Sound And Light Sensor system For A Musical Instrument

Steinþór Jasonarson

Faculty of Electrical and Computer Engineering
University of Iceland
2017

SOUND AND LIGHT SENSOR SYSTEM FOR A MUSICAL INSTRUMENT

Steinþór Jasonarson

24 ECTS thesis submitted in partial fulfillment of a
Baccalaureus Scientiarum degree in Mechatronics Engineering

Advisors
Þórður Halldórsson & Behnood Rasti

Faculty of Electrical and Computer Engineering
School of Engineering and Natural Sciences
University of Iceland
Reykjavík, May 2017

Sound and Light Sensor System
for a Musical Instrument

24 thesis submitted in partial fulfillment of a Baccalaureus Scientiarum degree in Mechatronics Engineering

Faculty of Electrical and Computer Engineering
School of Engineering and Natural Sciences
University of Iceland
Grænásbraut 910
235 Reykjanesbær, Reykjavik

Telephone: 578 4000

Bibliographic information:
Steinþór Jasonarson, 2017, Sound and Light Sensor System
for a Musical Instrument, B.Sc. degree, Faculty of Electrical and Computer Engineering, University of Iceland.

Printing: Svansprent, Auðbrekku 12, 200 Kópavogur
Reykjavik, May 2017

Útdráttur

Í þessari ritgerð er lagður grunnur að hljóðgreiningartæki fyrir tónlistarnám. Nýjungin við tækið er samtenging hljóð og ljóss til að styrkja og hraða lærdóm nemandans. Tækið gefur notanda tækifæri til að velja vissa liti og tengja þá við vissar nótur, hljóma eða týpur af hljómum.

Tækið samanstendur af nokkrum íhlutum sem, þegar þeir hafa verið tengdir saman, kallast ígreypt kerfi. Helstu íhlutir eru örtölva, (e. microcontroller), bluetooth samskiptabúnaður, hljóðnemi og Light Emitting Diode (LED) ljós. Greining og val á örtölvu er gerð til að finna örtölvu sem hentar best að greiningar þörfum tækisins ásamt útreikningum á orkunotkun frumgerðar.

Algrím sem hentar í greiningu á tónum og hljómum er rannsakað og hannað sem byggir á Fast Fourier Transform (FFT) greiningu hljóðs. fundin er leið til að auka nákvæmni tíðnigreiningar, til að útiloka bakgrunnshljóð, og eyða tíðnum sem eru margfeldi af grunntíðni nótu. Algrímið er svo prófað á þessum forsendum til að greina virkni þess.

forrit fyrir síma var hannað til að leyfa notanda að stilla liti við vissar nótur og hljóma ásamt því að virkja greiningu á nótum eða týpur af hljómum.

niðurstaðan er að frumgerð tekst tilætlunarverk sitt, nær að greina bæði nótur og hljóma og gefa frá sér litað ljós sem endurspeglaði val notanda.

Abstract

In this thesis, a prototype for a sound analysis device aimed for musical education is designed and developed. The novelty of the device is the confluence of the aural and visual medium to reinforce and hasten the learning process. This is in the form of user selectable colours to certain notes or type of chords.

The prototype device is comprised of several components that, when assembled are called an embedded system device. Its main components are a micro-controller, a bluetooth communication module, an audio sensor module and Light Emitting Diode (LED)s. The requirements for each part is explained along with the selection process.

An algorithm for characterizing sound into notes and chords is researched and designed based on FFT analysis of the sound. A method to accurately detect the frequency is detailed as well as a method to filter background noise and remove frequencies that are harmonic multiples of the base frequency. The algorithm is then tested on these methods to check for effectiveness.

A cellphone application (app) is outlined and a prototype of it developed that allows the user to enable or disable detection of single notes, intervals, triads or seventh chords, as well as set specific colours to specific notes and chords.

The prototype can successfully convert both notes and types of chords to a colour selected by a user through a cellphone app.

Preface

The original idea behind the project is sparked by what can only be described as an pet peeve since I first saw a CD player with a graph of just the FFT of the sound that was playing (detailed in ch. 2.1). The information contained in such a graph isn't helpful to the viewer to understand the music in a simple way.

Then last year, a discussion with Pianist Laufey Sigrún Haraldsdóttir regarding music and colours gave me the idea for this thesis. The idea was to convert notes and chords to colour, which incidentally describes the condition a few people have called Synæsthesia [1]. Taking the basic idea of converting sounds to colours is an enriching idea regarding the musical experience, and can be used to develop a learning tool when learning notes, chords or progressions. Allowing the user to configure what colours adhere to which notes gives them a platform to detect and enjoy the music on another visual level.

Steinþór Jasonarson

Contents

List of Figures

List of Tables

Abbreviations

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

I2S Inter-Integrated Circuit Sound

LED Light Emitting Diode

PAN Personal Area Network

PCB Printed Circuit Board

RF Radio Frequency

RGB Red/Green/Blue

Acknowledgements

I want to thank Laufey Sigrún Haraldsdóttir for the discussions about music, Krista Hannesdóttir for her assistance as well as my advisors Þórður and Behnood for their guidance in making this thesis.

1. Introduction

Sound analysis is a multi-faceted platform for research which can affect fields as varied as the medical, structural, analytical, and musical fields, depending on focus and interest. There have been large strides in various analysis, from analyzing voice commands, frequency tuners, tonal diagnostics of recorded data, and much more.

The analysis of music is one of the fields being researched, and many different applications have been made for the musician, for practice (like instrument tuners) or for study, but live analysis of music has not come as long a way. Earlier research into specific solutions have been done, for instance research into splitting a human voice from the background music [2], or the live analysis of a single tone [3].

Earlier visualization attempts of music try and procedurally create landscapes [1], or colours and shapes based on volume [2], but the conversion does not contain information about notes or chords being played.

Visualizing sound as coloured light is a neurological phenomenon in people called chromesthesia which is a form of synæsthesia [1]. People with chromesthesia show no general standards of which colours adhere to which sounds, rather each person has their own perception of what colour is associated with what sound [4].

The novelty of this thesis is the algorithm that converts sound to information about the notes being played, and the information is then used as a visual aid in the form of coloured light, where the colours are pre-chosen by the user.

This project aims to use a sound sensor connected to an embedded system to monitor musical notes from an instrument, and develop an algorithm that lights up RGB LED lights based on given notes. Also to develop a user interface through an app for the user to configure which colours conform to which notes or chords, and finally to create a prototype showing the functionality of the idea.

[1] `http://luminantmusic.com/`
[2] `http://www.atarihq.com/dedicated/videomusic.php`

1.1. Problem Statement

The main objective of this project is to create and develop an algorithm to convert sound sensor data to musical notes and chords data, to develop and create a prototype solution which could be comfortably placed beside a musical instrument, and under its own power analyze the notes being played and change the colours of an RGB LED light according to the notes. Also an android application will be developed which can communicate with the device and through it allow the user to configure the colour value of notes, chords or chord types.

The project aims to design and implement a microprocessor prototype device that can characterize the notes of a musical instrument and give visual feedback in the form of coloured light.

Project Limitations

The project is restricted to the following scopes:

- A comparison of suitable microprocessors.
- The development and testing of the algorithm.
- The development of a prototype device and user app that illustrates the basic functionality of the design.

1.2. Thesis Overview

The thesis is organized as follows:

Chapter 2 introduces the terminologies regarding music theory and also gives a mathematical background for the analysis used in the thesis.

Chapter 3 discusses the requirements of the device, and the individual parts chosen. 3 different microprocessors will be compared to each other and the most optimal chosen for the prototype.

Chapter 4 details the methods of the algorithm to detect, filter and convert sound into notes.

Chapter 5 focuses on creating the prototype and outlines the android app and its communication with the embedded system via Bluetooth.

Chapter 6 goes into the results from experimental measurements of the prototype and interprets the results.

Chapter 7 presents the thesis' conclusion and discusses the future of this project and what can be improved.

Additional literature, datasheets and a time plan for the thesis can be found in the appendix.

2. Background

In this chapter the mathematical tools and techniques used to analyze the sound as well as music theory concepts in regards of converting frequencies into recognizable notes are explained.

2.1. Fast Fourier Transform

The electrical signal from a microphone is a specified amount of voltage per time period. The frequencies that are within the signal are gotten by transforming the signal from a time dependent analysis to a frequency dependent analysis. This can be done through the Discrete Fourier Transform (DFT) or Fast Fourier Transform (FFT) [5].

The FFT algorithm is a fast type of implementation of the DFT algorithm. The FFT uses numerical optimization and recursion to take as many shortcuts to reduce computational need, but the results are exactly the same as using the formula for DFT, see [6].

The formula for DFT [5] is:

$$\mathbf{F(n)} = \sum_{\mathbf{k=0}}^{\mathbf{N-1}} \mathbf{f(k)} * \mathbf{e}^{\mathbf{-i*k*2\pi*n/N}} \tag{2.1}$$

The symbols in eq. 2.1:

- Capital N is the number of samples being analyzed.
- Lower case n is the bin being analyzed from the signal, in integers from 0 to $N - 1$.

- $f(k)$ value is the signal voltage value.
- $F(n)$ value is the total number of times the frequency is within the signal.
- i is notation for the imaginary unit.
- e is notation for Eulers' constant.
- k is the index of summation.
- π is the symbol for pi.
- $*$ is a symbol for the elementary, mathematical operation of arithmetic; multiplication.
- $-$ is the symbol for a negative value, or the elementary, mathematical operation of arithmetic; subtraction.
- $\sum$ is the symbol for summation.

2.2. Theory Behind Musical Notes

What defines a musical note in a sound from noise is the listener. There is no physical definition that unambiguously defines a sound as music rather than noise. That being said, a close imitation of the perceived differences of noise and music can be made. For instance, music often has clear and long repetition of the frequencies while an instrument is vibrating and the sound contains natural harmonics that reinforce the base tone. A note can be said to have a base frequency and natural harmonics at frequencies that are multiples of the base frequency. Noise then has the opposite qualities: a lot of frequencies that are unharmonious and short-lived [7].

2.2.1. Division of Frequencies into a Scale

The tuning of musical instruments can be varied, but the most common tuning is the equal temperament tuning. [8]. This tuning defines 12 equal steps within a

doubling of a frequency (called an octave), so from a doubling of 440Hz to 880Hz there are 12 individual steps called semitones (counting 440 Hz as note 1 and 880Hz as note 13, which is also note 1 of the next octave, so the division is 12 unique notes). This means each step should be $2^{1/12}$ from the next, or 1.059463094 times the frequency. This is a logarithmic scale based on how the human hearing senses frequencies, i.e. notes that are duplicates of each other sound the same (440 Hz and 880Hz both 'sound' the same).

2.2.2. Standard Tuning of Modern Western Music

The standard tuning is defined from the frequency 440Hz [9] as the A above middle C on a grand piano, the lowest note is A0, or 27.5 Hz, and the highest is C8 at 4186.01 Hz, see appendix table A.1 for exact frequencies.

2.2.3. Measuring Offset or 'Error' from a Defined Note Frequency

The frequency gap between two semitones are measured in 'cents' or 100 units within the same logarithm scale, so one cent is $2^{1/1200}$ off the base note.

2.2.4. Chords in Music

Triad chords are classified depending on the number of semitones between each note. Major, Minor, Augmented, Diminished, and Suspended [10].

Seventh chords, are triads with an extra note in, named for the 'seventh' note in a 8 note scale and can either be a large 7th or a small [10].

3. Requirements Analysis and Hardware Selection

This chapter will detail the requirements of creating a prototype that can convert notes and chords into coloured lights. The critical part will be the microprocessor system used and a comparison of 3 compatible systems will be analyzed. The bluetooth chip, the RGB LEDs, battery and battery charger can be any generic unit that fulfill the requirements detailed here.

3.1. Requirements for the Sound and Light System

The general requirements are:

- RGB LEDs to display the coloured lights.
- A Bluetooth dongle for the device to connect wirelessly with a cellphone application.
- A microphone unit to capture sound and convert it to a digital signal.
- A microprocessor unit to connect to the LEDs, Bluetooth dongle, and microphone unit, process the signal and send RGB values to the LEDs.
- Total price should stay within $250 as this is privately funded.

3.2. Bluetooth

The requirements made of the Bluetooth chip is that it needs to be able to connect and communicate serially with a cellphone. Any generic Bluetooth chip fits those requirements.

Bluetooth chip chosen: HC-06 [1] was ordered which cost $7 without shipping. It was chosen based on personal user experience in older projects.

3.3. Embedded System

In this section we will discuss the embedded system options, detail the systems based on a given number of criteria and value them. The embedded system chosen is based on these requirements:

- Max size : Must fit within a box of dimensions: 15x10x5 cm^3 or less.
- processing speed : Needs to be able to process 30 FFT samples per second and it must have capacity to spare for other operations.
- library support : Must support FFT functions to reduce developer overhead.
- audio capabilities : Library support necessary and preferably an embedded microphone solution inbuilt to reduce developer overhead.

3.3.1. PIC32MZ2048EFH064

- Model: PIC32MZ2048EFH064 [11].
- Size: Not applicable, needs PCB work.
- Processing speed: 252MHz.
- Library support: Supports FFT functions of variable Log2 size.

[1] `http://www.instructables.com/id/Add-bluetooth-to-your-Arduino-project-ArduinoHC-06/`

- Audio capabilities: Library support, but no audio solution embedded.
- Development overhead: Choosing this system would mean developing the PCB board and audio circuitry from scratch.
- Price: 11.32 - only the chip.

3.3.2. Teensy 3.2

- Model: Teensy 3.2 [12].
- Size: 18mmx34.5mm.
- Processing speed: 72MHz.
- Library support: Supports FFT functions of bin size 64, 256 and 1024.
- Audio capabilities: Fully supported and microphone implemented.
- development overhead: Choosing this system would mean no overhead, i.e. embedded system is on a PCB board already, and can be bought with an audio interface ready to be used.
- Price: $160 - for embedded system + audio + shipment.

3.3.3. FRDM-K66F

- Model: FRDM-K66F [13].
- Size: 53.5mmx96mm.
- Processing speed: 180MHz.
- Library support: Supports FFT functions of bin size 64, 256 and 1024.
- Audio capabilities: Fully supported and implemented.

Table 3.1: Comparison Table of Embedded Systems

Name	Size	Processing	Lib. supp.	Audio	overhead	Price	Total
PIC32	0	3	2	0	0	1	6
Teensy 3.2	3	1	1	1	1	0	7
FRDM-K66F	2	2	1	1	1	0	7

- development overhead: Choosing this system would mean no overhead, i.e. embedded system is on a PCB board already, and can be bought with an audio interface ready to be used.

- Price: $98 - for embedded system + audio + shipping.

3.3.4. Comparison of Embedded Systems

Table 3.1 contains the values given to each category, and both the Teensy and K66F come out with 7 points. Both are thus fine choices to develop an audio signal analysis solution. For this project we will pick Teensy 3.2 as there is more familiarity with the development environment.

3.4. Audio Analysis on the Embedded System

The grand piano has in total 88 playable notes, from 27.5 Hz to 4186.01 Hz when tuned to equal temperament. This makes a grand piano an excellent testing environment as well as an ambitious goal for the device to try and characterize notes into lights.

3.4.1. Converting Input into Frequency

There are a few ways to get the frequency information needed: i) using fast fourier transform, ii) McAulay-Quatieri (MQ) Analysis, or iii) wavelet transformations [14]. This project will limit itself to using the FFT transformation as the libraries that implement FFT of the ARM CPUs are well known and fast.

3.4.2. Limitations of the FFT on the Embedded System

The embedded system is a Teensy 3.2 with 72 MHz Cortex-M4 processor [12]. The unit comes with a audio library [15] that has an inbuilt FFT based on 1024 samples and 44.1kHz sample rate.

The library code for the FFT transform uses the I2S frequency clock to get the 44.1 kHz signal. This means a resolution of 43.066 Hz. As the lowest note on a grand piano is 27.5, and the step between the lowest 2 notes is $29.135-27.5 = 1.635$ Hz, the FFT step size and lowest note interval are not acceptable. By manually reducing the I2S clock rate we can get a more suitable frequency range..

The highest frequency detectable depends on the sample frequency. The highest note on the piano is 4186.01 Hz, the second harmonic of that note is 8372.02 Hz so to more accurately diagnose that note, we need that frequency to be detectable.

Reconfiguring the library down to a sample rate: 17045.45454 Hz, the resolution will be: $17045.45454/1024 = 16.645951699$ Hz and the highest detectable frequency is 8522.2727 Hz and thus the limit of what the Teensy and its FFT library can do out of the box has been reached and it doesn't fit requirements regarding lower frequency resolution. By using an algorithm there is a way to get to the real peak value, see chapter 4.1.1.

3.5. LED Requirements

For a prototype, it was simplest to purchase online a RGB LED module, see fig. 3.1, ordering 4 LED units plus shipment cost $17.5. Each RGB pin has 151 Ohm in serial. Calculating the current going through with $V = I * R$. Voltage input is 3.2V, so the equations for the current, see eq. 3.1:

$$\begin{aligned} I_{LED1} &: (3.2 - 2.66)/151 = 0.54/151 = 3.58mA \\ I_{LED2} &: (3.2 - 1.92)/151 = 1.28/151 = 8.48mA \\ I_{LED3} &: (3.2 - 2.57)/151 = 0.63/151 = 4.17mA \\ Total &: 16.23mA \end{aligned} \tag{3.1}$$

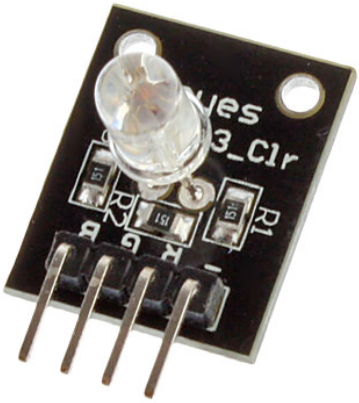

Figure 3.1: RGB LED Package

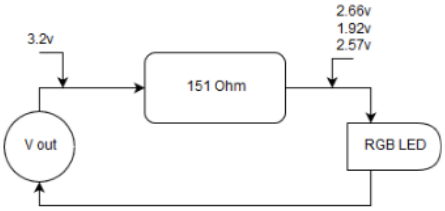

Figure 3.2: Measured Voltages

3.6. Battery Design

Battery design depends on the power needs of the embedded system plus the LEDs. The current that is drawn from the CPU is 30 mA, see [16], and the current from one LED is 16.23 mA. A single 18650 battery has 3.5 Ah, so total time with 4 LEDs and the cpu: 36.87 hours.

The minimum duration should be at least 8 hours, so with a single battery we have a factor of 4.6, which can easily compensate any errors in measurements of current requirements.

Cost of batteries, plus battery charger and shipping is: $25

3.7. Package Design

The packaging itself will be designed to be 3d-printed around the Teensy, a 18650 battery container, a power button, and have room for cabling, a prototype PCB board, and the bluetooth adapter.

The box will be divided into 4 parts that fit together with snap fit joints to simplify access to various parts. The lower case (see fig. 3.3 and 3.4) that contains the LEDs, cabling, embedded system and bluetooth adapter, the sides that snap to the lower case see fig. 3.5, and the upper cover that contains the light diffusing

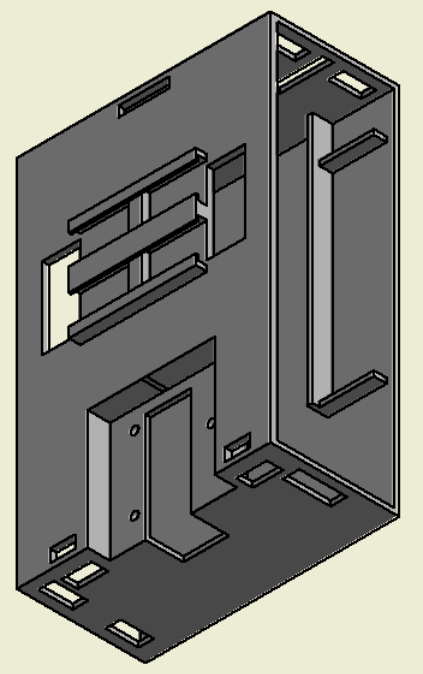

Figure 3.3: Lower Case

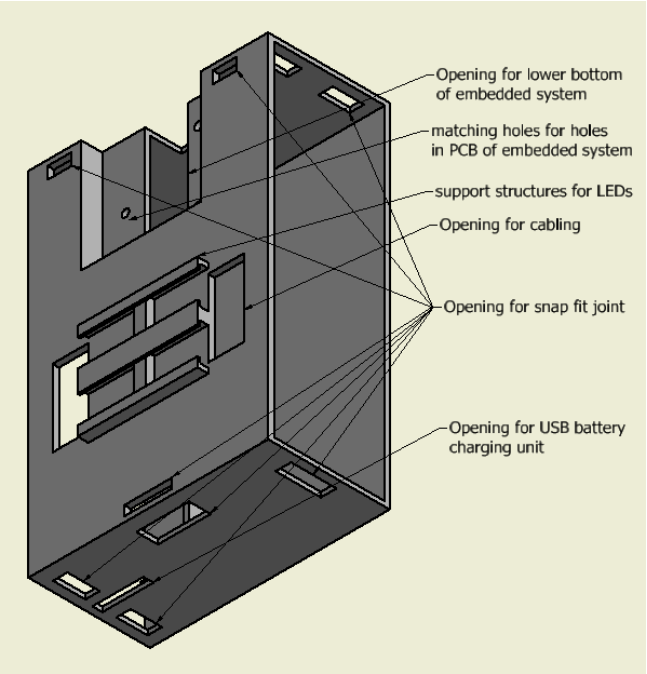

Figure 3.4: Lower Case

dome see fig. 3.6. This separation is so that there is easy access to all components, and simple to snap everything together.

3.7.1. Final price

Price of items plus shipping comes to a total of $209.5 which is acceptable for this prototype.

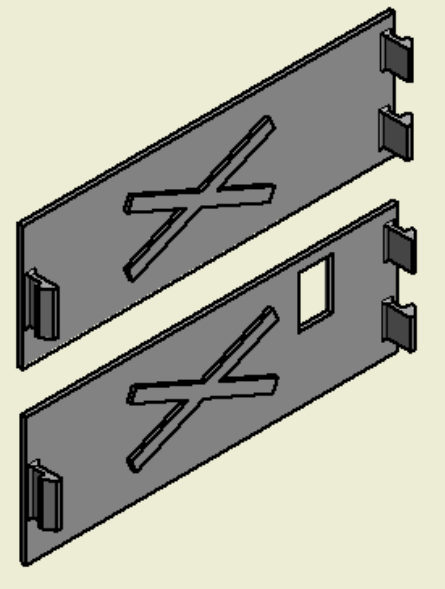

Figure 3.5: Sideplate Orthogonal View

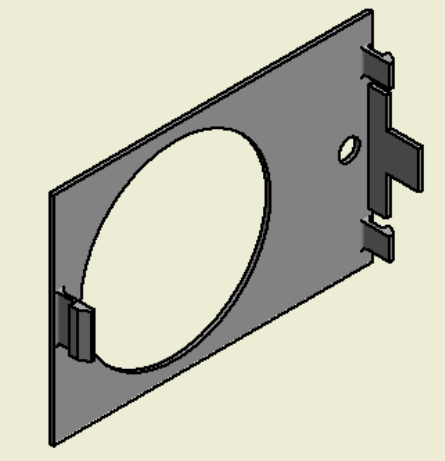

Figure 3.6: Front Cover

4. Algorithm Development

This chapter will go over detailed analysis and the algorithm decisions made to suit that analysis for as optimal diagnosis of notes and chords as possible within the timeframe of this project.

4.1. Audio Detection Design

This section details turning the incoming sound data into the frequencies contained within the signal.

4.1.1. Finding Peaks and Using Centroid Values

Frequency data is gotten from the FFT library function of the microprocessor library. The data contains peak values of the frequencies see fig. 4.1.

Table 4.1 compares 2 ways to find the peak in a pure sinusoidal signal. One way is to measure only the top peak and check the frequency matching the bin number, the other is an algorithm that aggregates each peak together from the lowest point before and lowest point after the peak and uses equation 4.1 to estimate the centroid value [17].

$$\text{Centroid of x} = \frac{\sum \mathbf{x} * \mathbf{f}(\mathbf{x})}{\sum \mathbf{f}(\mathbf{x})} \tag{4.1}$$

Chosen were the most difficult peaks to find, that represents a signal that is between 2 bins, thus in between the resolution of the original signal.

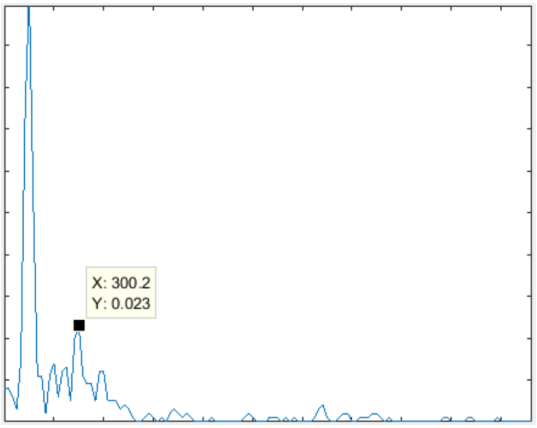

Figure 4.1: An Example of Peak Data from the FFT Function

Table 4.1: Comparing 2 Ways to Find the Frequency

Freq. (Hz)	**Top Hz**	**Cent Error**	**Centroid Hz**	**Cent Error**
110	116.7	102.36	110.1236	1.9442
82.4069	83.39	20.531	82.4924	1.7953
55	50.04	-163.62	56.3619	42.3464
41.2034	33.36	-365.57	40.4454	-32.145
30.8677	33.36	143.42	29.3612	-86.624
29.1352	33.36	234.42	30.4197	74.69
27.5	33.36	334.42	30.5684	183.13

Table 4.2: Comparing Window Types of 10 Consecutive Detections of Freq. 36.7081

Window type	Min. Hz	Max Hz	Av. Hz	Extra notes	no notes
Hanning	36.08	40.16	38.22	1	0
Bartlett	37.70	40.20	38.91	2	0
Blackman	34.79	40.12	38.58	0	0
Flattop	33.73	67.94	49.45	0	3
BlackmanHarris	34.22	40.12	37.83	0	0
Nuttall	34.10	39.96	37.89	0	0
BlackmanNuttall	33.74	40.47	37.57	0	0
Welch	36.25	41.20	39.03	4	0
Hamming	33.49	40.66	38.48	7	0
Cosine	36.27	41.37	38.80	5	0
Tukey	38.45	43.62	41.71	0	0

4.1.2. Analyzing Window Type

Table 4.2 shows a comparison made between the different window types available with the FFT library, using a sinusoidal signal of frequency 36.7081 Hz. Taking 10 consecutive samples, and checking the minimum, maximum and average frequency detected, as well as noticing whether any extra harmonics were detected, or if any of the samples were empty.

None of the window types got the average exactly right, some were adding extra notes, or had no notes. The average detected frequency of the BlackmanNuttall window type came closest to the real note.

4.2. Audio Analysis

This section will go over the analysis of the centroid Hz values found in the FFT data in regards to tracking the peak from one time instance to the next.

Given a multitude of factors like the design of an instrument, temperature, the way an artist plays an instrument (and many, many more) the frequency of a note that emanates from a musical instrument is never completely true. This means that a frequency found in one instance will almost never be exactly the same in

the next instance although it is indeed the same note being played.

This is solved by finding the closest unique Hz in the last peak detection, averaging between them and finding a centroid for the Hz value based on top peak value of the current value, the older value and the running average, see formula 4.2.

$$\text{Centroid of x} = \frac{\sum \mathbf{x} * \mathbf{f}(\mathbf{x}) + \mathbf{y} * \mathbf{f}(\mathbf{y}) + \mathbf{z} * \mathbf{f}(\mathbf{z})}{\sum \mathbf{f}(\mathbf{x}) * \mathbf{f}(\mathbf{y}) * \mathbf{f}(\mathbf{z})} \tag{4.2}$$

- x = current Hz value
- f(x) = current Hz peak value
- z = old Hz value
- f(z) = old Hz peak value
- y = average Hz value
- f(y) = average peak value

Each peak is analyzed in frequency order, this means the current peak being analyzed could have gotten the same result as the peak analyzed one cycle of the loop earlier, so a comparison must be made, which peak is closer to the older value: The current peak being analyzed, or the earlier peak. This is done in the frequency domain. If the earlier frequency was further away, as it has already been analyzed with its own earlier peak, the earlier peak must be a new peak not represented in the old data, and should be marked as a new peak.

Data gathered from the older peak:

- Lowest historical frequency value
- Highest historical frequency value
- Top peak value
- Frequency value
- Counter

- Top value average
- Frequency value average
- Frequency Centroid average
- Historical top peak max value

Care must also be taken regarding how much a peak can increase/decrease in Hz from the old closest peak, as there must be a limit beyond which the closest peak really isn't the old peak, but a new peak. This can be done by checking the historical variance of the peak, see eq. 4.3.

$$\text{Sigma} = \frac{\text{highest historical frequency value - lowest historical frequency value}}{\text{current frequency value}} \tag{4.3}$$

Given some boundary value at which the current frequency is a new value rather than a continued sound. At this moment the boundary value is arbitrarily set at a fixed value: 0.25 of the current note, which in relative terms spans a little over 2 notes on each side of the current Hz.

A new running average of both the frequency value is then calculated and current peak is either higher than highest historical peak, or lower than the lowest.

4.2.1. Noise Floor Data Acquisition

To recognize the difference between a real signal and noise, it is good to analyze the frequencies found specifically for noise data. Noise cannot be defined as unwanted sounds, as that is too generic and is based on how humans can hear and differentiate between sounds, instead noise is here defined as a constant peak value at a certain frequency range (i.e. a low constant hum).

An array which represents the 88 different keys on the keyboard is used as a template and stores the average value of noise for that range. The limitations are the same as for finding notes, values below 27.5 Hz go into the 0-to-27.5 Hz bracket, and notes higher than 4186.01 go into the 4186.01-or-higher bracket, and

the repercussions of that are that it might take a higher value than otherwise at these 2 extremes to detect a real signal.

Calculating the average value and the standard deviation, see eq. 4.4 and 4.5, a noise threshhold can be calculated and used later to filter out noise [17].

$$\text{new average} = \frac{\mathbf{Av_{old}} + (\mathbf{value} - \mathbf{Av_{old}})}{\mathbf{N}} \tag{4.4}$$

$$\text{new sigma} = \sqrt{\frac{(\mathbf{N-2}) * \mathbf{Sigma^2_{old}} + (\mathbf{value} - \mathbf{Av_{new}}) * (\mathbf{value} - \mathbf{Av_{old}})}{\mathbf{N-1}}} \tag{4.5}$$

4.3. Audio Filtering

This section explains the methods used to exclude or filter peaks so what is left are the notes from the instrument. Figure 4.2 and fig. 4.3 show 100 audio samples side by side where the signal has had no filtering, in fig. 4.3 each dot is a peak.

There are numerous ways to attack the problem on how to filter the signals to find real notes:

- Sort peaks by FFT value, the highest signal is the most likely real signal.
- Establish a noise floor based on average peak data and its standard deviation value, anything above that floor must be a likely real signal.
- Establish a continuity factor based on how many times a peak has been found in the data, anything that has been detected for more than x samples must be a likely real signal.
- Aggregate together frequency harmonics, anything that displays clear multiples of a frequency rather than just random frequencies must be a likely real signal.

Each item in this list has a potential problem with how it filters the signal.

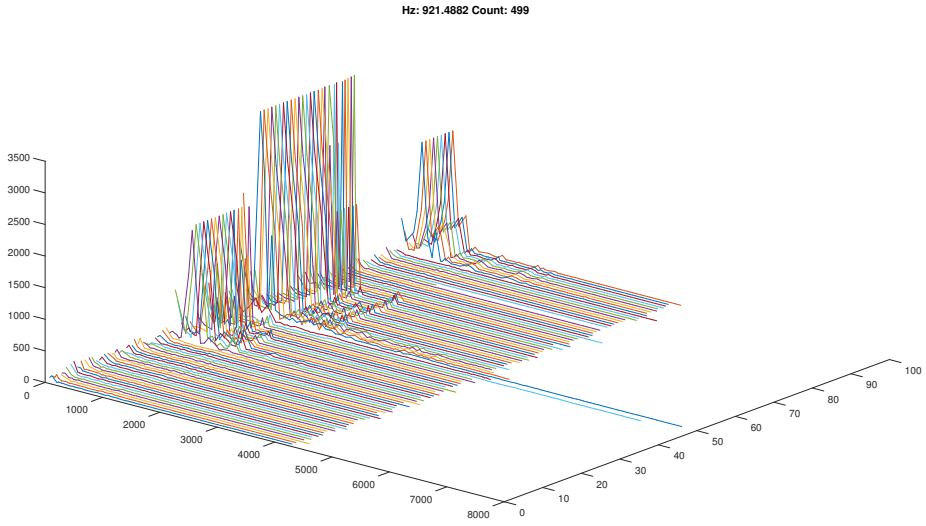

Figure 4.2: 100 Continuous Samples, Each Line is One Sampled Instance

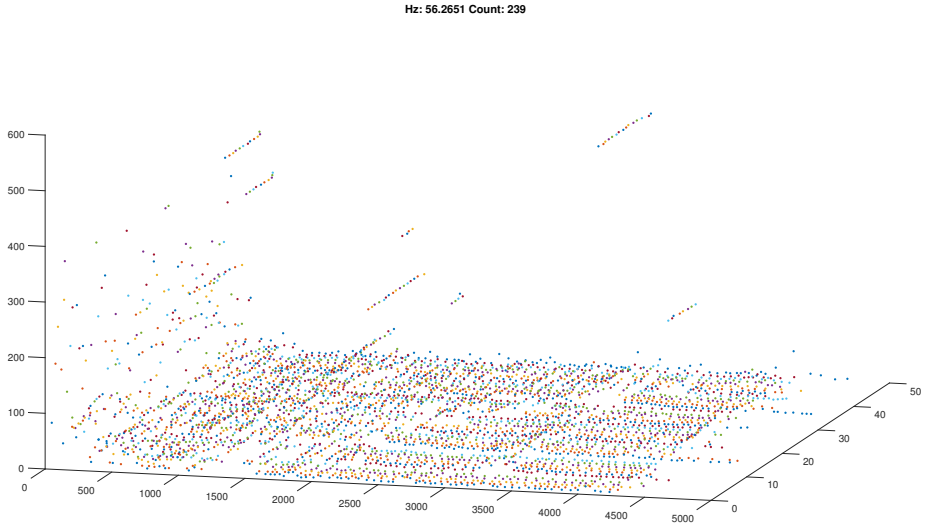

Figure 4.3: Each Dot is a Peak Value

- Sorting peaks by FFT data value only means an instrument that has strange harmonics where the note being played isn't the strongest detected frequency will always detect the wrong note.

- Filtering by removing peaks below a certain noise floor means a real signal might very well be filtered out.

- A continuity factor is useless if a note isn't accurately diagnosed as the same note from one instance to another, either by setting the old note to a new one, or allowing another new note to take over the old data.

- Aggregating frequency harmonics might remove real notes just because they incidentally match up, this can happen with C4 (261.626 Hz) and G5 (783.991 Hz) but the third multiple of C4 is 784.878 Hz which is within 0.114% difference.

A fixed continuity factor is used in the filter, a simple comparison with the number of times the peak has been counted in the data, as well as a noise-floor filter and a harmonic filter detailed below.

4.3.1. Noise Floor Filtering

Filtering the data is based on using formula 4.6. Sigma is the standard deviation of the average value, and normally 99.8% of data will be within the 3 lengths of the standard deviation (see table 3, page 443 of [17]). Measuring only the ambient sound, the average reduction of the number of peaks for a sigma of 3 was from 93.8 to 0.122 which is a 99.87% reduction. With a sigma of 4 it was a reduction of 92.8 to 0.213 which is a 99.77% reduction, the only explanation why *4 is less than *3 are incidental ambient sounds generated while measuring.

Given that a real signal can fall below the noise floor, instead of comparing the measured signal, we compare the historical top peak max value with the noise floor, this means that the peak is visible as long as it is continuously detected.

$$\text{noise limit} = \lceil \mathbf{averagevalue} \rceil + \lceil \mathbf{sigma} * \mathbf{3} \rceil \tag{4.6}$$

4.3.2. Harmonic Filtering

An instrument sounding a note can display harmonic bands both above and below the base note. 2 simplistic filters based on removing these multiples were developed, both use an array with its peaks sorted into order of the peak with the largest historical top peak value first, and the lowest peak last, and all multiplications of the earlier peak are removed, for filter 1 see eq. 4.7 and for filter 2 see 4.8.

Filter 1

$$\begin{aligned} ratio &= \frac{\text{higher frequency}}{\text{lower frequency}} \\ factor &= \text{round ratio to nearest integer} \\ difference &= \frac{ratio}{factor} \\ removeif &: lowerboundary < difference < upperboundary \end{aligned} \tag{4.7}$$

The difference in eq. 4.7 is compared to an upper and lower value, which is fixed as half a semitone above or below the multiplication of the note.

It would be better that the upper and lower boundaries would not be fixed, one hypothesis is that the absolutely lowest frequency in the sample should fit all the higher frequencies, this means finding the difference between the lowest frequency and all the higher ones, and tweaking the upper- and lower boundaries in accordance to that.

Filter 2

$$freq.difference = \frac{|\text{higher value - lower value}|}{\text{base frequency}} \tag{4.8}$$

remove if: freq. difference is 0.1 from an integer value or less.

The base frequency is the first value of the array and it compares the difference between all frequencies relative to itself. Filter 2 removes the lesser value if freq. difference is 0.1 from an integer value or less.

4.4. Tonal Analysis

This section is a description of finding the notes within the filtered data.

1) If the user has enabled the analysis of single tones, the frequency with the highest peak is selected.

2) If the user has enabled 2 note intervals, the 2 highest frequencies after filtering are compared based on their relative note values (i.e. 440 Hz has a relative note value of A and a fixed value of A_4). This has higher priority than 1).

If the user has enabled triads, the 3 highest frequencies are compared based on relative note value. This has higher priority than 1) and 2).

If the user has enabled sevenths, the 4 highest frequencies are compared based on relative note value. This has higher priority than 1), 2) and 3).

The device keeps arrays of the colours associated with each mode of interpretation which get updated when the user selects a different colour in the app.

Based on this data, a colour is selected from the proper array and pushed to the LED lights.

5. Prototype and Application Development

5.1. Prototype Development

In this section a development of a prototype will be illustrated. Here is a list of what the prototype contains which was purchased:

- 1x Teensy 3.2 embedded system with an accompanying audioshield.
- 1x 18650 battery with 3500 mAh capacity.
- 1x TP4056 battery Charging Holder Charging Board.
- 4x RGB 3-Color LED Module.
- 1x HC-06 Wireless Bluetooth Transceiver Radio Frequency (RF) Main Module Serial.
- Off/on switch.
- Diffuser cover from an LED bulb purchased from Ikea; name: "Ledare LED 1800 lm".
- Prototyping PCB board 5cmx7cm.
- 25 jumper wires.

As is detailed in Chapter 3, a package was designed and 3D printed to contain the contents, also a small prototyping PCB board is used to share ground for the Teensy, LEDs and bluetooth module, as well as split the RGB signal from the

Teensy to the LEDs.

Pin connections of the Teensy, see appendix figure A.1

- Ground: connected to shared ground.
- Vin: connected to an on/off switch, and from it to a battery.
- 3.3 V output pin: connected to 3.3 V input on the bluetooth module.
- Pin 0: Receiving pin, receives what the bluetooth module transmits.
- Pin 1: transmitting pin, sends commands to the bluetooth module.
- Pin 6: Analog output signal for the Red value.
- Pin 21: Analog output signal for the Blue value.
- Pin 20: Analog output signal for the Green value.

For design of the package, see chapter. 3.7.

5.2. Application Development

In this section an outline of a basic app will be illustrated.

The app was developed in Android Studio, a development environment specifically created for the android operating system which utilizes Java programming language [1].

Each screen is defined as an "activity" in the android OS. The first screen is designed to check if bluetooth is enabled, and connect to the device, see fig. 5.1.

The next screen shows a list containing the different modes of interpretations of music, i.e. single notes, 2 note intervals, triads and sevenths, see fig. 5.2.

[1] `https://developer.android.com/studio/index.html?gclid=COil96Cq99MCFc687QodxekNaA`

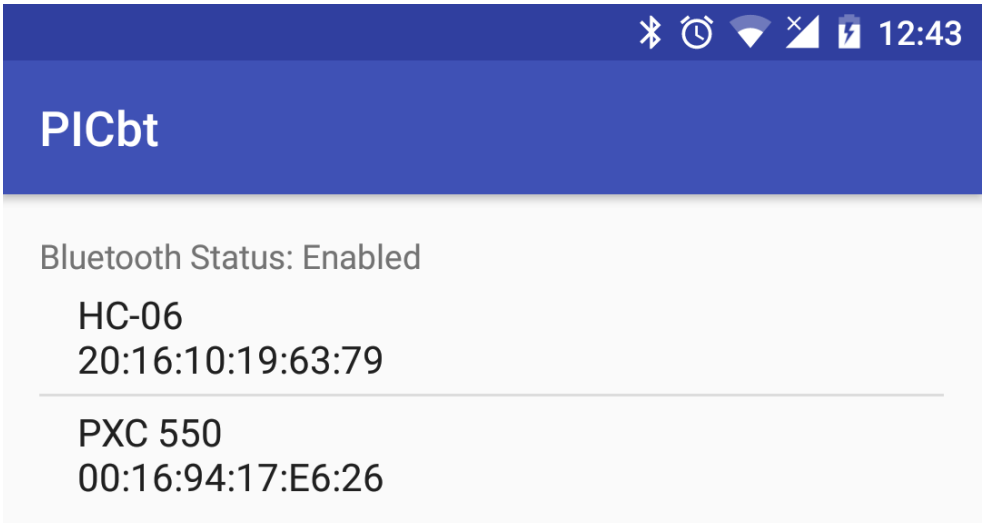

Figure 5.1: The First Screen in the Application

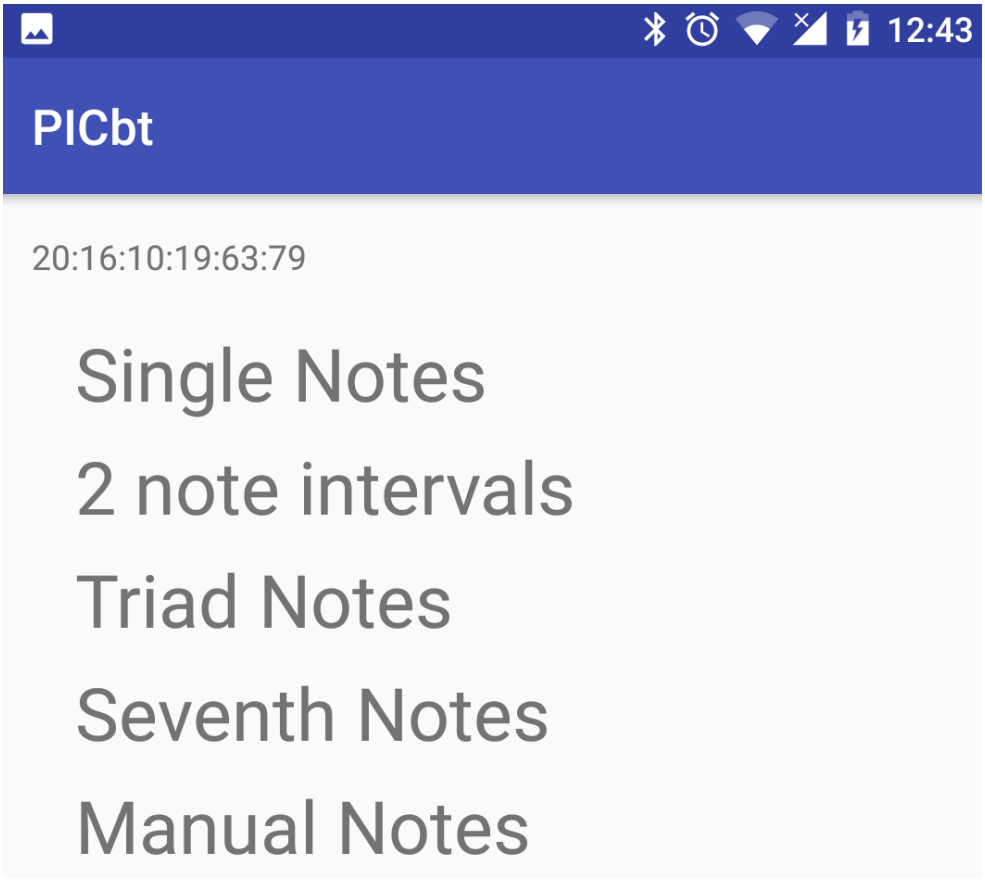

Figure 5.2: The Second Screen in the Application

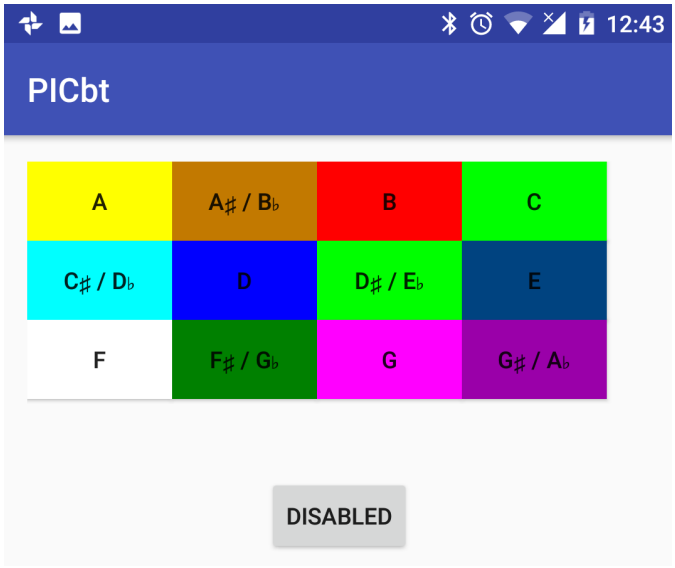

Figure 5.3: Application Screen 3, Single Notes

Selecting an option will take the user to a third screen which lists the colours and whether the mode of interpretation is enabled. All the modes can work together, or individually, see fig. 5.3, 5.4, 5.5 and 5.6.

Each note, or each chord has its own button that shows the colour of that note or chord as its background. clicking that button opens up a simple colour picker that when the colour has been selected, sends the colour to the device via bluetooth, and after getting verification back, saves the color in a database

The color picker code is a library from Github [2].

5.2.1. Communication Between App and Device

The bluetooth is connected to the serial communication pins on the Teensy, and the commands and settings are sent via ascii character strings. Here are some of the commands and responses:

- Command from app: **save,1,1,#FFFF00**> - no callback from device.

[2] `https://github.com/ItsPriyesh/chroma?utm_source=android-arsenal.com&utm_medium=referral&utm_campaign=3339`

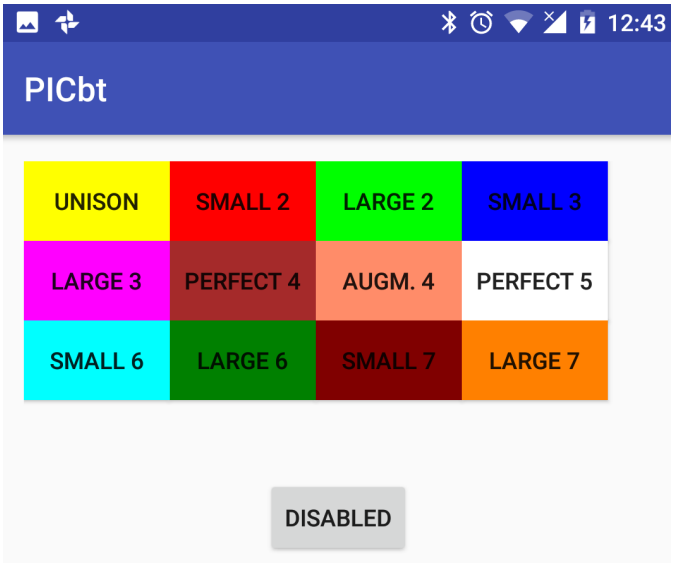

Figure 5.4: Application Screen 3, 2 Note Intervals

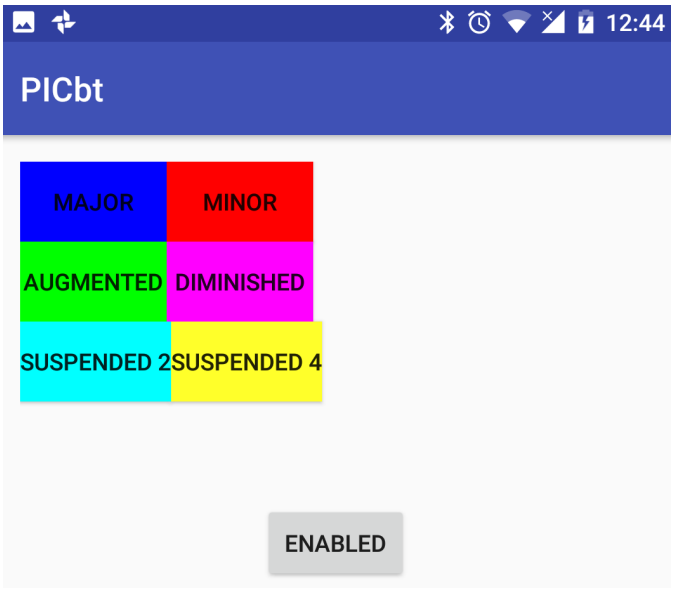

Figure 5.5: Application Screen 3, Triad Notes

Figure 5.6: Application Screen 3, Seventh Notes

- Command from app: <**get,1**> - callback: <**get1,1**>
- Command from app: **set1,1**> - callback: <**get1,1**>

There are 4 get and set commands for each of the boolean that tells the Teensy which mode of interpretation is enabled. Both colours and boolean values are saved to flash memory.

6. Results and Discussion

This chapter will detail results from tests made to gauge the validity of the analysis and filters in the algorithm and discuss the interpretation.

6.0.1. Test Environment

This test is done in an environment with a low hum from a computer and with some low random environment sounds. an electric piano "Kawai CN 390" is used for playing piano notes as well as the speakers for playing sine waves from a computer.

6.0.2. Frequency Accuracy

Table 6.1 shows a frequency accuracy results playing a sine wave of a single frequency within the range of 27.5 Hz to 4186.01 Hz. 100 continuous samples for each frequency was analyzed, starting from when the frequency is first detected.

Cent error is the best predictor of how incorrect the frequency is from the generated frequency as it is based on a multiple of the frequency being measured.

6.0.3. Noise Filter Reduction

Table 6.2 shows the result of noise filter measurements. The device was turned on, and number of filtered and unfiltered peaks were measured in a noisy environment, with 2 coolers, and many computers. 155 seconds were logged, results are split into 5 second intervals.

The constant value of unfiltered peaks vs. the diminishing value of peaks shows

Table 6.1: Frequency Accuracy Analysis from a Generated Sine Wave, Average of 100 Continuous Samples

Frequency	min	max	average	variance	cent error
27.5000	22.6469	47.6304	32.1986	47.8938	273.0796
29.1352	19.2014	55.5233	31.4925	48.7621	134.6939
30.8677	8.2108	50.1816	33.9193	8.3542	163.2101
32.7032	28.7175	46.8058	33.5532	2.8290	44.4204
34.6478	33.0968	44.4107	37.3432	2.9294	37.3432
36.7081	33.7062	63.8598	39.2375	3.2091	115.3618
38.8909	33.9736	45.0410	40.3579	1.2732	64.1022
55.0000	54.6205	57.9350	56.1689	0.5630	36.4079
58.2705	57.5478	63.5147	59.7509	0.9173	43.4337
61.7354	51.6876	63.8484	61.9248	1.4375	5.3033
65.4064	62.5405	67.0871	65.5766	0.9855	4.4979
77.7817	76.9849	90.6661	77.9011	1.4244	2.6548
110.000	108.3689	110.4025	110.0477	0.2061	0.7506
146.832	131.2428	147.3518	146.7266	1.6899	-1.2432
220.000	219.0132	221.9854	220.5428	0.6149	4.2662
293.665	292.3867	306.9396	294.4801	1.5348	4.7986
311.127	311.1915	313.3019	311.6469	0.2839	2.8905
329.628	320.1426	373.1899	330.7865	4.6448	6.0739
349.228	349.0957	354.1508	349.9945	0.6062	3.7956
369.994	367.0838	371.8387	370.7144	0.4622	3.3675
391.995	392.4543	394.7428	392.7923	0.2236	3.5177
415.305	409.6593	419.6949	416.1680	0.9427	3.5938
440	439.2155	442.2005	440.7003	0.1786	2.7532
622.254	612.1679	625.8228	623.1183	1.5656	2.4030
880	878.7475	891.8143	882.0464	1.2374	4.0213
1760	1745.4	1764.8	1763.4	1.9101	3.3076
3520	3320.4	3527.5	3524.5	20.5739	2.2089
4186.01	4189.9	4197.4	4194.2	1.12745	3.3897

that the filter is working.

6.0.4. Harmonic Removals

Table 6.3 shows measurements from the 2 ways to filter based on harmonics specified in chapter 4.3.2.

Filter 1 is clearly the better filter with less extraneous notes and diagnoses the chord better.

6.0.5. Code Speed Evaluation

Table 6.4 evaluates the speed of the code and whether the hardware is a limiting factor.

The FFT will theoretically get 33 results pre second which means a new result will be ready in 30.3 milliseconds and have that time to process the signal until the next sample is ready.

The algorithm takes less time than the time it takes to gather FFT data, so is successful in analyzing each sample fully.

Table 6.2: Noise Filtering Measurement

time interval	Filtered	unfiltered
average	3,23	90,00
5	2,44	65,48
10	3,57	80,78
15	4,99	86,85
20	10,79	89,28
25	11,88	91,98
30	8,61	91,93
35	7,36	93,66
40	6,04	94,60
45	3,71	95,13
50	1,62	92,76
55	1,37	92,38
60	3,25	93,81
65	1,55	93,01
70	1,22	92,93
75	1,12	93,02
80	0,23	85,21
85	0,00	82,87
90	1,49	83,92
95	1,16	89,42
100	6,71	95,25
105	5,77	93,84
110	1,72	91,86
115	3,09	94,56
120	0,95	92,34
125	2,78	94,36
130	1,35	91,92
135	0,73	92,71
140	1,90	92,32
145	2,47	90,88
150	0,14	85,30
155	0,04	85,57

Table 6.3: Harmonic Filter Comparison Based on A Major Chord With Different Base Notes

Filter	Base note	total : counted	ratio	ave no. of notes
1	880 Hz	51 : 41	0.805	3.4902
2	880 Hz	51 : 29	0.569	4.1765
1	440 Hz	10 : 7	0.7	1.9189
2	440 Hz	10 : 6	0.6	1.9369
1	220 Hz	35 : 35	1	4.9
2	220 Hz	12 : 9	0.75	4.75

Table 6.4: Time in Milliseconds on Average to Process the Algorithm

condition	min	max	average
background noise, no music	9	15	9.65
white noise, no music	27	38	33.73
random chords	7	19	16.76

6.1. Discussion

Building a system which can characterize the notes being played was the objective of this project, and given those parameters the project is a success. However, this section will discuss the limits to the design of the device and algorithm, and a hypothesis of how it can be combated will be layed out.

6.1.1. Limitations of the Algorithm

The centroid calculations in analyzing and finding the peaks in the FFT data was very successful in getting accurate frequency results, even better than I expected. But even with the more accurate way to calculate the true frequency of the note, there was a clear increasing error margin for the lower notes on the piano at the 60Hz margin and below, see table 6.1. Given that the accuracy was fine above 60Hz, the possible reasons could be inaccuracy of the note in the piano, or that the microphone is not sensitive enough at these lower frequencies.

The noise filter combined with finding the note from the older sample and comparing the top historical peak with the noise floor was very successful in keeping a real signal alive above the background noise. The limitation is the accuracy of

finding and deciding whether a note is really the same note as in the old sample, and currently there is a fixed value deciding how far from the old sample the new note can be, which is not ideal.

The harmonics filter worked surprisingly well, but it is a 2-tier loop, and as such there can be a huge difference between calculating for 20 peaks (400 loops), or 200 peaks (40 000 loops). Here is where hardware limitations of the Teensy shows itself, but as it is still above 30 samples a second, and more than 20 peaks happens only when white noise is being played, it is acceptable.

There is another limitation of the analysis. If a single note being played does not have its lowest frequency as the strongest note, an incorrect analysis of the real note can be made, as the lower valued frequencies are removed as multiples of the higher valued ones. The only true way to combat this is using neural networking and deep learning through user input to create a 'fingerprint' of each note. Then any strangeness of any instruments' harmonics would be completely accounted for. This would replace the harmonic 2-loop filtering done.

Another limitation is that sound that isn't noise, and isn't the instrument cannot be detected and analyzed as extraneous sounds which shouldn't be used in the conversion to a colour. The current filter makes sure that any peak is detected at least 4 concurrent times and that way the signal is somewhat stable. This could be enhanced in the future with a differential value of the standard deviation of the frequency, a lessening standard deviation differential would mean a more accurate signal. However the best solution would be the neural network 'fingerprinting' mentioned above, because fingerprinting the notes of the instrument you are interested in can be used to develop an algorithm to ignore any sounds that does not fit.

The design of the container was very useful for a prototype. Having snap-fit sides and top meant easy access to the insides, and the Teensy itself could be hidden away. Two of the smaller snaps on the sides snapped off, so I would look into improving the 3D printing process, making sure the snaps themselves would be printed with more material.

7. Conclusion and Future Work

The main objective was to develop a device that could characterize the notes and chords being played by a musical instrument which was successful within the parameters.

Embedded systems were analyzed and put into a selection matrix where the Teensy 3.2 development board was selected most suitable. Other components were detailed and a prototype was designed and successfully built that ran under its own battery power.

The prototype was then tested and the notes and chords being played were converted to user selected coloured light.

The algorithm was tested on detection, noise filtering, harmonic filtering and speed, and was found to give accurate results regarding frequency detection, allowed only active sounds through the noise filter, and was able to filter most harmonic frequencies successfully.

The speed of the algorithm was good for diagnosing a piano, and even at white-noise levels, detection was high enough to be sufficient.

7.1. Future Work

Future work could include:

- Creating a neural network algorithm is the next step in regards to harmonic filtering and incidental sound exclusion.
- Research into analyzing lower notes with more accuracy.

- Extending analysis to be able to analyze chord progressions.
- Analyzing microphone sound quality and its effects.
- Create a desktop app for projector usage.
- Extending app to be able to record to midi files.
- Extending app to be able to interface with lighting interfaces.

References

[1] J. Pearce, "Synaesthesia", *European Neurology*, vol. 57, no. 2, pp. 120–4, 2007. [Online]. Available: https://search.proquest.com/docview/194949082?accountid=135940.

[2] A. J. R. Simpson, G. Roma, and M. D. Plumbley, "Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network", *CoRR*, vol. abs/1504.04658, 2015. [Online]. Available: http://arxiv.org/abs/1504.04658.

[3] H. Þ. Birgisson. (2012). Musical tone recognition system for interval tone training on mobile devices, [Online]. Available: http://hdl.handle.net/1946/13278.

[4] R. E. Cytowic and D. M. Eagleman, *Wednesday is indigo blue: Discovering the brain of synesthesia*. The MIT Press, 2009, ISBN: 0262012790. [Online]. Available: https://goo.gl/x5yO2N.

[5] E. Brigham, *Fast fourier transform and its applications*. Pearson, 1988, ISBN: 0133075052. [Online]. Available: goo.gl/pmwrez.

[6] *Cmsis dsp software library*, ARM Limited [Online].

[7] H. E. White and D. H. White, *Physics and music: The science of musical sound (dover books on physics)*. Dover Publications, 2014, ISBN: 0486779343. [Online]. Available: https://goo.gl/7s9Wbl.

[8] D. Huynh, *Music equal temperament*. lulu.com, 2012, ISBN: 1300273321. [Online]. Available: https://goo.gl/CqcItU.

[9] "Acoustics – standard tuning frequency (standard musical pitch)", International Organization for Standardization, Geneva, CH, Standard, Jan. 1975.

[10] A. Schoenberg, *Theory of harmony: 100th anniversary edition*. University of California Press, 2010, ISBN: 0520266080. [Online]. Available: https://goo.gl/zbqkAr.

[11] (2017). Pic32mz2048efh064 - microcontrollers and processors, [Online]. Available: http://www.microchip.com/wwwproducts/en/PIC32MZ2048EFH064.

[12] (2017). Teensy 3.2 & 3.1, [Online]. Available: https://www.pjrc.com/teensy/teensy31.html#specs.

[13] (2017). Frdm-k66f|freedom development platform|kinetis® mcus|nxp, [Online]. Available: https://goo.gl/povWZQ.

[14] P. Burk, L. Polansky, Repetto, Roberts, and Rockmore. (). Music and computers, [Online]. Available: http://cmc.music.columbia.edu/musicandcomputers/chapter3/03_06.php.

[15] P. Stoffregen. (1999). Audio library, [Online]. Available: https://github.com/PaulStoffregen/Audio.

[16] *K20p64m72sf1*, Freescale Semiconductor. [Online]. Available: http://freerangefactory.org/pdf/K20P64M72SF1.pdf.

[17] R. A. Adams, *Calculus complete course*. Addison-Wesley Pub (Sd), 1994, ISBN: 0201828235. [Online]. Available: https://goo.gl/ZReObT.

A. Appendix

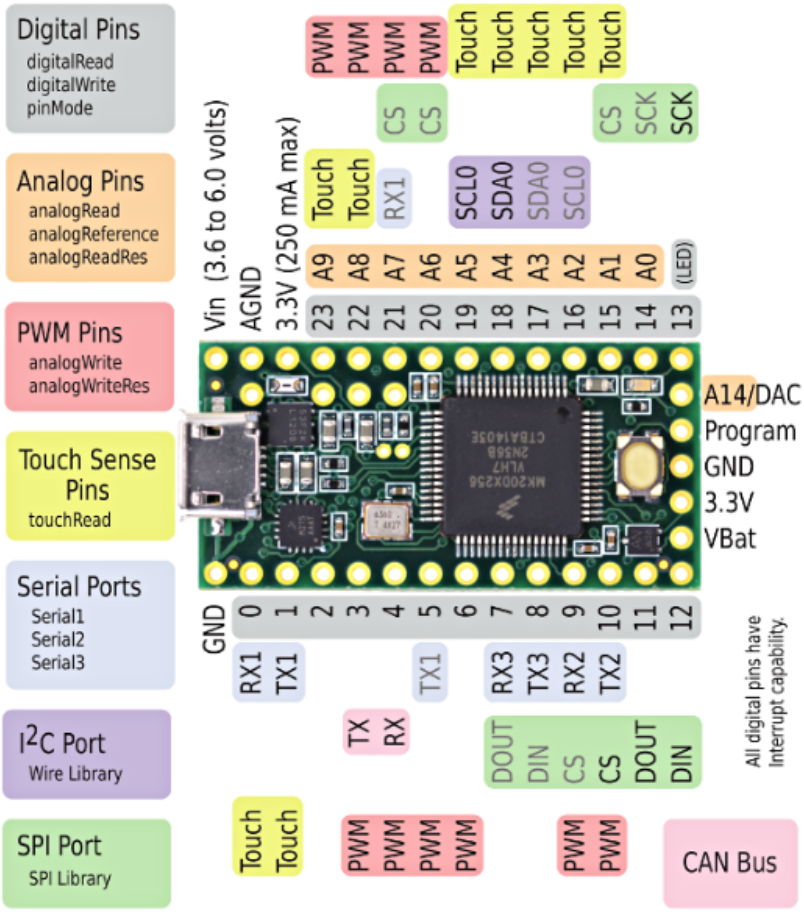

Figure A.1: Teensy Datasheet

Table A.1: List of Piano Notes and their Frequencies

Note	Freq. (Hz)	Note	Freq. (Hz)	Note	Freq. (Hz)
A0	27.maí	D#3	155,5634919	A5	880
A#0	29,13523509	E3	164,8137785	A#5	932,327523
B0	30,86770633	F3	174,6141157	B5	987,7666025
C1	32,70319566	F#3	184,9972114	C6	1046,502261
C#1	34,64782887	G3	195,997718	C#6	1108,730524
D1	36,70809599	G#3	207,6523488	D6	1174,659072
D#1	38,89087297	A3	220	D#6	1244,507935
E1	41,20344461	A#3	233,0818808	E6	1318,510228
F1	43,65352893	B3	246,9416506	F6	1396,912926
F#1	46,24930284	C4	261,6255653	F#6	1479,977691
G1	48,9994295	C#4	277,182631	G6	1567,981744
G#1	51,9130872	D4	293,6647679	G#6	1661,21879
A1	55	D#4	311,1269837	A6	1760
A#1	58,27047019	E4	329,6275569	A#6	1864,655046
B1	61,73541266	F4	349,2282314	B6	1975,533205
C2	65,40639133	F#4	369,9944227	C7	2093,004522
C#2	69,29565774	G4	391,995436	C#7	2217,461048
D2	73,41619198	G#4	415,3046976	D7	2349,318143
D#2	77,78174593	A4	440	D#7	2489,01587
E2	82,40688923	A#4	466,1637615	E7	2637,020455
F2	87,30705786	B4	493,8833013	F7	2793,825851
F#2	92,49860568	C5	523,2511306	F#7	2959,955382
G2	97,998859	C#5	554,365262	G7	3135,963488
G#2	103,8261744	D5	587,3295358	G#7	3322,437581
A2	110	D#5	622,2539674	A7	3520
A#2	116,5409404	E5	659,2551138	A#7	3729,310092
B2	123,4708253	F5	698,4564629	B7	3951,06641
C3	130,8127827	F#5	739,9888454	C8	4186,009045
C#3	138,5913155	G5	783,990872		
D3	146,832384	G#5	830,6093952		

Table A.2: Time Plan for the Thesis

Part	Estimated time*
1) diagnosing options for embedded systems and microphones	2-3 weeks
2) sound analysis and measurements	3-4 weeks
3) programming the sound analysis	10 weeks
4) design and production of PCB	2-3 weeks
5) design and production of packaging	2-3 weeks
6) programming the user interface	5 weeks
7) writing the report	10 weeks

*Estimated time is based on each week having at least 2 items in the list are being worked on at the same time, see fig. A.2.

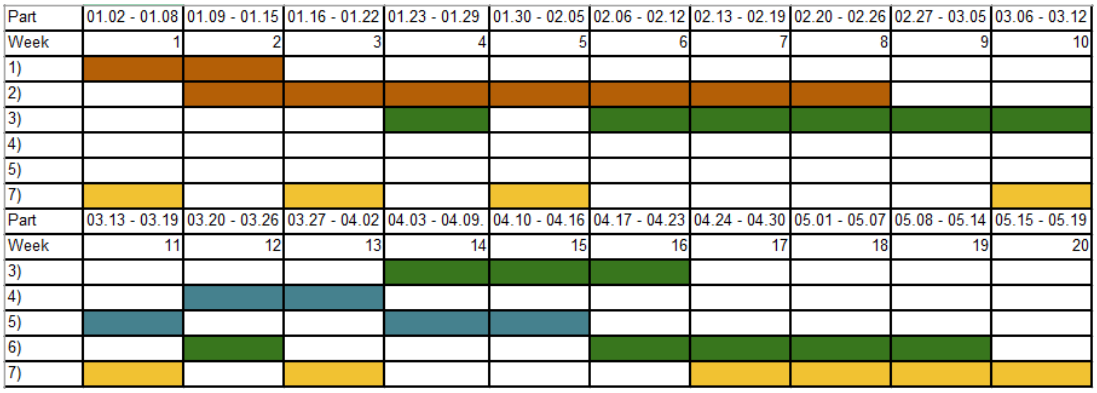

Figure A.2: Time Plan for the Thesis