Stand-Alone Device for Chord Detection

Timothy Palace

CpE 499: Capstone Design Project

Advisor: Palmyra Catravas

March 17, 2015

Abstract

Music is an art form that has existed in societies for thousands of years. Many people who are interested in further appreciating music beyond just listening to it, study the subject of music theory. From a technical stand point, much of music theory focuses on analyzing and understanding the harmonic structures of compositions through transcription and musical analysis. The most fundamental harmonic structure is a chord. My goal was to create a device that is as portable as a guitar tuner that is able to perform chord detection, creating a very useful tool for musicians. Using a Raspberry Pi Model B+ for data acquisition, a USB-Soundcard for receiving acoustic input, and a LCD screen for displaying the output, I was able to create a stand alone device that is able to accurately detect the 12 basic major and minor triads as well as the 12-notes in the western 12-tone music system

Table of Contents

Table of Figures and Tables

1. Introduction

Ever since I started playing the violin when I was three years old, both listening to and playing music as been one of my strongest passions. The study of music theory is something that I have pursued since senior year of high school, but even after years of study and practice tasks such as musical analysis and transcription, which allow for musicians to more deeply appreciate other artists' compositions still prove to be tedious and time consuming tasks. Both of these tasks require the knowledge of what chords are underlying in the piece. One of my favorite activities to do is get together with a group of my friends and make music through improvisation; knowing the chords that are being played makes the process of improvising with other people exponentially easier. The creation of a stand-alone device that performs chord detection from an auditory input would act as a very useful tool, for not only myself, but also other musicians. The heart of the motivation for this project is to create a useful tool for musicians which can be used by anyone, anywhere. Essentially creating a device that is small enough to fit in a musician's instrument case that is able to perform chord recognition.

The following report will outline the background research, the design requirements, the design alternatives, the initial proposed design, the final implementation, as well as the results and conclusions that came about from this project.

2. Background

The music industry is one that has been booming for the last century. From everything from the manufacturing of instruments and equipment to marketing of albums and concerts of artists. Although making a career in the industry is hard, much of the marketable aspects of the music industry are for those who enjoy playing and listening to music as a hobby. This is because music is something that has been integrated into the everyday modern life, whether it be directly or indirectly for example through a TV-show, website, or even standing on an elevator.

Instrument tuners such as tuning forks have existed throughout history almost as long as written music itself has existed. In the mid 1900s an electric tuner was invented which became a useful tool for musicians everywhere.[1] Instrument tuners today are devices that the many musicians rely on and carry with them because they are devices which are portable and allow for instruments to be tuned in a matter of minutes without having to rely on ones hearing. Currently, there are a plethora of softwares that are able to perform chord recognition (both open-source and proprietary), that use a various different algorithms to achieve the end goal. There are also various softwares that exist that are able to perform more advanced operations on music signals such as key detection and automatic transcription. Currently however, there is no device that is stand-alone and is able to perform any of these operations on musical signals. The creation of such a device would not only be a great tool but would also be very marketable from a musicians stand-point.

Music theory is a subject that focuses on studying and analyzing the components that music is comprised of such as sound, pitch, melody, harmony, rhythm, form, and notation. This subject is not only studied by musicians,but also by people who are interested in further appreciating and understanding musical compositions. At the heart of the project, is an idea of creating a device that is instilled with enough music theoretical knowledge in order to perform useful calculations and operations. The most basic component is that of a note. In the western tonal music system there are 12 different notes that span across many different octaves. For this project a note will be considered as a mapping of frequencies to various letter names and octaves. When certain combinations of notes are joined together they are able to build more complicated musical components such as scales, chords, and keys; all of these structures can be looked at from a technical standpoint as being varying harmonic structures.

One of the ways that a musical signal can be processed by a computer is using symbolic data type such as MIDI (Musical Instrument Digital Interface) format. The data can either be inputed as an acoustic signal and then transferred into symbolic data or directly input as symbolic data. Using symbolic data contains the advantage that more information about the signal such as velocity and pitch, in the form of on/off messages can be stored. Key recognition using symbolic data has proven to be extremely accurate.[4, 6]

When analyzing an acoustic signal, one of the most useful ways of approaching the problem is by using Digital Signal Processing. By using Digital Signal Processing, various filters and transforms can be applied to the signal in order to manipulate the original signal into something more useful. One of the most useful tools is using the Fourier Transform. When the Fourier Transform is applied to a signal, the signal is transferred from being a function of time and amplitude to being a function of frequency and amplitude. This is useful because after the Fourier Transform has been computed, it is much easier to extract information about various frequencies and therefore notes, that are contained within the signal that would have otherwise been impossible to extract from a signal in the time domain.

When musical signals are processed by a computer to perform these operations without using a symbolic data representation, it often incorporates the use of Pitch-Class Sets or Pitch-Class Profiles.[2, 3, 5, 7, 8, 9] A Pitch-Class Set is a way of extracting individual note information from the signal using frequency analysis, creating a set that is representative of the signal but in terms of the various notes in the 12-tone western music system. A Pitch-Class Profile is essentially a database made up of many Pitch-Class Sets. The Pitch-Class Profile is comprised of various chords or keys depending on if chord recognition or key recognition is trying to be accomplished. This idea is often used explicitly but can has also been used implicitly. One way of searching through Pitch-Class Sets is by using a pattern matching algorithm.[3, 8] Another technique that has proven to be effective is by using Hidden Markov Models. A Markov Model is a probabilistic reasoning tool made up of a series of states and transitions, where each transition is weighted based on the probability of that transition occurring and each possible future state is only dependent on the current state being looked at. A Hidden Markov Model is an extension of a Markov Model, taking the same structure of a Markov Model except that for state, only part of the current state is known. This is most useful when multiple chords are being predicted over a long period of time so that chord transitions can be predicted in order to help identify the tonality of a signal.[2, 5, 9]

3. Design Requirements

In order for this device to be able to be used as a useful tool for musicians, there are some essential requirements that must be met. The most basic of these requirements is the top-level definition of the device. The requirement for the input is that it should be a non-amplified acoustic signal. This means that the input to the device should come from either a microphone contained on the device or an auxiliary cable and should not be of a symbolic data type such as MIDI. Although applying chord detection algorithms to symbolic data is a much simpler task, it is not a very useful data type to be analyzing. Having an acoustic signal input also allows for any acoustic instrument to be able to be analyzed by the device as opposed to needing a specific instrument with a specific connection. The requirement for the output is that it must be displayed on a screen that is located on the device; the output that is to be displayed is not an intricate one so this will not be difficult to satisfy. Another fundamental requirement is that the device must be completely stand-alone. This means that this device should not simply be a program on a computer but rather should operate independently of what it is connected to, who is operating it, as well as the location of the device.

In accordance with how the device will take in and display data, the internal workings of the device must also have design requirements; the device must be accurate and operate quickly. These design criteria, unlike the previous ones are more difficult to quantify. It was determined that the device would be considered to be working reliably if it can identify the correct chord from the acoustic input at least 75% of the time. Although this may accuracy rate may not seem like a high enough one, when things such as chord ambiguity, noise, and mis-tunings are taken into consideration, it becomes a much more reasonable quantification of what accurate should mean for this project. The requirement for the time frame for how quickly the device must be able to detect a chord was chosen to be in between every one and two seconds; this can be considered to be a meaningful time rate for musicians. A very typical speed for rock/pop music to be played at is 120bpm (beats per minute) and if it is taken into consideration that there are four beats per measure and that chords change on average between once and twice a measure than the requirement of chord needing to be detected between every one and two seconds can be deduced easily as one that will be useful for improvisation and .

So far, all of the design requirements that have been discussed have been directly related to the processes of the data acquisition that will occur on the device. There are a few other requirements that must be taken into consideration which apply directly to the underlying goal of making a tool for musicians. The first of these requirements is that the device must be user-friendly; the device not only must be able to be operated easily but it should also be able to be operated by virtually any musician without needing a background in Electrical or Computer Engineering. This should be able to accomplished easily by having the front end have a minimalistic design. The device must also be cost affordable, which has been chosen to be under $200 which will be able to be met by purchasing hardware accordingly. This requirement was chosen so that with the success of the creation of this device, it can be reproduced in a financially affordable manner.

4. Design Alternatives

For this project the most important component of the hardware is going to be the part that is responsible for the actual data acquisition on board the device. When choosing the hardware, there were a couple of key factors that had to be kept in mind. These factors were, how well they would be able to perform, how easily they would be to implement with other parts of the device, and how much they cost. Due to the fact that this project is very software intensive and there is no pre-written software for the project, it is important that the hardware that it is going to be implemented on is chosen carefully.

The first option that was considered was using a type of micro-controller called an Arduino. The Arduino proved to be very cost efficient, costing around 50$. Arduinos are programmed using C/C++ and an open-source IDE which would also be okay for this project because I have some experience with programming in C++. However, what the Arduino does not offer is any direct input for a microphone or auxiliary chord or a direct output for a display. This means that the addition of a microphone, audio jack, or digital display would not only require extra programming but would also require extra wiring; programming, wiring, and of course the debugging that comes along with these can be extremely time consuming. Although other types of micro-controllers could have been considered, Arduino seemed like the best option to consider because of its vast amount of libraries and support.

The next option that was considered was using an FPGA such as the DE2 Altera. I have had some experience with using this FPGA from the ECE-318: Digital Design course that I took last year which helped me way the decision in using an FPGA. If an FPGA were to be used it would most likely be programmed in VHDL or Verilog. These are two programming languages that I have a little experience with, but not a considerable amount. An FPGA would be great for doing many filter calculations because of its extremely fast computing speed and I know from experience that the DE2 Altera has built in jacks for audio input as well as an audio core which can be accessed. However, the biggest setback with using an FPGA such as the DE2 Altera also comes from experience. I have previously tried to do a project involving the audio core on the DE2 altera, and it proved to be very difficult; I ran into many problems involving timing analysis as well as clocking issue when trying to use the audio core along side the video core, which would be responsible for the display. Creating a stand-alone device for chord detection is a project which is inherently much more complicated than the project I had previously attempted on an FPGA so it was decided that this would not be the best option either.

Another option that was considered was incorporating a Raspberry Pi as the base for the hardware portion of the device. The Raspberry Pi itself is a very diverse and adaptable device, so integrating a microphone and display into the device would be extremely easy since there is hardware that offers 'plug and play' compatibility. The Raspberry Pi is also able to be programmed using many languages including Java, Python, C, C++, and Ruby. This would be useful because Python and Java are languages that offer many libraries and support for digital signal processing; I also have the most experience with these two programming languages. The Raspberry Pi is also very cost affordable, costing only $40 for the base component.

5. Preliminary Proposed Design

The device that is to be constructed consists of a hardware and a software component that will have to be integrated together. The design of the hardware component consisted of choosing what parts would need to be purchased in order to complete this project, where as the design of the software component consisted of designing an algorithm to do the data acquisition on the hardware and planning on how to integrate all of the hardware together. The following figure(Figure 1) shows a top-level diagram of the preliminary proposed design that is to be integrated.

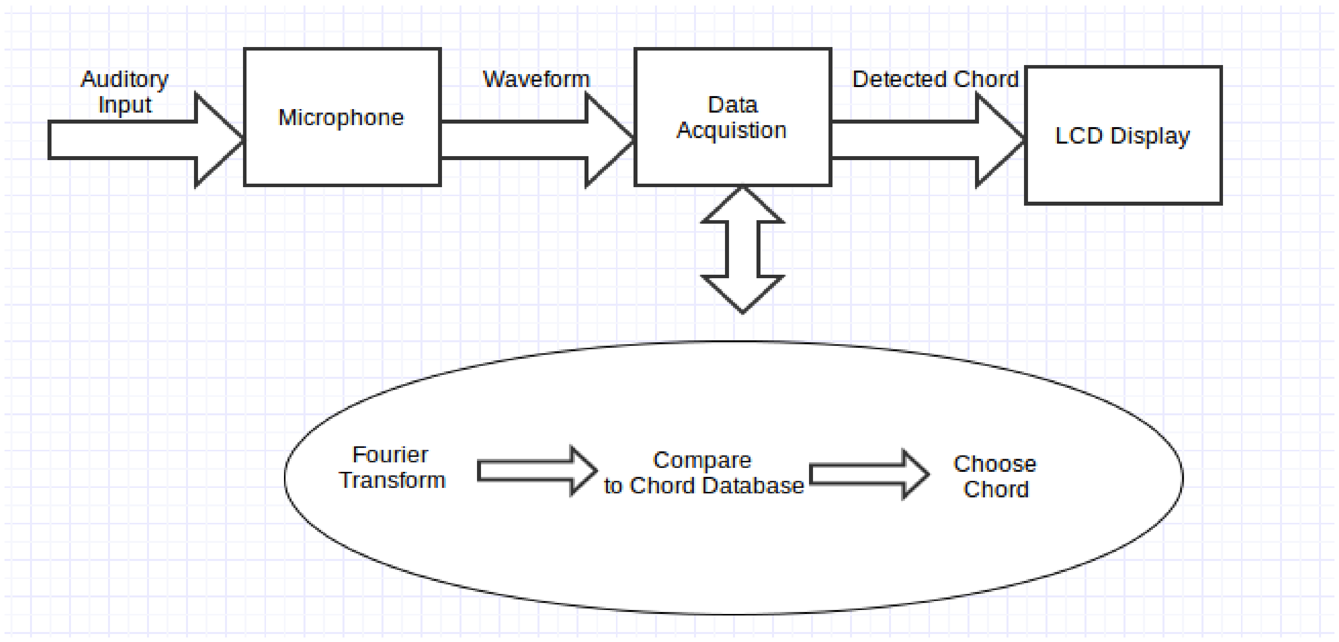


Figure 1: Top Level Diagram of Preliminary Proposed Design

5.1 Preliminary Hardware Design

The following table(Table 1) lists the hardware parts that were chosen for the preliminary proposed design in order to create this device; these are also the hardware components that were requested in the Student Research Grant Proposal Application.

Device	**Estimated Cost ($)**
Raspberry Pi Model B+ (http://www.adafruit.com/product/1914)	40
Wolfson Audio Card (http://www.adafruit.com/products/1761)	35
16x2 Character LCD Display (https://www.sparkfun.com/products/255)	15
SD Card 8gb	10
Total:	100

Table 1: Hardware and Cost Estimates for Preliminary Design

The Raspberry Pi B model was chosen as the base for the data acquisition system because it was deemed the most suitable for this project in accordance with the design requirements that have been outlined. The Wolfson Audio Card was chosen as an additive to the Raspberry Pi as an audio card component that will not only allow for better digital signal processing and sound manipulation to be done but also contains a higher quality audio input jack; this component was chosen in order to ensure that the input for the device would be able to be a non-amplified acoustic signal. The LCD display that was chosen is a very cost affordable and simple display that can easily be integrated with the Raspberry Pi. The output display that is to be shown on the device is simple and so it was determined that this display would be appropriate in accomplishing the output design requirement. The hardware will be driven by integrating the parts together and programming an algorithm controlling the data acquisition based on the proposed software design. This program will be stored on the 8gb SD card and is able to be loaded onto the Raspberry Pi from the SD card.

5.2 Preliminary Software Design

The software aspect of the preliminary proposed design encompasses an algorithm based on a couple of basic ideas. The first is the Fourier Transform, which is the heart of digital signal processing on auditory signals. The transform will be used to transfer the input data from the time domain into the frequency domain which will allow for the amount of each individual frequency to be quantized and extracted from the original signal. This data will be able to be stored for comparison. Applying the Fourier Transform to a signal will also allow for various filters to be applied to the signal. The first step of the algorithm that is going to be incorporated is to apply the Fourier Transform to the input data, as well as a series of filters that will be used to reduce noise and highlight specific harmonic structures.

Another basic idea that was incorporated into the software design of this project was that of a Pitch Class Profile. A Pitch Class Profile is essentially a database that is created before the program is ran in real-time. This database or library will contain information about the harmonic structure of various chords. The various filters that will be applied to the input signal will allow for the the input signal to be compared to the library in order to try and match the input signal to a chord in the library. The matching progress will be done using probabilistic reasoning. Since no input signal will identically match anything that is contained in the library, a sensitivity will be created and the algorithm will try and detect which chord is in the input signal by finding which item in the library has the highest probability of being correct. The next steps in the algorithm are to compare the filtered input signal to the chord database and use probabilistic reasoning in order to detect which chord is most likely to be contained in the input signal. The following figure(Figure 2) shows the steps that outline the algorithm of the preliminary design that is to be implemented for the data acquisition on this device:

1.) Receive acoustic input signal from microphone/auxiliary cable

2.) Apply Fourier transform and various filters to signal

3.) Compare results to chord database

4.) Choose which chord is contained in the input signal using probabilistic reasoning

5.) Display output by way of the LCD screen on the device

Figure 2: Algorithm for Software of Preliminary Design

6. Final Design and Implementation

In the following sections I will discuss the final design and implementation that were used for my senior project. The following sub-sections will discuss the hardware implementation, the hardware integration, as well as the algorithm design and software. The figure below(Figure 3) shows the top-level schematic of the final implementation of the system

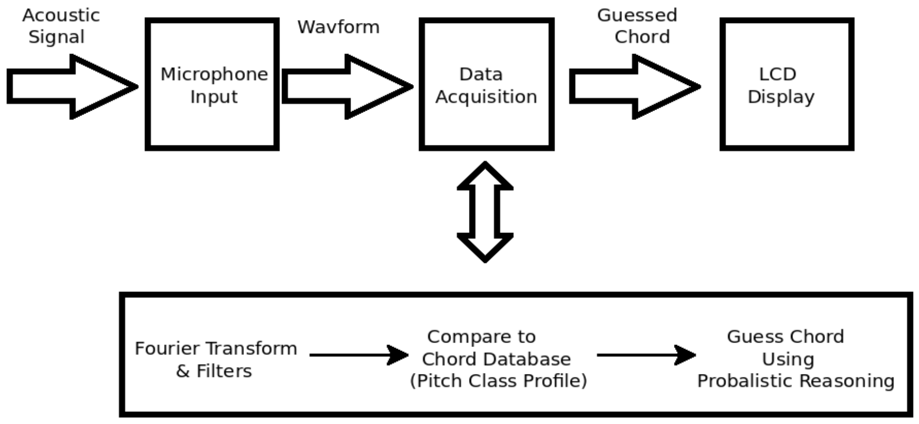

Acoustic Signal
Microphone Input
Wavform
Data Acquisition
Guessed Chord
LCD Display
Fourier Transform & Filters
Compare to Chord Database (Pitch Class Profile)
Guess Chord Using Probalistic Reasoning

Figure 3: Top Level Diagram of Implemented Design

6.1 Hardware Implementation

The first part of the hardware that had to be decided on was the component that would be performing the data acquisition as well as linking and integrating the other hardware components together. After considering various micro-controllers as well as an FPGA, it was decided that a Raspberry Pi Model B+ would be the most suitable choice for this project; the Raspberry Pi includes 4 USB ports, GPIO (General Purpose Input Output) pins, an Ethernet port, as well as the ability to have an underlying operating system on the device (which would make it easy to integrate devices, drivers, as well as program the device remotely). The following figure(Figure 4) shows a picture of the Raspberry Pi model that was purchased and used.

Figure 4: Raspberry Pi Model B+

The second piece of hardware that had to be decided on was the component that would be responsible for displaying the output of the device. Since the output that needed to be displayed is simple and can be easily represented solely with ASCII characters, the need for an elaborate or large display was deemed unnecessary. The hardware that was decided on was a LCD 16x2 Character Display that included push buttons. The push buttons would prove to be very useful during the debugging process as well as for controlling the input to the device. Once the LCD display was purchased and arrived, it needed to be assembled and soldered together. Once this had been done, the display was abled to be attached to the Raspberry Pi by way of the GPIO pins. The following figure(Figure 5) shows a picture of the display that was used after being attached to the Raspberry Pi.

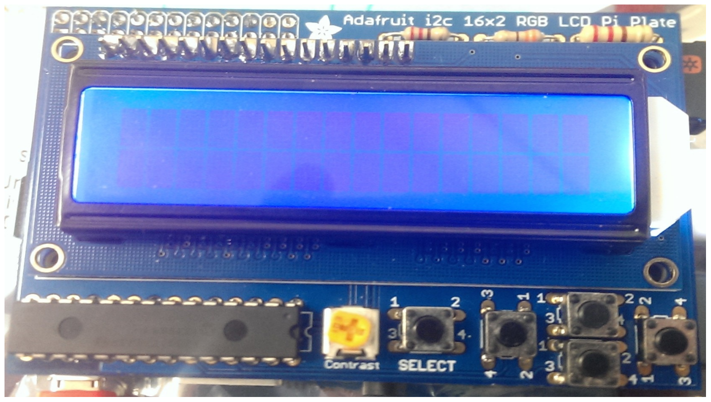


Figure 5: 16x2 LCD Character Display Attached to Raspberry Pi

The final piece of hardware that was decided on was the the piece of hardware that would be responsible for handling the acoustic input to the device. The device had to be chosen in accordance with the design requirements for the input which included the input not needing to be amplified as well as not being of a symbolic data format such as MIDI. The first choice that was chosen was the Wolfson Audio Card which offers a high-quality audio input jack as well as a direct connection the Raspberry Pi. However, this was not able to be used because in the newer versions of the Raspberry Pi (like that was purchased and used) the 8-pin GPIO audio header has been removed so an alternative solution had to be sought out. After considering alternative ways to receive an auditory input to the Raspberry Pi and since the Raspberry Pi does not offer a line-input jack by itself, it was decided that a USB sound card would be used. A USB sound card would be accurate enough to receive non-amplified signal through a microphone for data acquisition and also be very easy to attach. The tricky part would then be working on integrating the USB sound card with the rest of the system, however since the Raspberry Pi has an underlying operating system running on it, this meant that this could be done through the operating system. The following figure(Figure 6) shows a picture of the USB sound card after being attatched to the Raspberry Pi, that was purchased and used for the project.

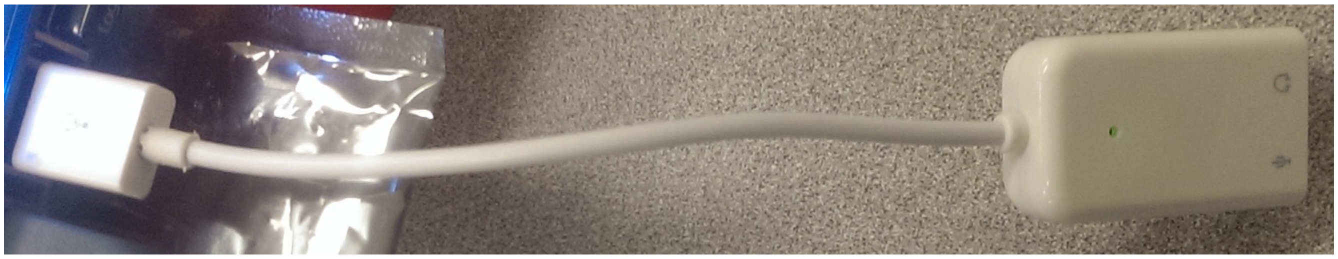


Figure 6: C-Media Electronics USB Sound Card

There were two other hardware components that were used for this device, however these components came as a result of selecting the rest of the hardware. The first component was a micro-SD card. The micro-SD car was used as memory for the Raspberry Pi which stored the operating system as well as all the other files and drivers on the Raspberry Pi. An 8gb micro-SD card was chosen and purchased which was considered to be enough memory for the project. The following figure(Figure 7) shows a picture of the micro-SD card that was purchased and used.

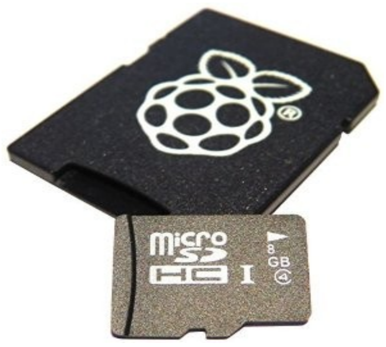


Figure 7: Micro-SD Card

The final hardware component that was needed for this project was a microphone that would be plugged into the USB Sound card. Since one of the design requirements was portability, cost affordable, as well as being able to operate independently of user and location. For these reasons it was decided that instead of purchasing an expensive and bulky microphone that a simple headset microphone would be used instead. The following figure(Figure 8) shows a picture of the headset and microphone that was used for acquiring the auditory input to the USB sound card for the project.

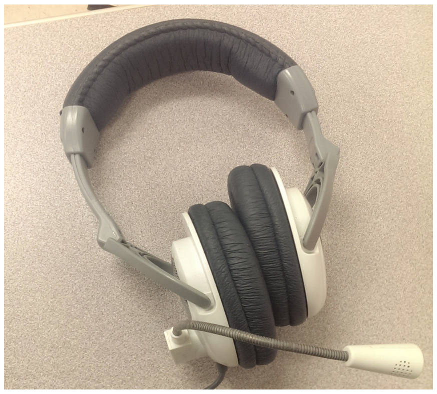


Figure 8 : Headset with Microphone used for Auditory Input

The following section will discuss how these devices were integrated together using software along side the Raspberry Pi.

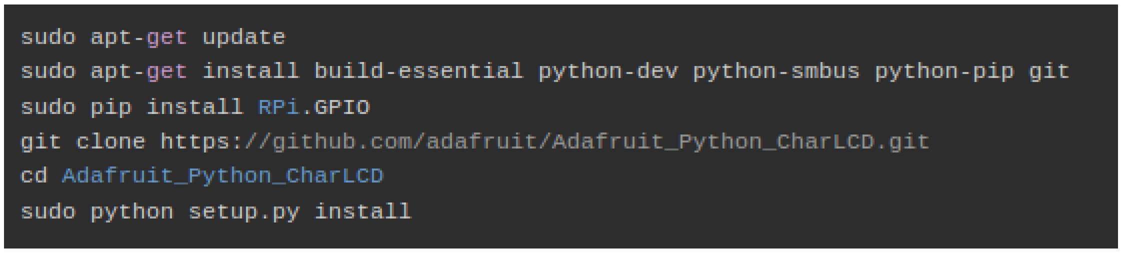
6.2 Hardware Integration

The first step that was needed to be done in order to integrate the hardware was deciding what operating system that the Raspberry Pi should run on. After considering various operating systems that have been made for the Raspberry Pi such as NOOBS and RISC OS, it was decided that Raspbian would be used. Raspbian is a debian based distribution of Linux made specifically to optimize the hardware of the Raspberry Pi; having a version of Linux installed as the operating system would greatly reduce the learning curve of learning a new operating system entirely as well as allow for drivers to be made more accessible (if a version of the driver already exists for a version of Linux).

Installing and setting up the Raspberry Pi with the Raspbian OS was a fairly simple task. First, the image file of the operating system had to be loaded on to the micro-SD card, which was done by using a program called "Pi-Baker" on another computer which contained a card reader. Once this had been done, the micro-SD card was inserted into the Raspberry Pi, an external screen was connected using an HDMI cord, and a keyboard and mouse were connected to the Raspberry Pi by way of the USB ports. Once the raspberry Pi was given power, a setup screen was displayed on the screen that was easily navigable by using the keyboard and mouse.

Once the operating system had been configured, the IP Address for the Raspberry Pi was found by using the 'ifconfig' command in terminal which would allow the Raspberry Pi to be accessed remotely by the 'ssh' command from a laptop. This was useful because it allowed for the project to be worked on without having a screen, mouse, or keyboard attached to the Raspberry Pi but it also meant that everything would be done from command line.

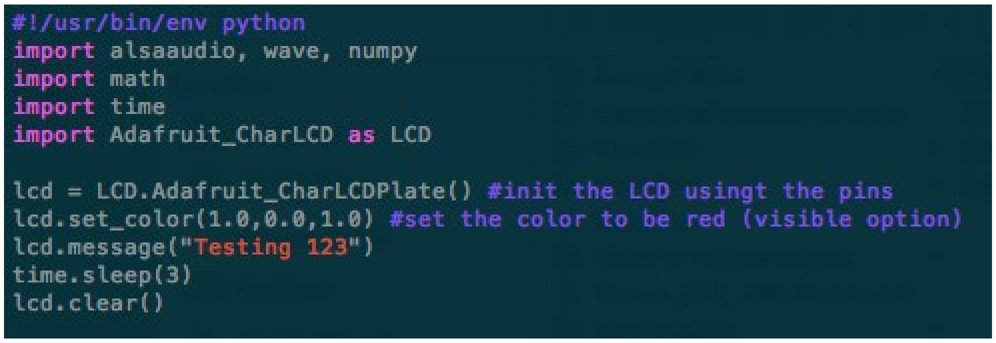
The next step that was done was integrating the display system with the Raspberry Pi. Once the LCD Display had been assembled and soldered it was connected by simply plugging the male ports of the display into the female GPIO ports on the Raspberry Pi. In order to get the display working with the Raspberry Pi in a python program 'python-smbus' and 'i2c-tools' must be installed using the '> sudo apt-get install' command in terminal. Once these dependencies have been installed, the i2cdetect program can be called by typing '> sudo i2cdetect -y 0' into terminal. This will detect the pins that are attached to the GPIO pins; if 0x20 appears in the window then the the LCD display is connected. Next, a few more dependencies must be installed in order to easily access the display. The following figure(Figure 9) shows a screen shot from the Adafruit website of the commands needed in order to install the LCD Display for use in python scripts.



```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus python-pip git
sudo pip install RPi.GPIO
git clone https://github.com/adafruit/Adafruit_Python_CharLCD.git
cd Adafruit_Python_CharLCD
sudo python setup.py install
```

Figure 9: Installing Adafruit_Python_CharLCD Library[10]

The Adafruit_Python_CharLCD library comes with example codes as to how to control the messages displayed on the LCD display as well as how to retrieve push button information, so from these examples a basic understanding of how the library works was easily deduced. The following figure(Figure 10) shows python code that initializes and displays a simple message on the LCD display. The contrast of the LCD display was adjusted using a screwdriver to turn the yellow contrast dial until the message was completely visible.



```
#!/usr/bin/env python
import alsaaudio, wave, numpy
import math
import time
import Adafruit_CharLCD as LCD

lcd = LCD.Adafruit_CharLCDPlate() #init the LCD usingt the pins
lcd.set_color(1.0,0.0,1.0) #set the color to be red (visible option)
lcd.message("Testing 123")
time.sleep(3)
lcd.clear()
```

Figure 10: Python Code for Displaying a Simple Message on Adafruit Display

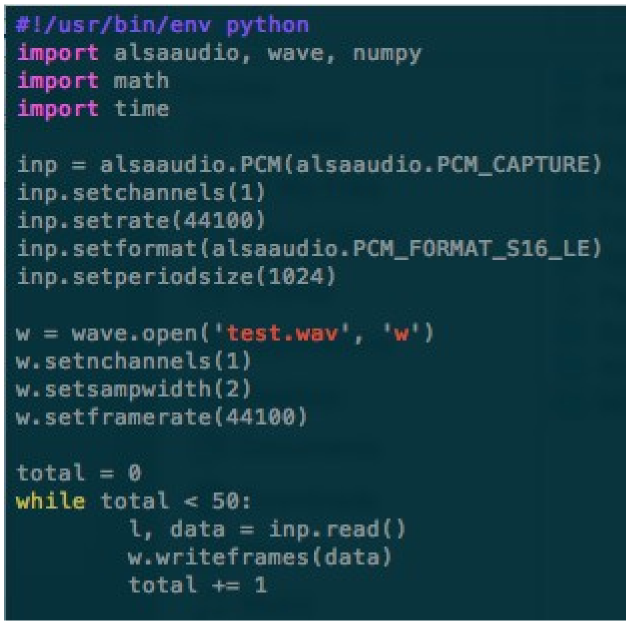
The final step that needed to be done in order for all of the hardware to be integrated together was getting the USB Sound Card working with the Raspberry Pi and being able to control it from a python script. This was done by way of using the ALSA (Advanced Linux Sound Architecture) audio drivers. The first step was to make sure the drivers were installed by typing the following command into terminal: '> apt-get install libasound2 alsa-utils alsa-oss '. After the drivers have been installed, they must be tested to see if they are working properly which can be done by attaching a microphone to the input and headphones to the output of the USB Sound Card, making a short recording, and then playing it back to see if something was being properly recorded. This is done by using the following 2 commands:

“> arecord -f SIG_LE -r 16000 -D default > test_record.wav”

“> aplay test_record.wav”

However, this did not initially work because the ALSA driver was prioritizing the on-board sound of the Raspberry Pi above the USB Sound Card. After trying to debug this problem using the ALSA mixer with the user interface, it was found that the problem needed to be fixed by editing a driver file. By editing the file “/etc/modprobe.d/alsa-base.conf” and changing the line: “options snd-usb-audio index = -2” to “options snd-usb-audio index = 0” the problem was fixed and after rebooting the system was tested using the commands listed above.

Once the ALSA drivers had been seen to be working, the python library for ALSA had to be installed which was done by using the terminal command: '> sudo apt-get install python-alsaaudio' The python module was tested by writing a short python script which made a recording. A simple python script incorporating the ALSA library can be seen in the following figure(Figure 11):



```
#!/usr/bin/env python
import alsaaudio, wave, numpy
import math
import time

inp = alsaaudio.PCM(alsaaudio.PCM_CAPTURE)
inp.setchannels(1)
inp.setrate(44100)
inp.setformat(alsaaudio.PCM_FORMAT_S16_LE)
inp.setperiodsize(1024)

w = wave.open('test.wav', 'w')
w.setnchannels(1)
w.setsampwidth(2)
w.setframerate(44100)

total = 0
while total < 50:
        l, data = inp.read()
        w.writeframes(data)
        total += 1
```

Figure 11: Simple Recording Using ALSA Python Library

Now, that all the hardware components have been integrated together on the Raspberry Pi and can be accessed and controlled in python scripts, the data acquisition algorithm which is discussed in the following section could be implemented.

6.3 Software Implementation and Algorithm

The program that runs on the Raspberry Pi was written entirely in python, utilizing many python libraries. The algorithm that was used in the final implementation differed slightly from the preliminary proposed algorithm because it uses normalization as well as comparing the input to templates using the Pitch Class Set instead of a database. Using chord templates was ideal because it allowed for a multitude of space to be saved, as well as using a search algorithm based on music theory intuition. For this implementation the use of probabilistic reasoning became obsolete because when the system is trying to detect a chord in the input signal, there is no knowledge of the prior chords that have already been detected. The following figure(Figure 12) outlines the steps in the algorithm that was used in the final implementation:

1.) Obtain input from microphone and USB Sound Card

2.) Apply Fourier Transform and other filters

3.) Extract note information from frequencies to form Pitch Class Set

4.) Normalize Pitch Class Set and create boolean Pitch Class Set

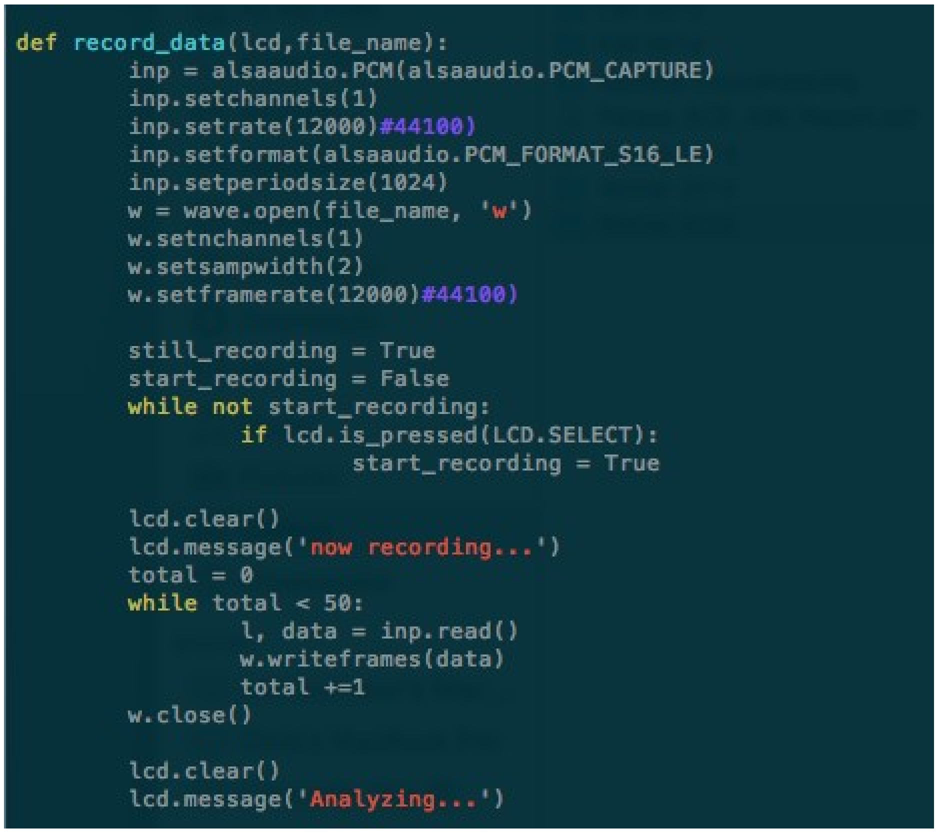
5.) Compare boolean Pitch Class Set to Chord Templates to guess 'chord' or 'note'

6.) Display guessed chord on LCD Display

7.) Repeat

Figure 12: Algorithm for Final Software Implementation

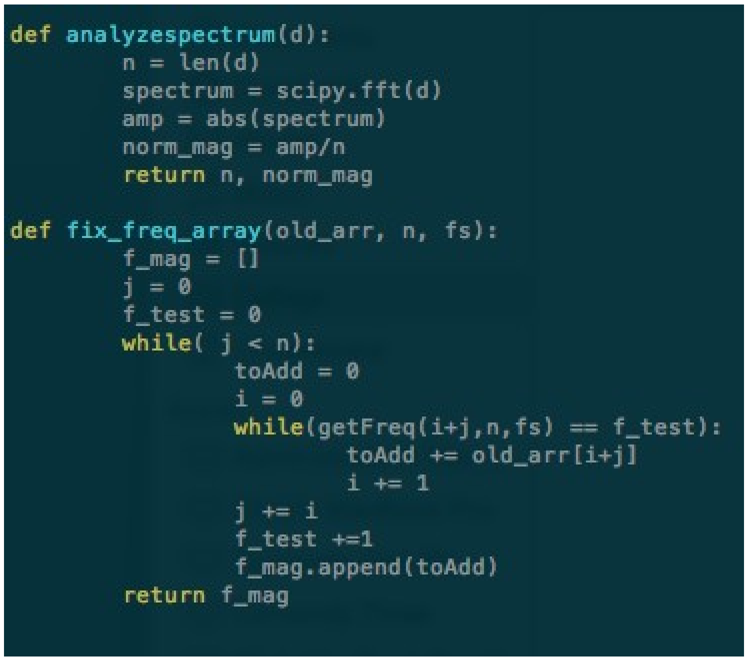
The first step of the algorithm was to obtain the input data, which was done using the ALSA audio library for python. The ASLSA module is initialized and is continuously reading in data at a sampling frequency of 12000Hz from the microphone; this sampling frequency was chosen to optimize speed as well as accuracy of the data. Originally the system was sampling at 44100Hz which would mean the file would be of standard CD quality. However sampling at this frequency caused the program to run much slower, and only the highest frequency that is analyzed is around 6,000Hz so therefore 12,000 Hz was deduced as an appropriate sampling frequency. The data is not used until the select button on the LCD display is pressed. Once the select button on the LCD display is pressed, the data from the microphone is written into a .wav file for approximately one second; While data is being written into the .wav file, the LCD display displays that the system is recording data. After the data has finished being written the LCD display displays that the system is now analyzing. The following figure(Figure 13) shows the python function that executes this part of the algorithm.



```
def record_data(lcd,file_name):
        inp = alsaaudio.PCM(alsaaudio.PCM_CAPTURE)
        inp.setchannels(1)
        inp.setrate(12000)#44100)
        inp.setformat(alsaaudio.PCM_FORMAT_S16_LE)
        inp.setperiodsize(1024)
        w = wave.open(file_name, 'w')
        w.setnchannels(1)
        w.setsampwidth(2)
        w.setframerate(12000)#44100)

        still_recording = True
        start_recording = False
        while not start_recording:
                if lcd.is_pressed(LCD.SELECT):
                        start_recording = True

        lcd.clear()
        lcd.message('now recording...')
        total = 0
        while total < 50:
                l, data = inp.read()
                w.writeframes(data)
                total +=1
        w.close()

        lcd.clear()
        lcd.message('Analyzing...')
```

Figure 13: Python Function That Records Microphone Data on Button Press

Once the data has been written to a .wav file, the second part of the algorithm can now be executed. This step is responsible for computing the Fourier Transform of the signal which transfers the signal from the time domain to the frequency domain; the Fourier Transform allows for information about the amount of each frequency contained in the signal to be obtained. This is done by using a python library called 'scipy' which includes a Fast Fourier Transform function. The .wav file that was written to in the first step of the algorithm is now read from, transferred into the frequency domain, and stored into an array. Initially the array index corresponds with the bin value of the transform, so the array is manipulated so that the array index corresponds with the frequency in hertz of the signal. The following figure(Figure 14) depicts the python functions that were used for this step of the algorithm.



```
def analyzespectrum(d):
        n = len(d)
        spectrum = scipy.fft(d)
        amp = abs(spectrum)
        norm_mag = amp/n
        return n, norm_mag

def fix_freq_array(old_arr, n, fs):
        f_mag = []
        j = 0
        f_test = 0
        while( j < n):
                toAdd = 0
                i = 0
                while(getFreq(i+j,n,fs) == f_test):
                        toAdd += old_arr[i+j]
                        i += 1
                j += i
                f_test +=1
                f_mag.append(toAdd)
        return f_mag
```

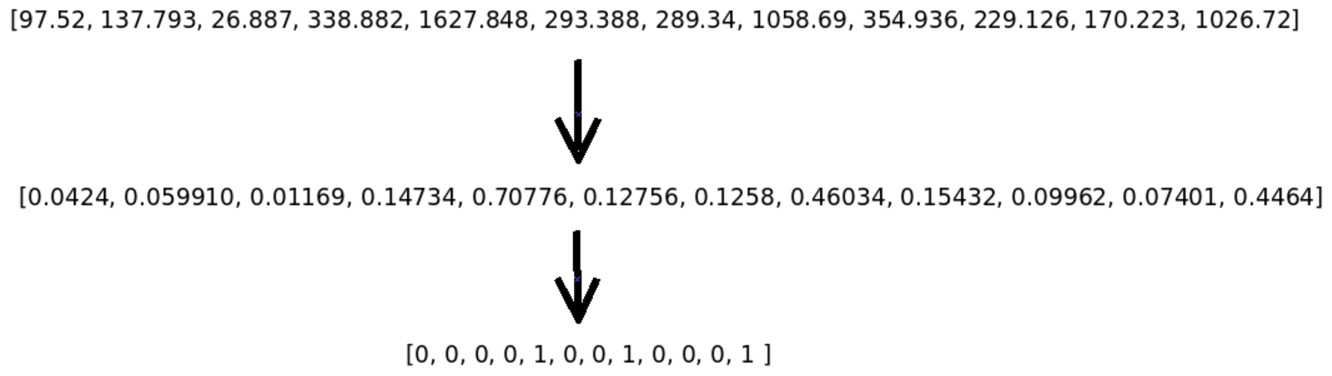
Figure 14: Python Functions that Analyze the Spectrum and Returns Array where Index is in Hz

The next step in the algorithm is to extract note information and form a pitch-class set. In this program, a pitch-class set is essentially an array of length 12 where each index of the array corresponds with one of the 12 notes in the 12-tone system. Although there are eight octaves for each note, this is taking into account that a note can be identified regardless of octave. This is done for each note by accessing the eight indexes which correspond with the eight frequencies for each note, adding all of these values together, and then placing this value into the index corresponding with the note in the pitch-class set array. This forms a pitch-class set array of the following form where each index corresponds with the approximate total amount of the note contained in the signal(Figure 15).

[C, C#\Db, D, D#\Eb, E, F, F#\Gb, G, G#\ Ab, A, A#\Bb, B]

Figure 15: Array Representation of the Pitch Class-Set

Now that the pitch-class set has been created, the next step is to normalize the pitch-class set and create a boolean pitch-class set in preparation for it to be compared to the templates. The array is normalized by using simple vector math, considering the array to be a vector and dividing each value in the array by its length. This makes it so that all values in the pitch-class set are less than 1 which makes interpreting the data as well as the debugging much easier. The boolean pitch-class set is formed by making the three max values in the pitch-class set one and all the other values 0. This is done because it allows for easy comparison to the templates, as well as taking into account that all triads are based off three notes (the root, the third, and the fifth). Currently, the device is only able to identify triad chords, so this method sufficed. The following figure(Figure 16) depicts the array manipulation done on the pitch-class set.



```
[97.52, 137.793, 26.887, 338.882, 1627.848, 293.388, 289.34, 1058.69, 354.936, 229.126, 170.223, 1026.72]

[0.0424, 0.059910, 0.01169, 0.14734, 0.70776, 0.12756, 0.1258, 0.46034, 0.15432, 0.09962, 0.07401, 0.4464]

[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1 ]
```

Figure 16: Array Manipulation on Pitch-Class Set

The next step in the algorithm is to compare the boolean pitch-class set to the chord templates. Each chord template is also of a boolean form where there are only three values out of 12 that are 1 and the rest are 0. Since the system only handles the basic 12 major and 12 minor triads there are only 24 templates that are contained within the program (one for each chord). If the pitch-class set that was computed from the input signal matches one of these templates then a chord has been successfully been detected and this chord is displayed on the LCD display. If the pitch-class set does not match any of the templates then the program finds the note with the highest amplitude from the non-boolean pitch-class set and displays this on the LCD display as a note. Once this has been done, the program waits until the select button is pressed and then repeats the algorithm.

7. Performance and Results

Based on the preliminary design, expected results were able to be estimated. Since the preliminary design hardware and software was based on satisfying the design requirements, the design requirements were essentially the expected results for the performance of the device.

The device was expected to have its input be a non amplified acoustic signal obtained from a microphone or auxiliary cable and not be of a symbolic data type (such as MIDI format). The final implementation incorporated a USB Sound Card containing an 1/8 inch input jack, which a microphone or auxiliary cord can be plugged into. Since this is the only operable input to the device, symbolic data is not supported. Additionally the device was tested with an acoustic guitar and so amplification was not needed. For the input to the device, the final implementation met the expected results as well as the initial design requirements.

The device was required to have an output that was displayed on a screen located directly of the device. This was the expected result because of the preliminary design containing the 16x2 LCD Char Display. The final implementation contained the same LCD Display as the preliminary design as well as having it functioning properly as the output as well as the buttons functioning for controlling the input.

Another one of the design requirements was that the device must be accurate, meaning the device must extract information accurately at least 75%. This percent of accuracy was chosen when taking into factors such as chord ambiguity, mis-tunings, and resonance. This was also the expected result based on the preliminary design however there are some discrepancies that came about when testing the final implementation of the device. The device has an accuracy of about 95% when detecting the correct note being played on a tuned instrument playing only a single note. Quantifying the accuracy of the device when detecting cords was a little bit trickier because there were more variables that needed to be taken into account. It was discovered that the device is much more accurate when the duration of the chord is shorter having an accuracy of about 85% in comparison to chords that are left to resonate having an accuracy of about 70%. Also, the device was solely tested on a acoustic guitar so the true accuracy of the device across many instruments is not entirely known. It was also noted that when playing chords that if a single string was out of tune on the guitar that the device's accuracy would decrease to rate of about 40%. The design of the algorithm could be changed to increase the accuracy of the device by accounting for mis-tunings, as well as having a trigger activated system instead of a push-button activated system.

The device was also required to operate at a meaningful rate for musicians which was chosen to be between 1 and 2 seconds. This rate was chosen by taking into consideration that an average speed of music is 120bpm(beats per minute), with a common time signature, and chords changing between one and two times a measure. The current implementation of the system has the device record for one second after the push-button has been pressed and then analyzing the results. Analyzing the results takes about one second to accomplish, so this meets the original design requirements and expected results. Using the newest version of the Raspberry Pi, the Raspberry Pi 2 would help increase the speed of the system, however this version had not been released yet when the project was being implemented.

Another one of the expected results was that the device would be stand-alone in accordance with the design requirement of being stand-alone as well as the definition of the project. The hardware components were carefully chosen in order to satisfy this result. Stand-alone for this project was meant to mean that the 'device' would not simply be a computer program, that all data acquisition would occur on the machine, and that the device would operate independently of location, user, and attachment cords. The final implementation of the device is portable and can operate independent of location and user, however currently a power cord must be plugged in so that the device is able to receive power. In the future, this will be fixed by adding a battery pack to the device. When the device is initially powered up, the Raspberry Pi must be connected to the Internet via an Ethernet can so that the main program can be started remotely. Once the program is running on the device however this cable can be disconnected. This can be fixed however by having the program run on the start up of the Raspberry Pi, however this was removed so that the program can be continuously modified.

The final design requirements were that the device must be user friendly, cost affordable meaning all components must total less than $200, and that the device must not cause copyright infringement on music. The final implementation of the device is very easy to use, the user must only press one button to operate the device. The device was also very cost affordable, tallying a total cost of just over $100. The device also will not cause copyright infringement because the current design is activated via a push button, meaning that it does not continuously sample. On top of this, the device only detect chords and does not perform automatic transcription so at the very most, only chord changes in a song would be able to be detected using this device and not actual melodies or harmonies which is where the dilemma of copyright infringement would come about.

8. Production Schedule

The production of this project was based on several phases that arose from logical planning of how to create the final product. The following figure(Figure 17) displays a flowchart showing the phases of the project.

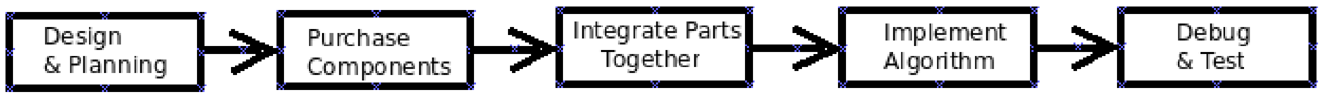


Figure 17: Flow Chart for Work Phases of the Project

The first phase of the project was the longest and consisted of designing and careful planning for which components to purchase, how to integrate them together, as well as the algorithm. The next step was to order the all the hardware components that would need to be purchased for the project. Once the parts arrived, they then needed to be integrated together so that all of the parts could communicate with each other and be controlled from a python program. Once this had been accomplished the actual algorithm could be programmed implementing all the hardware components to be controlled by one program. The final phase of production was the debugging and testing phase. This work flow worked well with the exception that the hardware components were ordered later than would have been convenient so the phases following this were pushed back.

9. Cost Analysis

The following table displays the cost of all components that were needed to be purchased in order to complete this project.

Component	Cost ($)
Raspberry Pi Model B+ (http://www.adafruit.com/product/1914)	39.95
USB-Sound Card (http://www.adafruit.com/products/1475)	4.95
16x2 Character LCD Display (https://www.sparkfun.com/products/12825)	19.95
Micro-SD Card (8gb) (https://www.sparkfun.com/products/12998)	11.95
Total:	76.80

Table 2: Purchased Components Used In Final Implementation

When implementing the project, the power cord for the Raspberry Pi and the microphone connected to the USB-Sound Card did not need to be purchased because they were already readily available. The power cord that was used was a recycled phone charge and the microphone was from an old headset. However, neither of these components are particularly expensive. The power chord that the Raspberry Pi uses is a micro-USB 2.0. These chords are fairly inexpensive and cost around $5.00 . The actual model of the headset contain the microphone that was used is not known, however similar headsets with microphones that have an 1/8 in jack input cost around $10.00 . The following table(Table 3) shows the estimated total cost of all the components that were used in the project if all the components were individually purchased.

Component	Cost ($)
Raspberry Pi Model B+ (http://www.adafruit.com/product/1914)	39.95
USB-Sound Card (http://www.adafruit.com/products/1475)	4.95
16x2 Character LCD Display (https://www.sparkfun.com/products/12825)	19.95
Micro-SD Card (8gb) (https://www.sparkfun.com/products/12998)	11.95
Power Chord	~ 5.00
Microphone	~ 10.00
Estimated Total:	~ 93.00

Table 3: Estimated Total Cost of Components Used in Final Implementation

As shown, even with the extra cost expenses for the power chord and the microphone, the total cost of all the components used in this device is still inexpensive and satisfy the design requirement goal of the device being cost affordable and less than $200.00 . The total cost of all of the components is actually about half of what was allocated in the design requirements. Recently there has been a newer version of the Raspberry Pi released that costs the same price, is backward compatible with components, and is much faster. If this device was to be reproduced, it would be able to be done in a cost affordable manner.

10. User's Manual

The device is very easy to use, however there are a few things that must be done before the program will be running propperly on the device. The first step in using the device is making sure that all components are properly attached to the Raspberry Pi. This includes ensuring that the USB Sound Card is plugged completely into the USB port on the Raspberry Pi, that a microphone is connected to the input of the USB sound card, that the LCD Display is fully connected to the Raspberry Pi's GPIO pins, and that the micro-SD card is loaded into the Raspberry Pi's memory slot. The following diagram illustrates where on the Raspberry Pi each of these things are located.

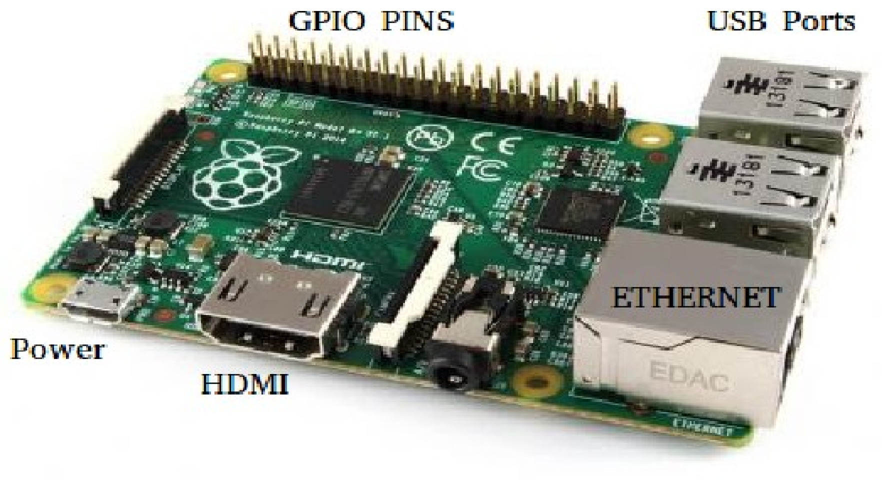


Figure 18: Raspberry Pi With Ports Labeled

Then the Raspberry Pi must be given power by plugging in a power chord to the Raspberry Pi and connecting it to a standard wall outlet using a standard micro-USB 2.0 with a wall adapter connection. Following this, the Raspberry Pi must be logged into, which can either be done remotely or directly. To login remotely the Rasspberry Pi must be hooked up to a network through its Ethernet port; the device can then be connected to at pi@ip-address. To login directly to the Raspberry Pi connect a keyboard to a USB port on the Raspberry Pi and a monitor through the HDMI port. The user will be prompted for the login info for which the login is 'pi' and the password is 'raspberry'. The final steps in setting up the device is to change the directory to 'Capstone_Project' and running the program by typing 'python demo_program.py' into terminal.

Once these steps have been done and the program is running, all the user needs to do in order to detect a chord is press the 'select' button on the LCD Display. The program will record for one second, analyze the data, and then display the results on the LCD Display. The following figure(Figure 19) illustrates the location of the 'select' button on the LCD Display.

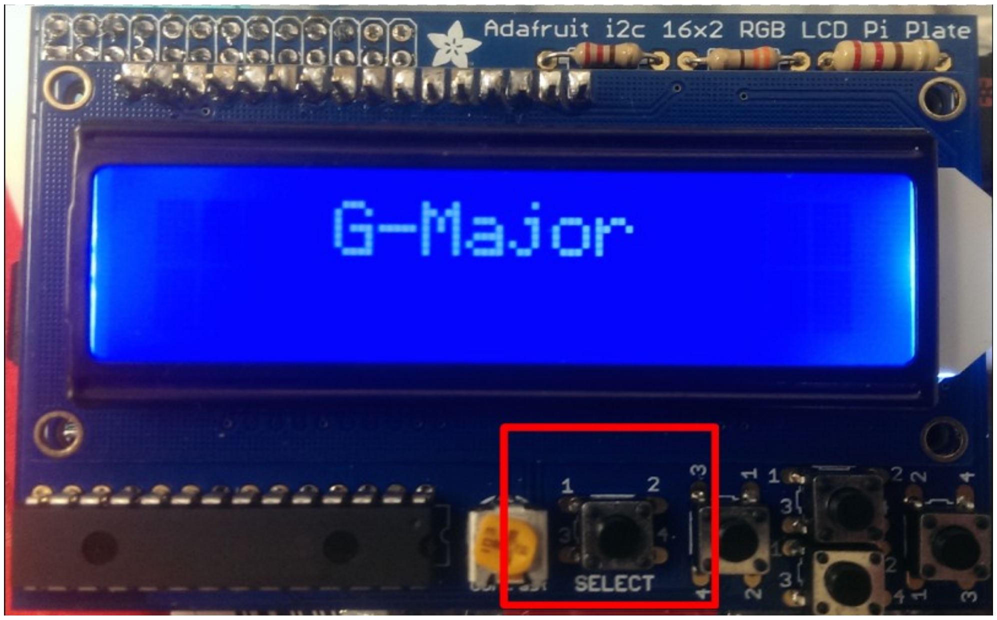


Figure 19: LCD Display with "Select" Button Highlighted

11. Conclusions

The overall goal for this project was to create a device that could serve as a useful tool for musicians. More specifically, the aim was to create a stand-alone device that can accurately recognize harmonic structures in an acoustic signal obtained from a microphone input. Using and integrating a Raspberry Pi, a USB-Sound Card, and an LCD Display a device was constructed that accomplishes this. All the design goals for this project were either met or nearly met. The device is nearly stand-alone with the exception of a power cord and connections needed to be made during the initial setup. The input is an acoustic signal received from a microphone, the output is displayed on the device, and the device is user friendly and cost affordable. The device is able to quickly detect the harmonic structures in the signal at a rate that would be useful for musicians. The accuracy of the device is high for single note detection of the 12-notes in the 12-tone western music system and detecting the 12 basic major and minor triad chords of short durations of a well tuned instrument. With some improvements to this device, it will be able to be used efficiently as a musician's tool. The following figure(Figure 20) depicts the constructed device next to a tape measure for scale.

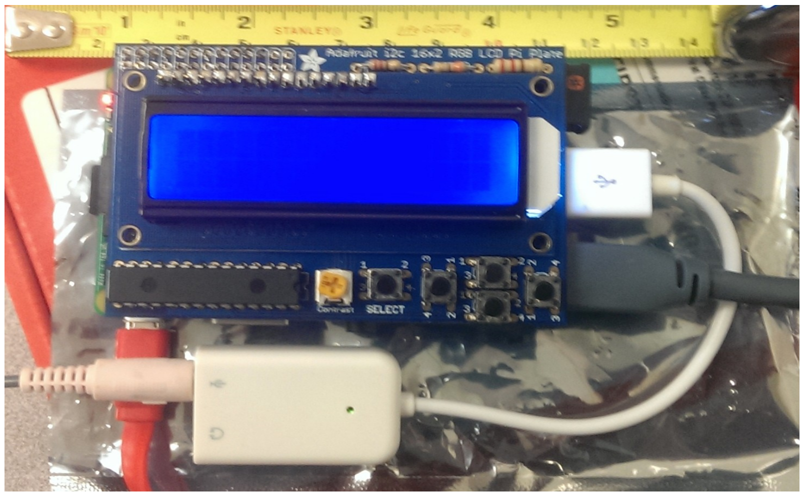


Figure 20: Final Implemented Device With Scale

11.1 Future Work

There are some improvements that can be made to both the device and the software of this system that can improve the usability. The improvements that can be made to this device that stand out the most are those pertaining to software improvement. The software can be improved by first adding error checking to account for mis-tunings and other errors in the signal by using more filtering. Adding functionality for the harmonic structures of chords with embellishments such as 7th , Diminished, and Augmented chords to also be recognized would make this device much more useful. The final addition that can be made to the software which would increase the devices usability is to add functionality so that the system is trigger activated as opposed to the push button activation that exists now. There are two improvements that could be made to the device itself which would improve durability and portability. Adding a battery pack to the device which would replace the necessity of needing a power chord would improve the portability of the device and creating a housing for all the components would improve the durability of the system.

12. References

[1] “Electronic Tuners: History, Convienience and Limitation – Musekatcher's Blog – Flatpicker Hangout.” *Flatpicker Hangout.* N.P., n.d. Web.

[2] Masterarbeit. “Automatic Chord Detection in Polyphonic Audio Data.” (n.d.): n. pag. Web

[3] Fujishima, Takuya. “Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music.” *ICMC Proceedings 1999*(1999): n. pag. Web

[4] Hausner, Cristoph. “Design and Evaluation of a Simple Chord Detection Algorithm.” Thesis. Universität Passau, 2014. 8 Aug. 2014. Web.

[5] Lee, Kyogu, and Malcolm Slaney. “Acoustic Chord Transcription and Key Extraction From Audio Using Key-Dependent HMMs Trained on Synthesized Audio.” *IEE Xplore.* N.p., n.d. Web.

[6] Sillem, Robin. "Using Digital Signal Processing to Transcribe Polyphonic Music." *IEEE Xplore*. Proc. of Audio and Music Technology: The Challenge of Creative DSP, United Kingdom, London. IET, 18 Nov. 1998. Web. 14 May 2014. <http://ieeexplore.ieee.org/stamp/stamp.jsptp=&arnumber=757354>.

[7] Zenz, Veronika, and Andreas Rauber. *Automatic Chord Detection Incorporating Beat And Key Detection*. Proc. of 2007 IEEE International Conference on Signal Processing and Communications, United Arab Emirates, Dubai. *IEEE Xplore*. 14-17 Nov. 2007. Web. 14 May 2014.

[8] Zhu, Yongwei, Mohan S. Kankanhalli, and Sheng Gao. "Music Key Detection for Musical Audio." *Music Key Detection for Musical Audio*(2005). *IEEE*. Web. 14 May 2014. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1385971>.

[9] Lee, Kyogu, and Malcolm Slaney. "Acoustic Chord Transcription and Key Extraction From Audio Using Key-Dependent HMMs Trained on Synthesized Audio." *Acoustic Chord Transcription and Key Extraction From Audio Using Key-Dependent HMMs Trained on Synthesized Audio* 16.2 (2008): n. pag. IEEE. Web. 14 May 2014. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4432647>.

[10] https://learn.adafruit.com/adafruit-16x2-character-lcd-plus-keypad-for-raspberry-pi/