations, the second half of the output mirrors the first half. Because of this, the obtained array has to be halved and multiplied by two in order to add the second half, then expanded over half of the sampling frequency. Using this method, a plot of amplitude over frequency is obtained. To convert amplitude units to decibels, logarithm of amplitude with base 10 was multiplied by 20 and used as vertical axis (figure2.6). The information for this chapter was sourced from the following literature: [5] and [6]

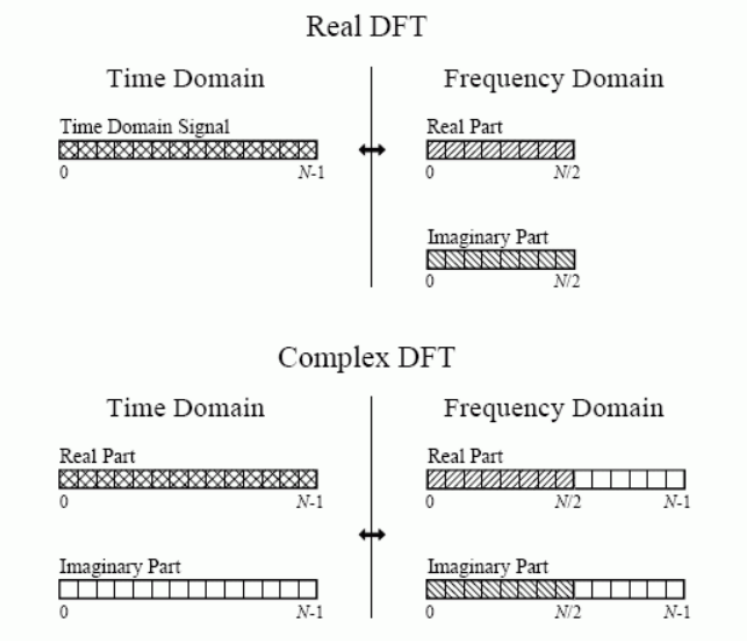

Figure 2.5: Crosshatched regions indicate the values common on the two transforms. Source:[6]

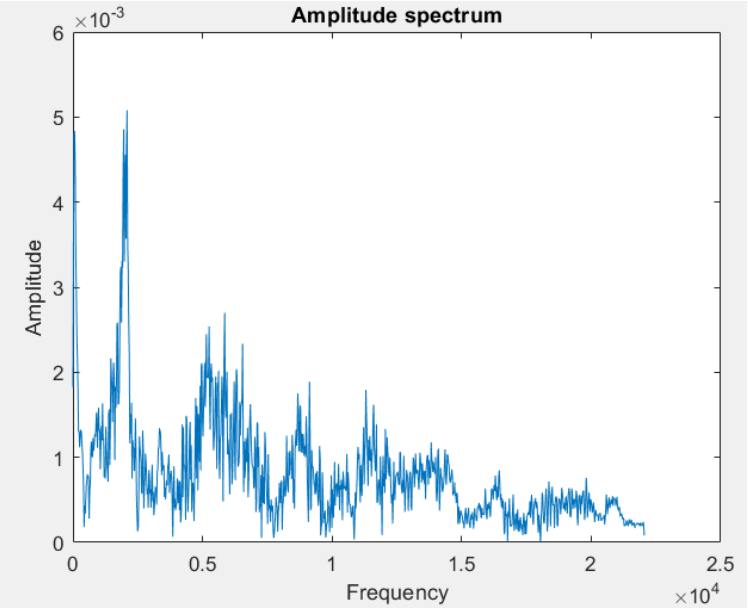

Figure 2.6: Plot of amplitude over frequency

Chapter 3

MATLAB implementation

3.1 Setup

The purpose of MATLAB implementation is to clarify how this system is expected to work as well as obtain proof of concept. For this part of the project a computer microphone is used. A testbench in MATLAB is then built to easily create and test various inputs (See appendix A).

3.1.1 Setup test

Firstly, in order to test the setup and make sure the computer's sound card supports audio sampling rate of 44100Hz, a phone was used to create a test with a 20kHz audio signal. The result (figure 3.1) visible in the produced FFT was exactly as expected.

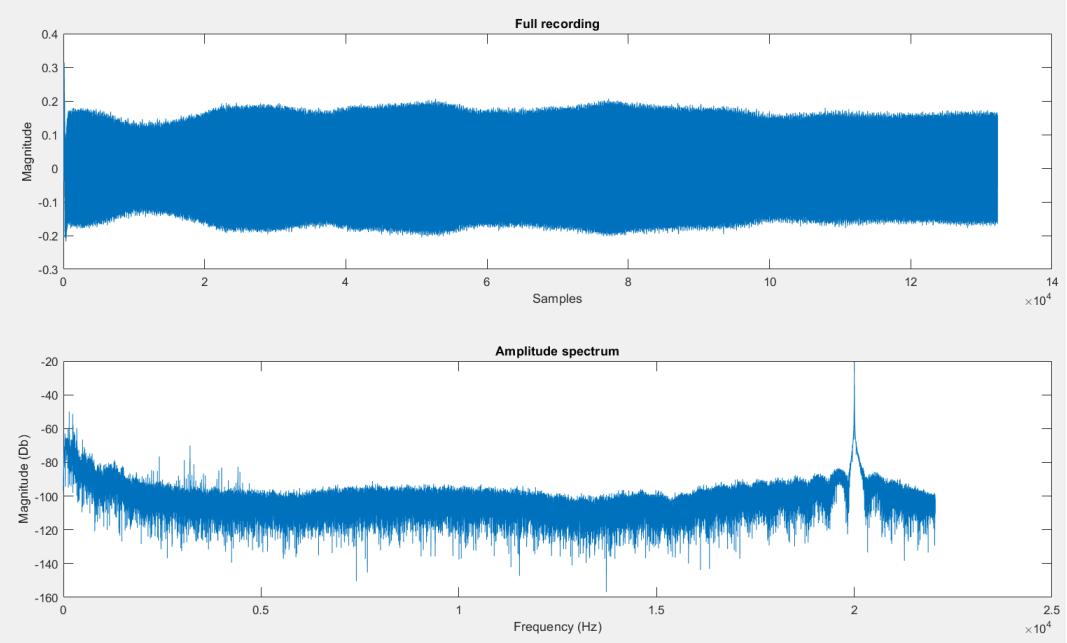

Figure 3.1: Top graph: 3 second long 20kHz audio input signal
Bottom graph: Signal's amplitude spectrum obtained through Fourier transform

3.2 Clap identification

Initially, to see the difference between a Fourier transform of a clap and speech audio input, 10 second samples of each were made (figure 3.2). However, this comparison has left a lot of space for interpretation: samples come in different peak amplitudes, clap resembles more of an impulse while speech input resembles continuous input. Due to this difference majority of the hand clap signal sample's length is occupied by nothing more than background noise, while most of the speech signal is dominated by speech input.

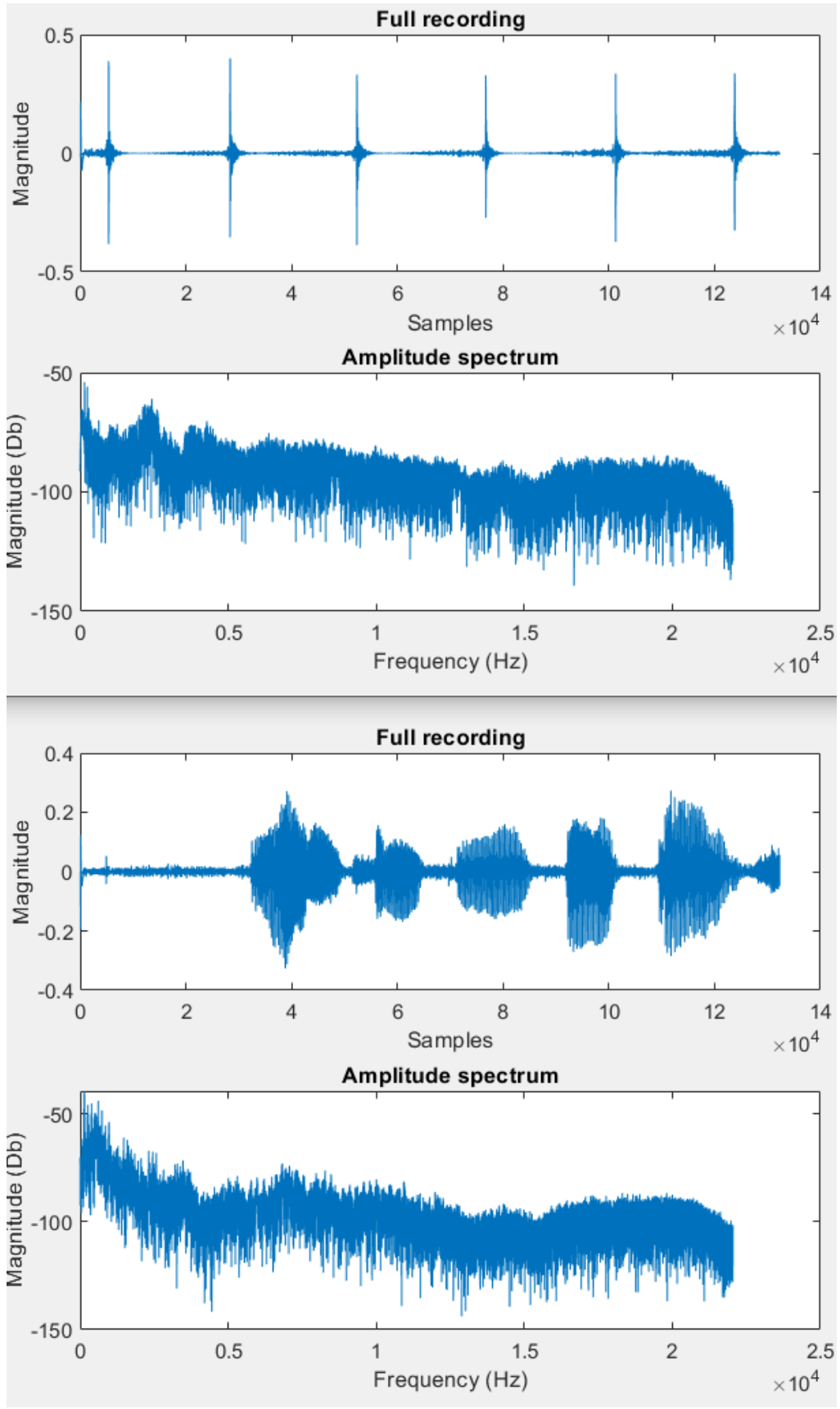

Figure 3.2: Comparison between clap and speech audio samples described top to bottom:
3 second audio sample with six claps;
Clap audio sample's amplitude spectrum;
3 second audio sample of speech;
Speech audio sample's amplitude spectrum;

Another array was introduced in order to record a shorter sample so that the wanted signal would occupy majority of it. When the input signal exceeds a certain threshold, a short recording is created. It was found that a hand clap oscillates for about 50ms. Knowing that input range is form -1 to 1, trigger threshold was set to absolute value of 0.2. The results can be found in figure 3.3

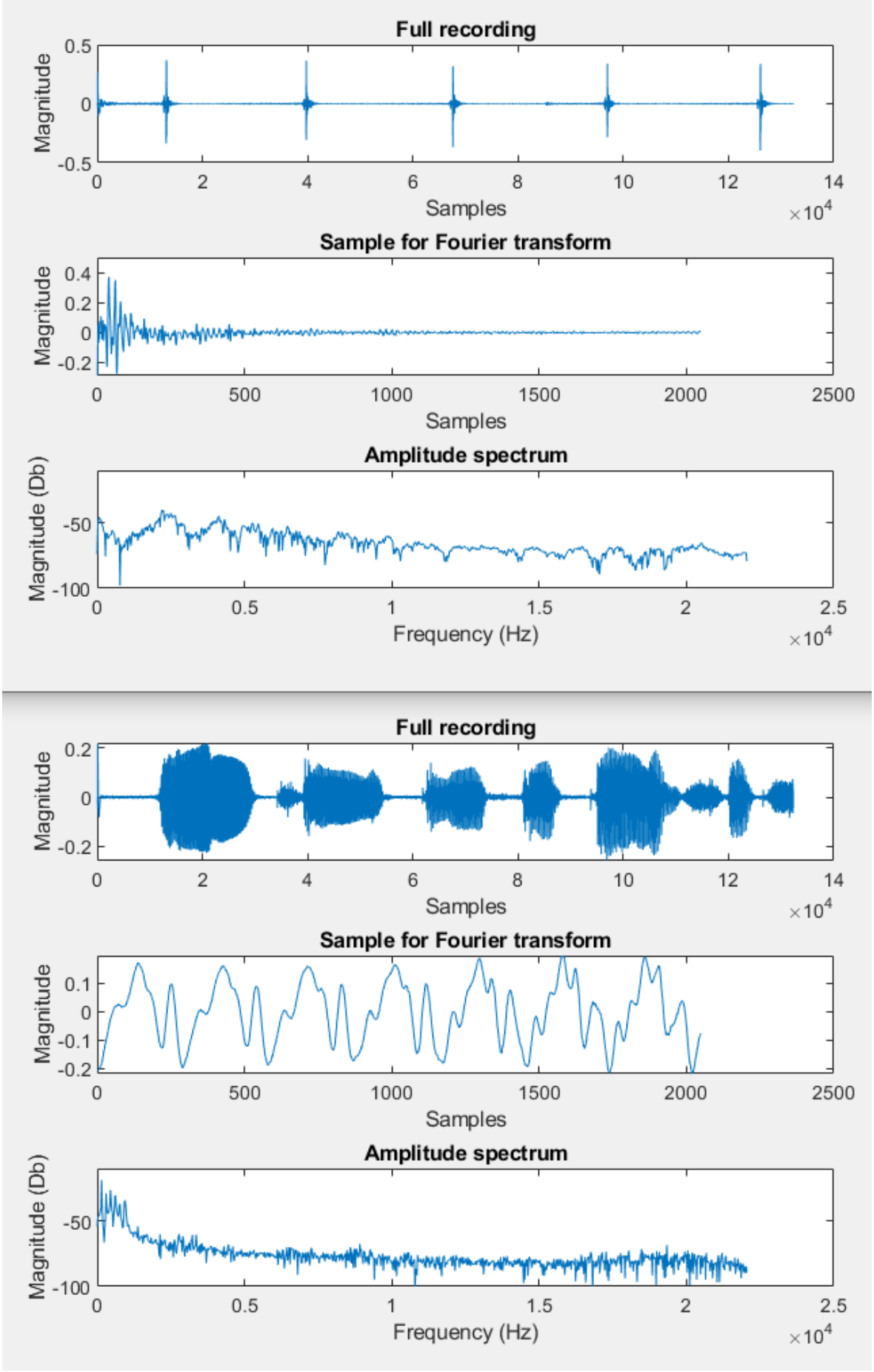

Figure 3.3: Comparison between clap and speech audio samples:
Top row represents full 3 second samples
Middle row represents the 50ms samples taken for FFT creation as the threshold was passed

From the Fourier transform plot it is clear that the hand clap signal's frequency spectrum is quite different: initially, within the range between 0 and about 7kHz the amplitude of speech signal's Fourier transform gradually decreases while the hand clap signal's transform is somewhat chaotic. However, after 10 kHz the hand clap signal's average FFT value is quite clearly higher than one of speech signal's. Adding a sample of silence into comparison (figure 3.4 was obtained by taking a sample of values from Data(40000) to Data 42048)) confirms that looking for a high average value in the region between 10 and 20 kHz should increase a chance of successfully identifying a handclap among other sounds.

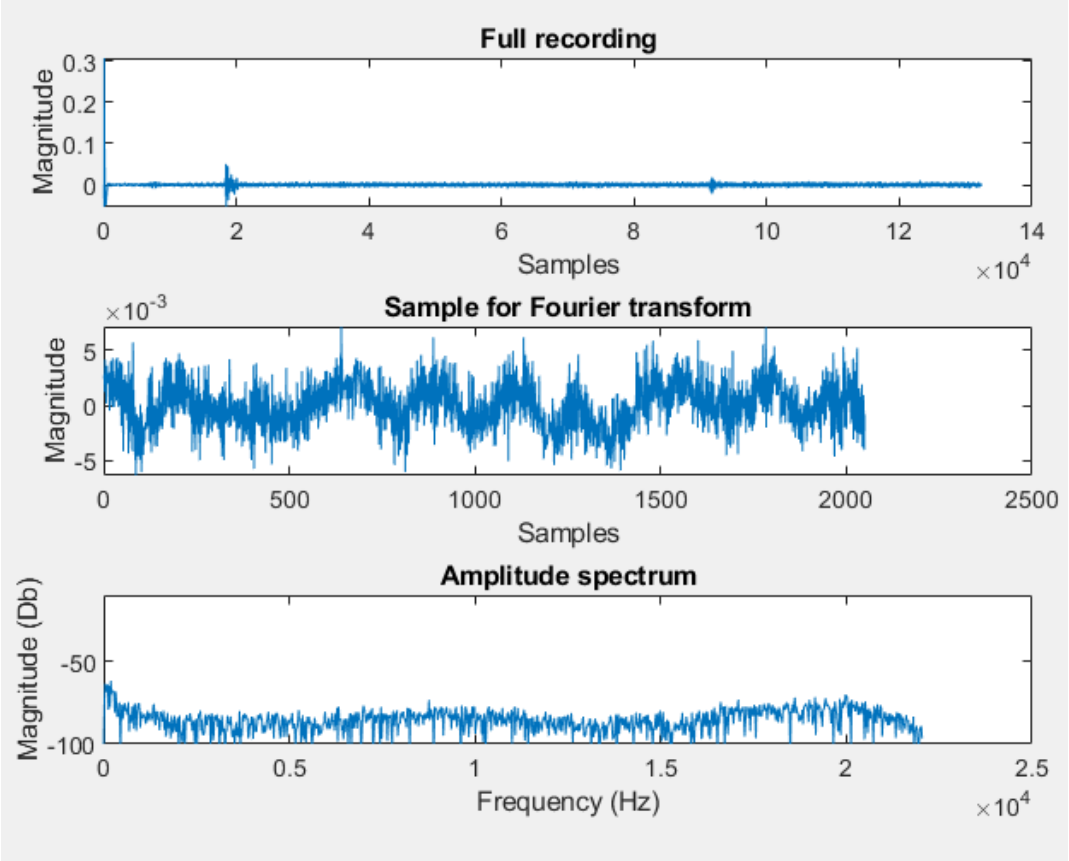

Figure 3.4: Audio sample of silence (background noise)

3.3 Frequency based clap detector

Having a proof that the hypothesis made in the problem analysis chapter is correct, a frequency based clap event detector was built. The audio sample could be adopted from the previous testbench provided in Appendix A. The threshold value and sample length were used from there as well. The pseudo-code for the program can be found here. It also includes function for Fast Fourier transform built by following Cooley-Tukey algorithm described in Fourier analysis chapter,

```
array data; //imported from previous testbench
array Sample;
array FFT;
array Spectrum;
array Half_Spectrum;
int Clap_count=0;
int i=0;
int a=0;
int Sample_length=2048;
int High_f_average=0;
int Low_f_average=0;
int threshold=0.2;
int frequency_threshold;
int first half; //Sample value at which the the frequency
//spectrum is split
function perform_FFT


while(i < length[data]{ //check all samples
    if (abs(data[i])>threshold){// if threshold was reached
        High_f_average=0; //reset values
        Low_f_average=0;
        while (a<length(Sample)){//record a new sample for FFT
            Sample[a]=Data[i];
            a=a+1;
            i=i+1;
            }
        FFT=Perform_FFT(Sample);
        Spectrum=abs(FFT/length(Sample);
        for (int i=0, i<length(Spectrum)+1, i+1){//calculate magnitude
        //spectrum
            Half_Spectrum[i]=20*log10(Spectrum[i]*2);
            if(i<first_half){ //store values
```

```
                Low_f_average=Low_f_average+Half_Spectrum[a];
            else
                High_f_average=High_f_average+Half_Spectrum[a];
            }
        }
        Low_f_average=Low_f_average/first_half; \find average values of
        //spectrum within frequency ranges
        High_f_average=High_f_average/(1025-first_half);
        if ((Low_f_average/High_f_average)> frequency_threshold)
        // if a threshold was reached, register a clap
            Clap=Clap+1;
        }
    }
    else{
    i=i+1;
    }
}
function Perform_FFT(a){
array x_even;
array x_odd;
array even;
array odd;
array zeros;
array exponent;
matrix return;
int b=0;
N=length(a)
for (int i=0, i<length(a), i=i+1){
    x_odd[b]=a[i];
    i=i+1;
    x_even[b]=a[i];
    b=b+1;
}
if (N != 2){
    odd=Perform_FFT(x_odd);
    even=Perform_FFT(x_even);
    for(int l=0, l<N, l=l+1){
        zeros[l]=0;
        if (l<N/2-1){
            exponent[l]=exp(-2i*pi*l/N};
        }
```

```
        array product[1]=exponent[1]*even[1];
        return[1]=[(odd + product)];
        return[2]=[(odd-product)];
        return return;
    }
}
else{
    return[0][0]=return[0][1]=return[1][0]=a;
    return[1][1]=-a;
    return return;
}
}
```

Initially it was expected that splitting the frequency ranges at 10kHz would provide a clear difference between the average frequency values. However, it was found that the difference between average frequency amplitude values of clap and speech signal when the frequency ranges are split at 10kHz is very small. Looking back at figure 3.3 it was noticed that the frequency amplitude values start dropping at approximately 5kHz.
In order to find a fitting threshold value, an array was created to store the coefficient of average frequency values every time the threshold was reached. It was decided to search for the highest value reached by speech input. It was found that the speech input would not exceed the threshold of 0.78. Although using this threshold value the system occasionally filters out a clap input, it assures that speech will not become a trigger.

3.4 Comparison

In order to compare the effectiveness of the developed frequency-based clap detector with a amplitude-based clap detector, another testbench was built. Just as previously the testbench provided in Appendix A was used to record samples. This way it was ensured that inputs are identical.

3.4.1 Amplitude based clap detector

To have a point of reference for comparison, an amplitude based clapper was developed. Initially it was set to be identical to the frequency based clapper.
However, it was found that when the amplitude threshold is set to a value as low as for the frequency based clapper, the system would get triggered multiple times by speech input. That was the issue with the original commercial clapper. This problem could either be ignored or the threshold value could be increased, yet increasing the threshold value would decrease the sensitivity.

Another issue was the setting of clap frequency. The original clapper device [7] was recording number of threshold reaches during a time period (1.5 seconds). An assumption is made that the original clapper had a debounce function to avoid being quickly triggered multiple times by loud speech. It was decided to wait for about a third of a second (15000 samples) after a threshold is passed as it made sense that under normal circumstances clap frequency would not exceed 3 claps per second. Pseudo-code of the program looked can be found here:

```
array data; //imported from previous testbench
int Clap_count=0;
int Sample=0;
int wait=15000; //third of a second
threshold=0.3;
while(Sample < length(data){
    if (abs(data(Sample))>threshold){
        clap=clap+1;
        Sample=Sample+wait;
    else
        Sample=Sample+1;
    }
}
```

3.4.2 Results

Having both frequency and amplitude based clap detectors working, comparison of the two could be performed.
Five different samples were taken into consideration: Clap sounds with silent background, speech in silent background, clap sounds and speech peaking at about the same level and q few samples of domestic noises: knocking on different materials and vacuum cleaner.All of these samples can be found in figure 3.5
Samples 1, 2, 4 and 5 were recorded using a computer microphone. Sample 3 was synthesized by resizing sample 2 and adding it to sample 1. For sample 4 knocking on wood and placing a glass on plastic table was sampled. For sample 5 a vacuum cleaner Bosch BGL25MON6 was used.
The provided samples were used as input for program provided in appendix A.

Test 1

Inputting clap sounds with silent background has provided a perfect result from both systems (figure 3.6).

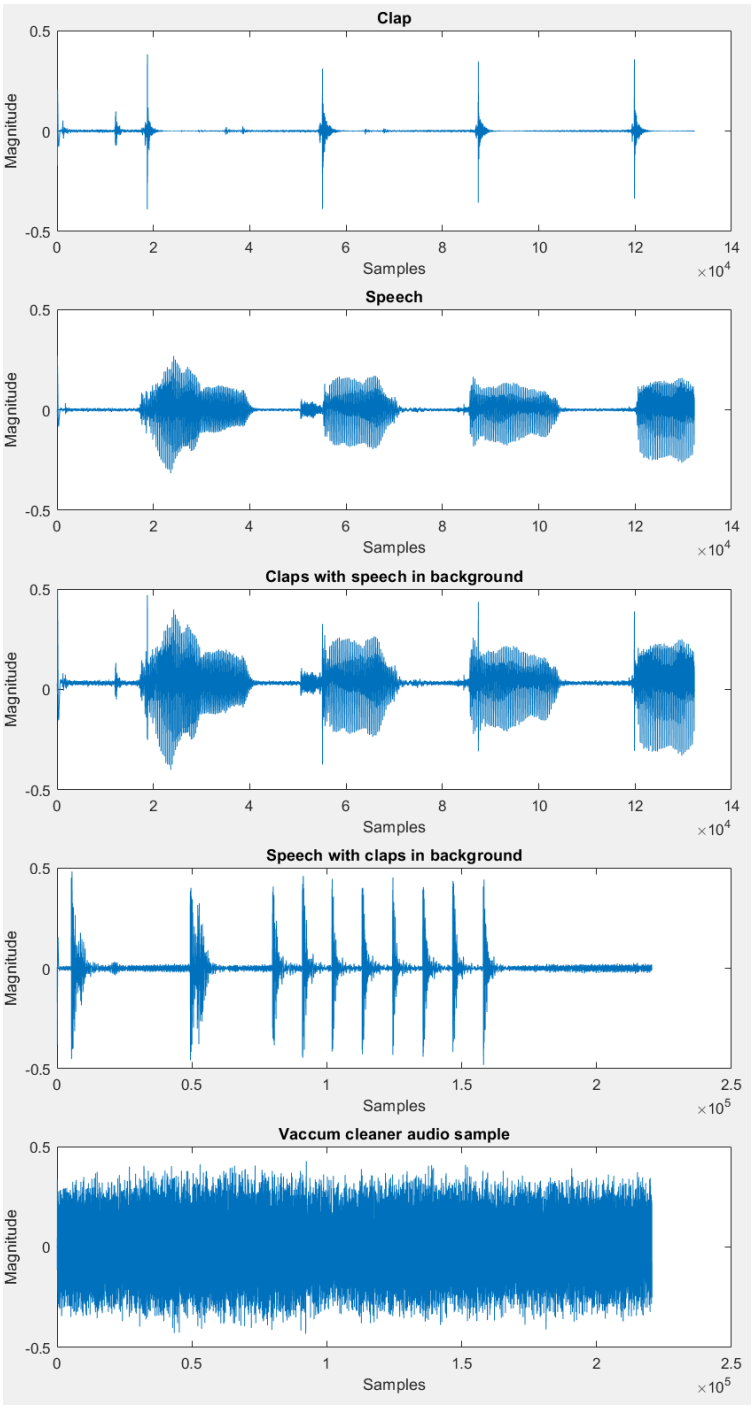

Figure 3.5: The five samples used as inputs to generally evaluate the developed system from to to bottom:
Test 1 sample: 4 hand claps over time period of 3 seconds;
Test 2 sample: speech sample of counting up from 1 to 4 over time period of 3 seconds;
Test 3 sample: synthesized sum of the two previous samples resized to have even peak values (to vary within the same range);
Test 4 sample: sounds of placing a glass on a wooden table twice; and knocking on the same table 8 times over time period of 5 seconds;
Test 5 sample: constant vacuum cleaner sound over time period of 5 seconds

;

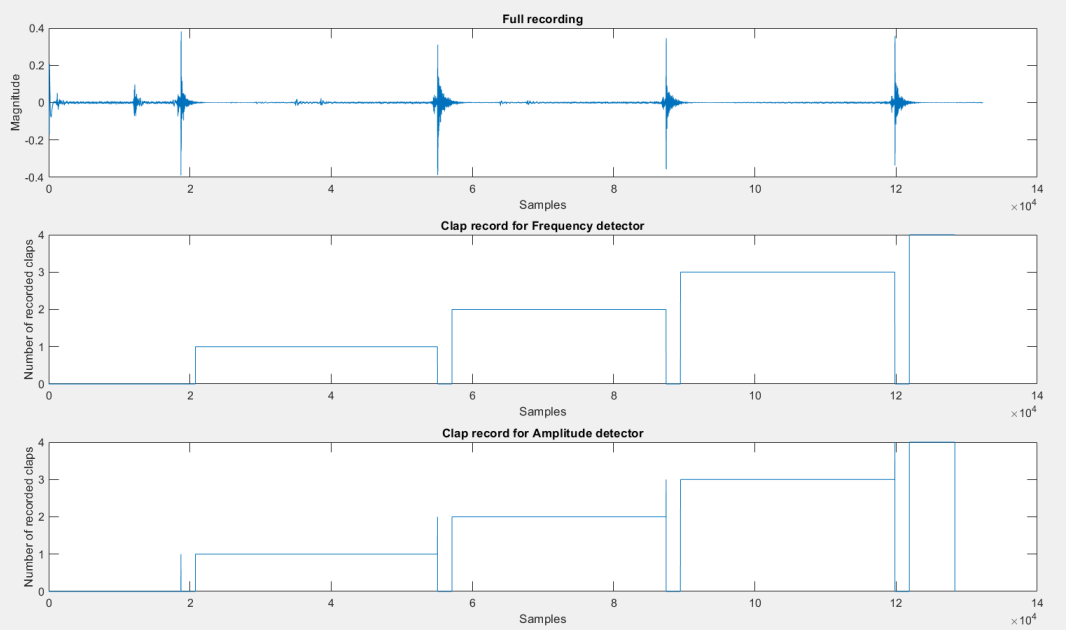

Figure 3.6: Comparison between the two system responses, test 1:
The top figure represents the input: 4 hand claps over time period of 3 seconds;
The middle figure represents the frequency based trigger response to the input;
The bottom figure represents the amplitude based trigger response to the input;
The presence of gaps where the clap record goes to 0 among values indicate that the threshold is reached;

Test 2

Inputting speech into aforementioned systems has proven the benefit of a frequency based trigger over an amplitude based one: the frequency based trigger can be calibrated for higher sensitivity yet avoid being triggered by some sounds that are situated within the higher end of human hearing frequency spectrum. Results can be seen in figure 3.7. However, it should be mentioned that both times the

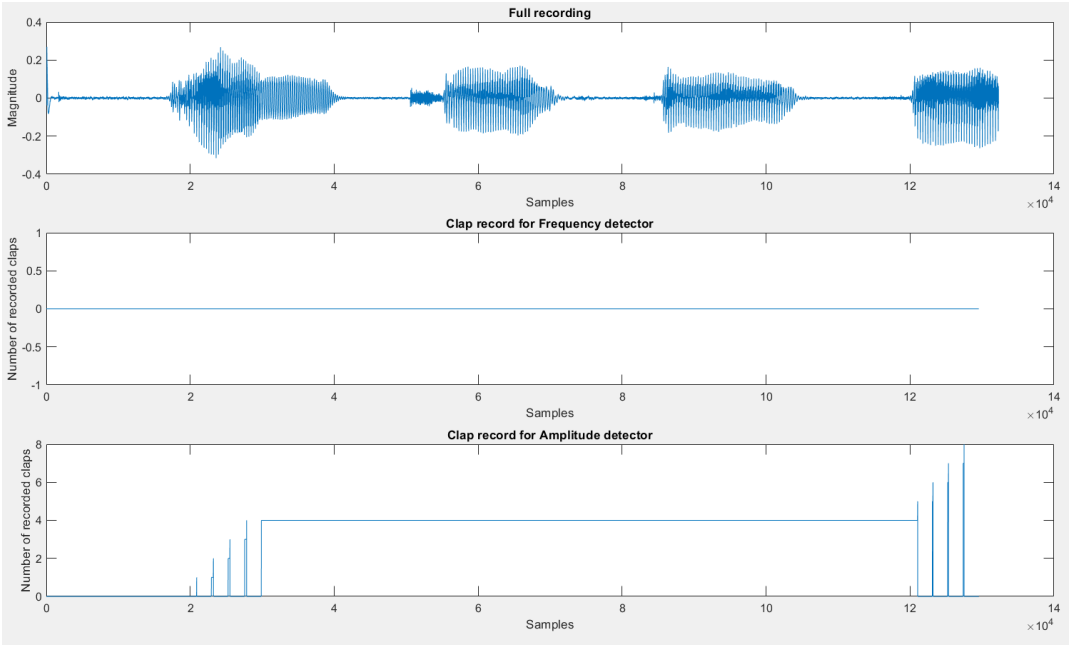

Figure 3.7: Comparison between the two system responses, test 2:
The top figure represents the input: speech input of counting up from 1 to 4 over time period of 3 seconds
The middle figure represents the frequency based trigger response to the input
The bottom figure represents the amplitude based trigger response to the input
Gaps where the clap record goes to 0 among inbetween values indicate when threshold is reached

amplitude based counter has been triggered, it counted up to 4 which would be considered as an unintentional signal by the sequence detector used in the patented clapper [7] and therefore would not trigger any device toggles.

Test 3

The results of test 3 can be found in figure 3.8. The test indicates imperfections in both systems. The frequency based filter only detects half of the claps that were produced during the recording period. On top of that, it also takes two buffers of speech input and registers these as clap signals. However, the amplitude based clapper detects 19 claps, which is also incorrect. Although neither of the detectors

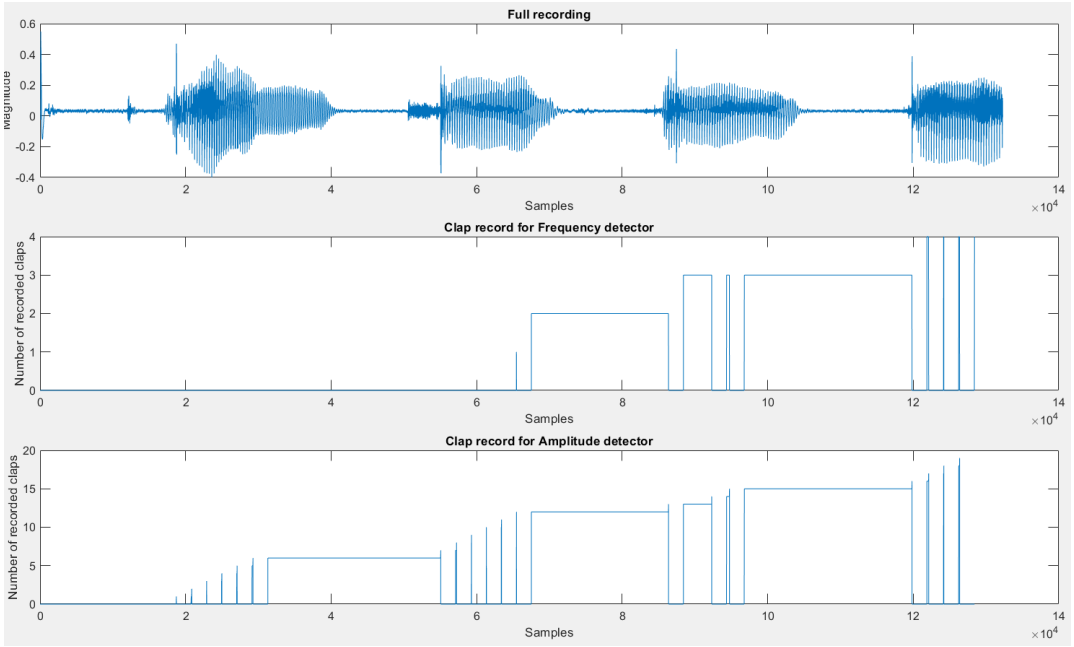

Figure 3.8: Comparison between the two system responses, test 3:
The top figure represents the synthesized sum of the samples from tests 1 and 2 resized to have even peak values. The middle figure represents the frequency based trigger response to the input
The bottom figure represents the amplitude based trigger response to the input
Gaps where the clap record goes to 0 among inbetween values indicate when threshold is reached

performed great during this test, the accuracy rate of the frequency based clapper has still proven to be far higher than one of the amplitude based trigger.

Tests 4 and 5

The purpose of these tests was to see how the frequency based clap trigger would compare to the amplitude based one in real life. These tests were quite abstract: the amplitude and frequency of the sounds recorded in these samples may vary widely depending on multiple factors. None the less, these tests do provide a rough view of what could be expected. Test 4 (figure 3.9) was created in order to see if there are any differences between a sound of clap and a sound of interaction with wood.

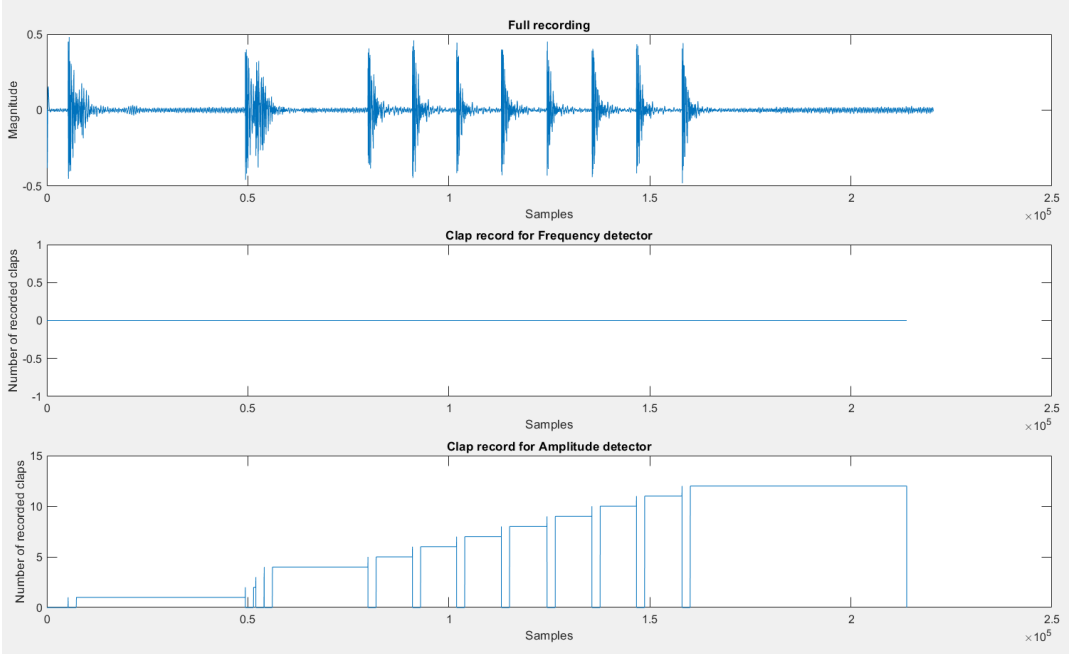

Figure 3.9: Comparison between the two system responses, test 4:
The top figure represents the input: first 2 impulses representing the sound of a glass touching a wooden table followed my 8 impulses of hand knocking on the same wooden table over time period of 5 seconds
The middle figure represents the frequency based trigger response to the input
The bottom figure represents the amplitude based trigger response to the input
Gaps where the clap record goes to 0 among inbetween values indicate when threshold is reached

Unexpectedly it was found that impulses mostly dominated within the 0 to 500 Hz area of the frequency spectrum and therefore did not trigger the frequency based system.
Furthermore, the vacuum cleaner that was used also appeared to use the lower frequency range what also didn't trigger the system (figure 3.10

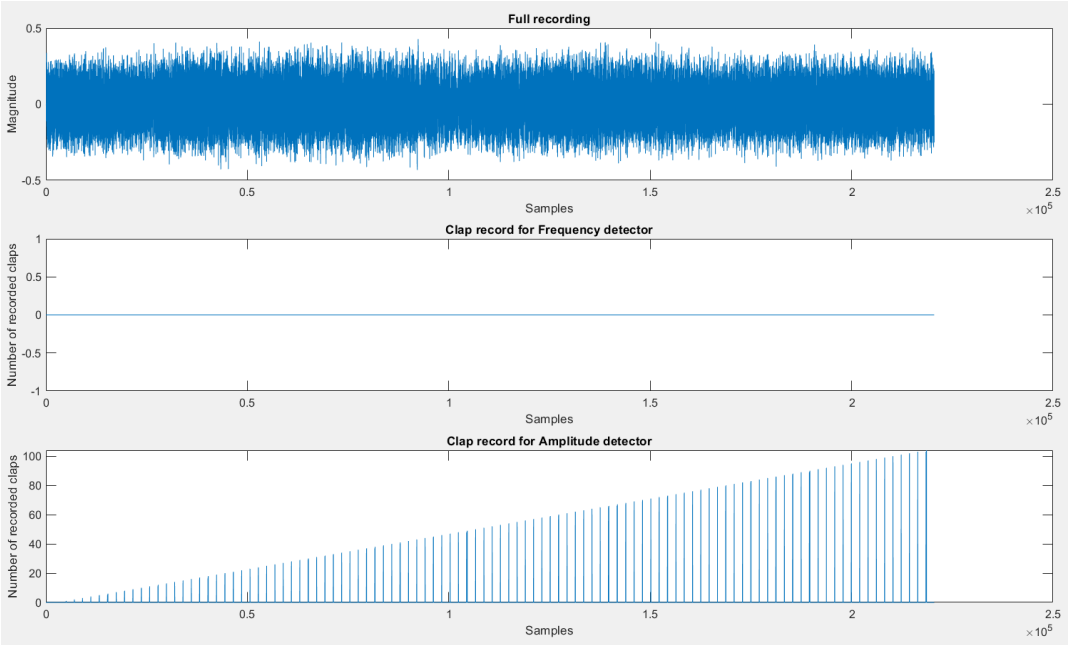

Figure 3.10: Comparison between the two system responses, test 5:
The top figure represents the input: Constant vacuum cleaner sound over time period of 5 seconds
The middle figure represents the frequency based trigger response to the input
The bottom figure represents the amplitude based trigger response to the input
Gaps where the clap record goes to 0 among inbetween values indicate when threshold is reached

3.5 Conclusion

The MATLAB implementation has utilized the mathods presented in the previous chapter. It was a successful step towards implementation and has provided clear insights into what could be expected from the implementation with a microcontroller. It also indicated the potential strengths and weaknesses of such system.

Chapter 4

Microcontroller implementation

4.1 Hardware selection

4.1.1 Microcontroller

As the target signal frequencies for this project are between 10 and 20kHz, the microcontroller needs to be capable of sampling at 40kHz.
Considering that single sample size of 10 bits should suffice and it is planned to take samples for about 20ms, the recording will demand at least 8820 bits of memory.
Having these requirements in consideration and aiming for comfortable usage ATMEL SAM3X8E microprocessor was chosen. For the sake of convenience it is embedded in Arduino DUE microcontroller.

4.1.2 Microphone circuit

The microphone needs to produce as little noise as possible. Ideally the signal preprocessing would be performed before the signal reaches the microphone in order to keep the micro-controller as minimally occupied as possible.
It was decided to use an electret microphone (figure 4.1). This type of microphone is similar to condenser, yet is cheaper, very widely applied in the market and therefore easy to obtain. Furthermore, it does not require phantom power. [3].

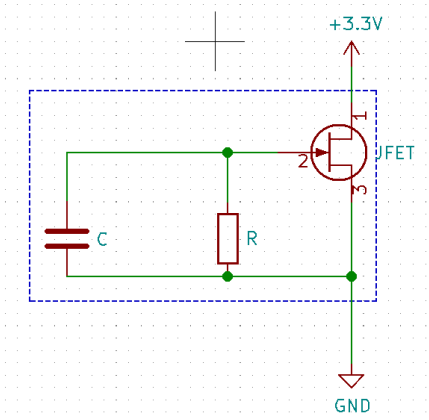

Figure 4.1: Electret microphone structure. The capacitor represents the diaphragm moved by changes in air density (waves)

It is possible to obtain the signal using a resistor and a capacitor connected as shown in figure 4.2. However, the obtained signal is very weak and unfitting for the analog-to-digital conversion. In order to condition the signal for this application, a following circuit was developed.

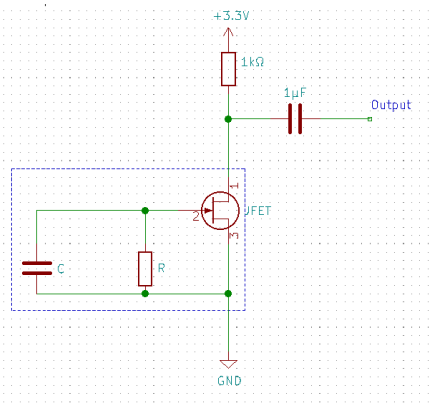

Figure 4.2: Initial circuit for to obtain audio signal

The circuit consists of the previously used resistor and two capacitors connected in parallel to eliminate DC offset and replace the old one while maintaining it's capacitance value. The capacitors are connected to a resistor. Because the capacitor and resistor create a high-pass filter, resistor value was chosen to be 51kΩ in order to set the cutoff frequency to approximately 3.3 Hz according to equation 4.1.

$$f_c = \frac{1}{2\pi \cdot C \cdot R2} \tag{4.1}$$

In order to significantly increase the gain, a resistor of 1MΩ was placed to create a

factor 20 according to operational amplifier formula 4.2.

$$V(out) = \frac{V(in) \cdot R3}{R2} \tag{4.2}$$

To perform unipolar conversion, operational amplifiers positive input was connected to a voltage divider to create a output signal that would vary around the point of half the input voltage 4.3.

$$V(out) = \frac{V(in) \cdot R4}{R4 + R5} \tag{4.3}$$

The entire circuit schematic can be found in figure 4.3 as well as its realisation in figure 4.4

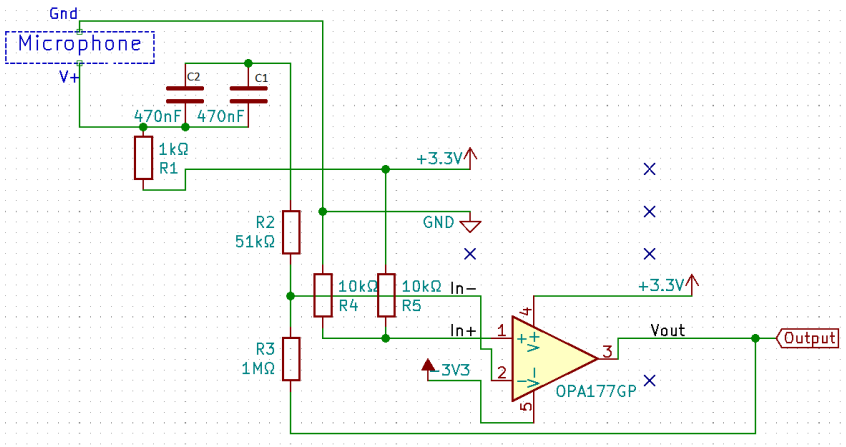

Figure 4.3: Schematic of the initial circuit for to obtain audio signal. The main objective of this schematic is to clarify the connections in figure 4.4. Therefore it is intended to resemble the picture as much as possible

4.2 Program

4.2.1 Reading samples

In order to implement a clap detector, a program to read ADC samples was required. Using arduino, this could be done by using ***analogRead*** command. However, by using digital output to indicate the start and end of each output it was found that the controller does not sample at consistant frequency: it varies around 100kHz. This sampling method was deemed unfitting for the application because in order to perform frequency analysis a constant sampling speed is required. Therefore it was decided to use ATMEL SAM libraries to directly control the microcontroller's registers.

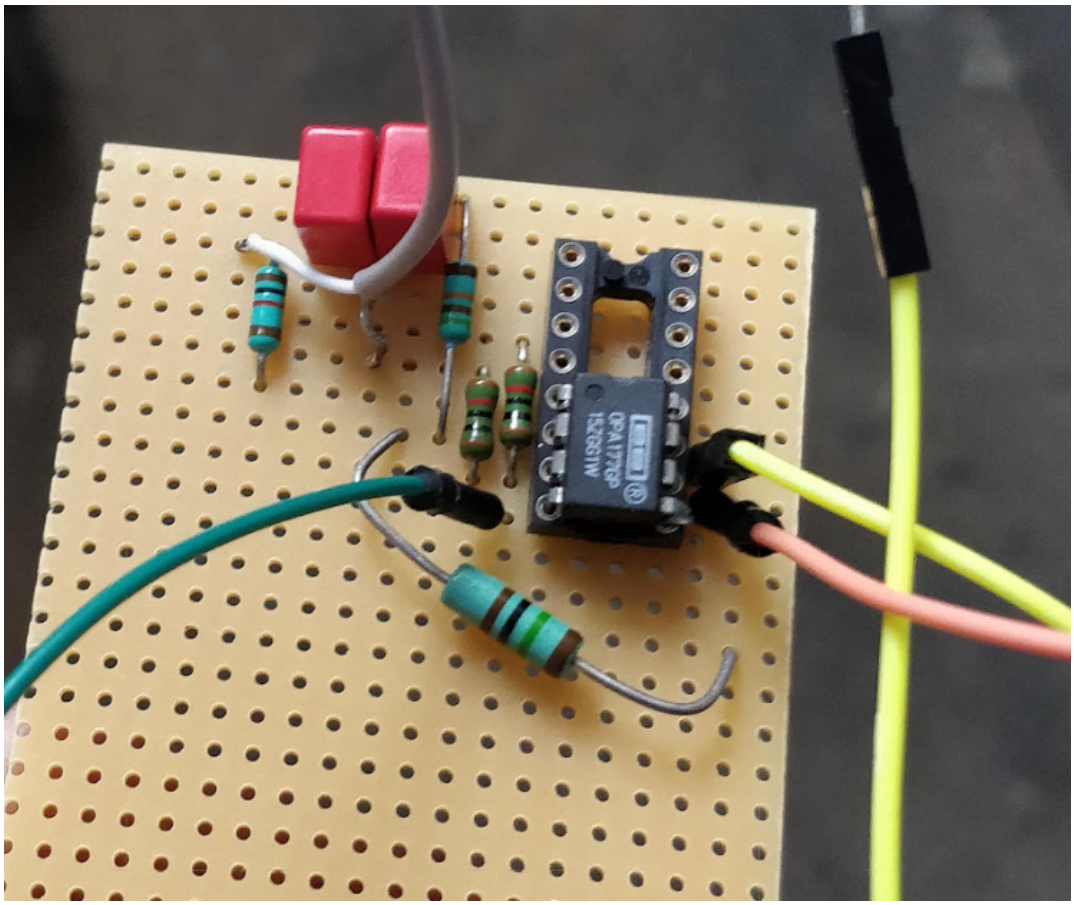

Figure 4.4: Initial circuit to obtain audio signal. The circuit was fed by external power supply

4.2.2 Structure

The objective of the program were implementation of Cooley-Tukey algorithm and clap detection.
In order to have a constant sampling speed, an interrupt service routine was used. A buffer was created to store the values obtained after detecting amplitude threshold reach. In order to track the timing within the program, digital pins were toggled and observed through an oscilloscope.
It was decided that during Fourier transform computation sampling should not be performed for multiple reasons:

- In order to obtain the transform as quickly as possible, no other actions should take place during the computation (a short exception was made for the ISR).
- If the computation would turn out to be too slow, values that were stored in the buffer could be overwritten by new samples.
- Even if Fourier transform would be computed faster than the sampling rate,there is no way the next sample would contain another clap signal (multiple claps within that short time period would be impossible) and therefore it would indicate a false detection what would deem the output to be worthless.

Having these in mind it was decided to create a structure in the ISR to avoid having both sampling and Fourier transform computation running simultaneously. The structure can be found in figure 4.5

4.3 Fast Fourier Transform implementation

The implementation of the transform in MATLAB has bee relatively easy since MATLAB can compile complex numbers, return matrices from functions and perform multiple other computations without the user's interference. This has caused multiple issues while attempting implementation of this algorithm in C++.

4.3.1 Complex numbers

Since arduino IDE does complex values, complex numbers were split into two arrays: one to store real and one to store complex part of the values.

Recursion

As recursion demands a function to return a value, it was written to return an array which would further be used in FFT computation as shown in figure 2.2.

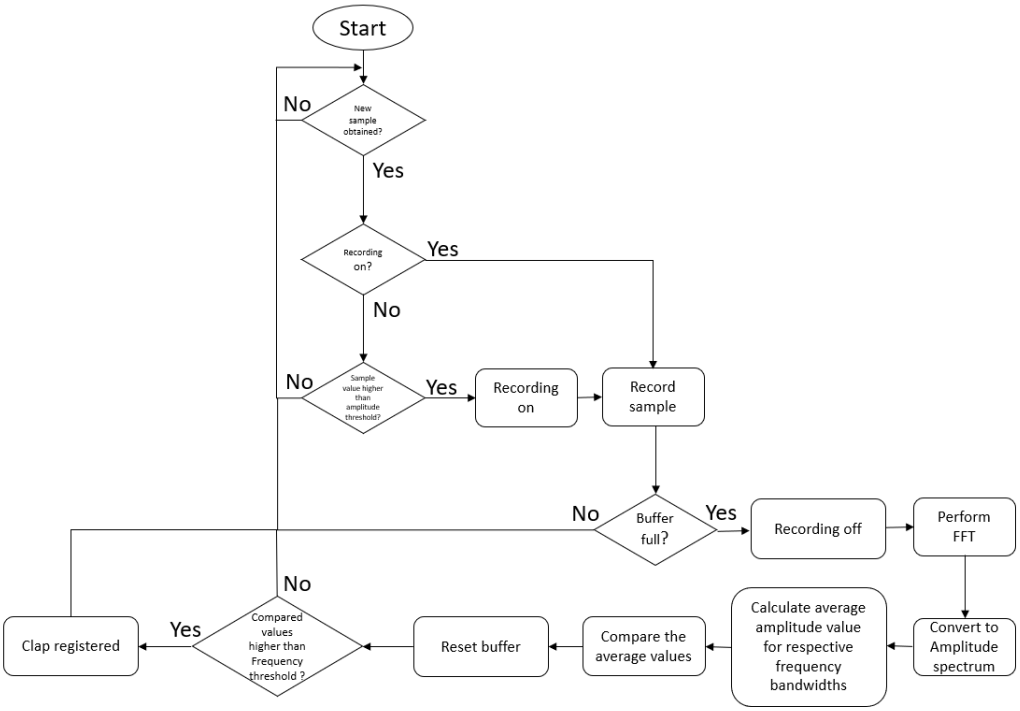

Figure 4.5: State diagram of the program

However, since functions cannot return arrays, pointers were used to link towards the target arrays.
On top of the previous issue, the recursive function was returning multiple errors due to mismatching variable dimensions. As this problem has not been resolved within a couple of days, it was decided to try out a different method.

4.3.2 Predetermination function

Instead of using a recursive function if was decided to create a struct that would contain arrays all the variables that change every time the recursive function is called, Before Fourier transform computation, another function would be called to compute the values of the variables that could be predetermined as well as number of times the function would be repeated. Then, instead of using the function recursively, it would be used the number of times that was determined. The program this method was inspired by can be found here:[4].

4.3.3 Result

Using the predetermination function approach has proven to be successful as the program was compiling successfully. However, introducing multiple additional

arrays has caused the program to slow down to the point it was no change in digital pin output would be noticed within a minute after the program was compiled.

4.4 Conclusion

As the circuit was developed to input the values to microcontroller, a program could be written to read these values at constant frequency. Having the constant reading speed, a setup structure could be made to prepare the values for FFT computation. As the setup was ready, FFF function was adapted to the program. Unfortunately, the method used to compute the result has made the algorithm obsolete for this application.

Chapter 5

Conclusion

The project was started by analysing cases where Fourier analysis has increased the accuracy of an unreliable system. Then an analogy was drawn between clapper and early earthquake detection systems, indicating the similarities between the cases. As the case was exposed, a hypothesis could be made that a similar to early early earthquake detection systems' solution could increase the successful detection rate.
A research was done before starting diving into development. Having a hypothesis, testbench was developed in MATLAB to analyse the problem and find whether the hypothesis holds true. Having the hypothesis confirmed, tests were made to further analyse the properties of the system and indicate it's potential strengths and weaknesses. An attempt was made to implement the obtained system into a microcontroler. Although the program was capable of running, demanding implementation methods have made the completely unfitting for the application

5.1 Perspectives

5.1.1 Fourier analysis

Knowing that frequency phase related data is irrelevant for this project, real fourier transform could be used instead of complex one. This solution would help improve the program performance.

5.1.2 MATLAB analysis

It was found that MATLAB implementation has enabled program testing as well as detailed analysis and troubleshooting of programs that have branched from it. Although the current tests could not be considered conclusive due to small amount of samples, it has given an approximate idea of areas where such system would excel over standard clapper.

5.1.3 Microcontroller implementation

Although a system was successfully created to sample data output Fourier transform, further improvements need to be made in order to obtain a working prototype:

- As mentioned previously, real Fourier transform could be implemented instead of complex.
- A recurive program should be reattempted to create as it should compute faster. However, some sources call the Cooley-Tukey version that was presented as pseudo-code in Matlab implementation section slow. A similar implementation claims to be 8 times faster **??** It would be interesting to implement it and compare the computation speed of the programs.

When the sampling program would be working, a sequence detector could easily be added to have an identical system to the original clapper(just without the 'away' mode).

Bibliography

[1] Charles J Ammon. *Seismic Waves and Earth's Interior*. URL: http://eqseis.geosc.psu.edu/cammon/HTML/Classes/IntroQuakes/Notes/waves_and_interior.html.

[2] Charles J Ammon. *Waves, Seismograms, and Seismometers*. URL: http://eqseis.geosc.psu.edu/cammon/HTML/Classes/IntroQuakes/Notes/seismometers.html.

[3] The Broadcast Bridge. *What's the Difference Between a True Condenser and an Electret Condenser Shotgun Microphone? - The Broadcast Bridge - Connecting IT to Broadcast*. URL: https://www.thebroadcastbridge.com/content/entry/6180/whats-the-difference-between-a-true-condenser-and-an-electret-condenser-sho.

[4] *Cooley-Tukey algorithm implementation in C++*. URL: https://github.com/otsaregorodtsev/cooley-tukey-fft.

[5] James Schloss. *FFT*. URL: https://www.algorithm-archive.org/contents/cooley_tukey/cooley_tukey.html.

[6] Steven W Smith. *The Scientist and Engineer's Guide to Digital Signal Processing By Steven W. Smith, Ph.D.* URL: https://dspguide.com/.

[7] Carlile R Stevens and Dale E Reamer. *US5493618A - Method and apparatus for activating switches in response to different acoustic signals*. URL: https://patents.google.com/patent/US5493618A/en.

[8] T. Xia et al. "Wide-area Frequency Based Event Location Estimation". In: *2007 IEEE Power Engineering Society General Meeting*. 2007, pp. 1–7. DOI: 10.1109/PES.2007.386018.

Appendix A

Programs

A.1 MATLAB recording and automatic FFT function program

```
%% CLEANING EVERYTHING
clc;
clear;

%% SETUP
Freq = 44100;              %Sampling rate frequency in Hz
BitsSample = 16;           %Bits per sample
Channels = 1;              %Channels 1(Mono) or 2(Stereo)
Microphone = audiorecorder(Freq,BitsSample,Channels);
Sample_length = 2048;
Threshold = 0.2;

%% RECORDING
get(Microphone);
disp('start')
record(Microphone,10);
pause(10);
Test_Recording = getaudiodata(Microphone);
disp('stop')

%% SAVING FILES
filename1 = 'Test1.flac';
audiowrite(filename1,Test_Recording,44100);
```

```
%% CREATE SHORT DATA SAMPLE
[Data,Fs] = audioread('Test1.flac');
i=0;
while abs(i) < Threshold
    i=Data(j);
    j=j+1;
end
a=1;
for a=1:Sample_length
    Sample(a)=Data(j);
    a=a+1;
    j=j+1;
end
%% PLOTTING
figure
hold on
subplot(3,1,1);
plot(Data);
title('Full recording')
xlabel('Samples')
ylabel('Magnitude')
subplot(3,1,2);
plot(Sample);
title('Sample for Fourier transform')
xlabel('Samples')
ylabel('Magnitude')

%% FFT
fs = Freq
n = length(Sample);
l = fft(Sample);
P2 = abs(l/n);
P1 = P2(1:n/2 + 1);
P1(2: end - 1) = 2*P1(2: end - 1);
f = fs*(0:(n/2))/n;
subplot(3,1,3);
plot(f, 20*log10(P1));
%ylim([-220 -50])
title('Amplitude spectrum')
xlabel('Frequency (Hz)')
ylabel('Magnitude (Db)')
```

A.2 MATLAB clap triggers comparison program

```
%% CLEANING EVERYTHING
clc;
clear;

%% SETUP
Length = 5;                %Length= 5 or 3 depending on sample number
Freq = 44100;              %Sampling rate frequency in Hz
Sample_length = 2048;
FThreshold = 0.8;
AThreshold = 0.2;

%% CREATE SHORT DATA SAMPLE
[Data,Fs] = audioread('Test5.flac'); %Test samples 1 to 5
Clap=0;
clap=0;
plots=zeros(Length*Freq,2); %plotting the number of claps for both systems
b=0;  %Testing variable to track coefficients Favg/Savg
%and find the threshold
k=5000;
j=5000; %%5000 because initial with higher values samples
%trigger low Threshold for samples of silence
%% Clappers
while k < (Length*Freq)
    if abs(Data(k,1)) > AThreshold
        clap=clap+1;
        plots(k,2)=clap;
        k=k+Sample_length;
    else
        plots(k,2)=clap;
        k=k+1;
    end
end
while j < (Freq*Length)
    if abs(Data(j)) > AThreshold
        Favg=0;
        Savg=0;
        for a=1:Sample_length
            Sample(a)=Data(j);
            j=j+1;
```

```
        end
        Sample=Sample';
        A=myFFT(Sample);
        N=length(Sample);
        P2 = abs((A)/N);
        P1 = P2(1:N/2 + 1);
        P1(2: end - 1) = 2*P1(2: end - 1);
        P1=20*log(P1);
        Sample=Sample';
        for a=1:length(P1)
            if a<205
                Favg=Favg+P1(a);
            else
                Savg=Savg+P1(a);
            end
        end
        Favg=Favg/205;
        Savg=Savg/820;
        b=b+1;
        Avg(b)=Favg/Savg;
        disp(b);
        if (Favg/Savg) > FThreshold
            Clap=Clap+1;
        end
        plots(j,1)=Clap;
    else
        j=j+1;
        plots(j,1)=Clap;
    end
end
%% PLOTTING
subplot(3,1,1);
plot(Data);
title('Full recording')
xlabel('Samples')
ylabel('Magnitude')
subplot(3,1,2);
plot(plots(:,1));
title('Clap record for Frequency detector')
xlabel('Samples')
ylabel('Number of recorded claps')
```

```
subplot(3,1,3);
plot(plots(:,2));
title('Clap record for Amplitude detector')
xlabel('Samples')
ylabel('Number of recorded claps')

%% FFT
function X= myFFT(x)
N = length(x);
xodd = x(1:2:end);
xeven = x(2:2:end);
if N>=4
    odd = myFFT(xodd);
    even = myFFT(xeven);
    X = zeros(N,1);
    exponent = exp(-1i*2*pi*((0:N/2-1)')/N);
    P = exponent .* even;
    X = [(odd + P);(odd -P)];
else
    X = [1 1;1 -1]*x;
end
end
```