Spring 2020

Visual Music Assistant

David Klett
dgk10@zips.uakron.edu

Visual Music Assistant

Final Design Report (Fall Semester)

Larry Fritz
Bridger Garman
David Klett
Kyle Vasulka

Dr. R. J. Veillette

November 27, 2019

Table of Contents

Abstract **5**

1. Problem Statement **6**
1.1 Need LF BG DK KV 6
1.2 Objective LF BG DK KV 7
1.3 Background LF BG DK KV 7

2. Engineering Analysis **14**
2.1 Circuits BG KV 14
2.2 Electronics BG 15
2.2.1 Touch Screen BG 15
2.2.2 ADC BG KV 16
2.2.3 Mic DK 17
2.2.4 Diff amplifier BG 17
2.3 Signal Processing KV LF BG 17
2.4 Communications KV DK 30
2.5 Computer Networks KV 32
2.6 Embedded Systems DK 36
2.7 Raspberry Pi LF 37
2.8 Software Memory Usage LF KV 38
2.9 Software Speed Analysis DK KV LF 40
2.9.1 DSP DK 40
2.9.2 Raspberry Pi LF 41
2.9.3 HoloLens KV 42
2.10 Storage Capacity LF 42

3. Engineering Requirements Specification BG **44**

4. Engineering Standards Specification BG, DK **46**

5. Accepted Technical Design LF BG DK KV **47**
5.1 Hardware Design: BG 47
5.2 Software Design: LF KV DK 55

6. Team Information DK **111**

7. Parts List **111**

8. Project Schedules LF BG DK KV **113**

References **116**

List of Figures ii

Figure 1: Power Block Diagram 14
Figure 2: ADC Input voltage/Output Discrete Value 16
Figure 3: Python code making a summation of sinusoids of various frequencies 19
Figure 4: Summation of sinusoids real and imaginary 20
Figure 5: Frequency real and imaginary 21
Figure 6: Frequency real and imaginary 22
Figure 7: Code for summation of random frequency sin waves 23
Figure 8: Random frequency sinusoids in the time domain 24
Figure 9: Code to output guessed frequencies 25
Figure 10: Guessed output frequencies and actual frequencies 26
Figure 11: Signal Processing Flow Diagram. 28
Figure 12: Server client network configuration 33
Figure 13: Peer to peer configuration 1 34
Figure 14: Peer to peer configuration 2 34
Figure 15: Discrete Fourier transform (Wikipedia, 2019) 40
Figure 16: Level 0 Hardware Diagram 47
Figure 17: Level 1 Diagram 47
Figure 18: Level 2 Hardware Diagram 48
Figure 19: Hardware level 3 Diagram 49
Figure 20: Operational Amplifier Circuit and Waveform 51
Figure 21: Operational Amplifier Simulated Waveform 52
Figure 22: Operational Amplifier Symbol Schematic 53
Figure 23: Voltage Regulator Circuit 54
Figure 24: Software Level 0 Diagram 55
Figure 25: Software Level 1 Diagram 56
Figure 26: DSP Software Level 2 Diagram 57
Figure 27: HoloLens Software Level 2 Diagram 59
Figure 28: HoloLens Score Loop Software Level 3 Diagram 61
Figure 29: HoloLens MIDI File Level 3 Diagram 63
Figure 30: Raspberry Pi Level 2 Diagram 64
Figure 31: config.h 66
Figure 32: Header file config.h continued 67
Figure 33: main.c 68
Figure 34: Continuation of Main Program file: main.c 69
Figure 35: Continuation of Main Program file: main.c 2 70
Figure 36: Continuation of Main Program file: main.c 3 71
Figure 37:Continuation of Main Program file: main.c 4 72
Figure 38: Continuation of Main Program file: main.c 5 73

Figure 39: Watch windows during program operation 74
Figure 40: Header file fft.h 75
Figure 41: C file: twiddle_factors.c 1 76
Figure 42: Continuation of C file: twiddle_factors.c 77
Figure 43: Continuation of C file: twiddle_factors.c 2 78
Figure 44: Continuation of C file: twiddle_factors.c 3 79
Figure 45: Continuation of C file: twiddle_factors.c 4 80
Figure 46: Continuation of C file: twiddle_factors.c 5 81
Figure 47: Continuation of C file: twiddle_factors.c 6 81
Figure 48: TCP index.js 82
Figure 49: Continuation of TCP index.js 83
Figure 50: Continuation of TCP index.js 2 84
Figure 51: Example Program output 85
Figure 52: VMA.py 86
Figure 53: Continuation of VMA.py 87
Figure 54: Continuation of VMA.py 88
Figure 55: Example program output 2 89
Figure 56: Midi packets formulation 90
Figure 57: client.cs 91
Figure 58: client.cs continued 92
Figure 59: client.cs continued 93
Figure 60: client.cs continued 94
Figure 61: client.cs continued 95
Figure 62: client.cs continued 96
Figure 63: DisplayMIDI.cs 97
Figure 64: DisplayMIDI.cs continued 98
Figure 65: MidiFile.cs 99
Figure 66: Continuation of MidiFile.cs 100
Figure 67: MidiTrack.cs 102
Figure 68: Continuation of MidiTrack.cs 103
Figure 69: Continuation of MidiTrack.cs 2 104
Figure 70: Continuation of MidiTrack.cs 3 105
Figure 71: Continuation of MidiTrack.cs 4 106
Figure 72: MidiNote.cs 107
Figure 73: Level 0 block diagram (entire system) 108
Figure 74: Mechanical sketch of system KV 110
Figure 75: Build of material list 111
Figure 76: Cost of Build of material list 112
Figure 77: Gantt Chart 113

Figure 78: Updated Gantt chart (11/26/29, put in by David) 114

List of Tables iii

Table 1: List of Marketing Requirements 13
Table 2: Frequency Spacings Given the Fourier Transform Rate 27
Table 3: Repeats Required to Keep Frequency Spacing 29
Table 4: Music Data Types and Sizes 30
Table 5: Sizes of Python Data Types 38
Table 6: Design Requirement Specifications 44,45,46
Table 6: Safety standards and protocols 46
Table 7: Functional Requirements Table 1 108
Table 8: Functional Requirements Table 2 109
Table 9: Team Information
111

Abstract

Visual Music Assistant's (VMA) aim is to accelerate learning, increase accuracy of performance, and stimulate user recollection by teaching users how to play piano through modern technologies and techniques using augmented reality and real time feedback.

The VMA project teaches piano in a music lesson format. The VMA takes the form of a portable box connected to a midi piano and a Microsoft HoloLens. The VMA displays to the user a visual representation of a midi song file. A user played audio input is observed by a microphone and an A/D converter; The keynote frequencies played are then determined using a fourier transform algorithm and transposed into midi. The VMA streams notes that the user plays to the hololens and is scored against the midi song file notes. The VMA generates feedback scoring the users performance. The VMA scoring algorithm should increase the users' learning, accuracy, and recollection of a new music composition. By creating a new outlet to learn a musical instrument, the Visual Music Assistant will help engage individuals who have an interest in learning a new instrument by using the exciting field of augmented reality which adds a new channel to experience the world. By taking advantage of AR, this system will help enable the next generation of musicians and creatives.

Key Features:

- Augmented reality display of midi songs/notes
- Near real time scoring feedback
- Portabile box that works with midi and non midi pianos

1. Problem Statement

1.1 Need LF BG DK KV

For the last century, technology and music have particularly coexisted as two fields that lead to innovation and growth. Technology has lead to new ways of experiencing music, with also music itself acting as an incentive to develop better advances in technology, from signal processing to data representation in CD's.

One area in which advances are being made is the field of augmented reality, implementations such advances can be made to address the problem of learning a musical instrument, particularly playing piano. With growing costs of individual music lessons to accessibility of instructors, learning a new instrument can be challenging.

Exposing children to music at an early age has proven to have positive results: from enhancing their understanding to increasing self-esteem and motivation. Unfortunately, current participation in music learning is on the decline (ChildTrends). Technology use is increasing at a rapid rate. Also, there are many cases of individuals that want to learn a musical instrument but struggle with finding instructors that are available outside normal work hours. Individuals seeking to progress in playing a musical instrument may find it difficult when progression is not facilitated by an instructor. A device that logs performance can help to facilitate the progression of learning an instrument in the absence of an instructor. The cost of music lessons and the scarcity of local instructors is increasing (Hart). There is a need for an intuitive way for individuals to practice and explore music while increasing the understanding of what is being practiced.

1.2 Objective LF BG DK KV

Learning a new instrument can be quite an undertaking. Aspiring musicians have to navigate through music teachers that can accommodate the musician's schedule. There is a learning curve that aspiring musicians will encounter. Utilizing the new technology of augmented reality, this project aims to provide a gateway for learning instruments. There are many positive additions that the proposed device can offer including intuitive UI telling the user what notes to play, tracking individual user progress, real-time feedback that evaluates user input, and turns the learning process into a game to increase attention span and interest. For specified notes to play, the user interface will highlight which notes are to be played via the augmented reality headset. The system will know which notes to highlight by initial user calibration. The user calibration will include prompting the user to select the furthest notes on the keyboard.

1.3 Background LF BG DK KV

With today's technology, there is potential for an easier way to interface with musical instruments. This project aims to utilize augmented reality (AR) to innovate the music learning process. With the recent emergence of augmented reality technology, it has become possible to teach musical instruments in a modern learning style. The basic theory for the project is to utilize a signal sampling device (later denoted as music module) that converts the input of an instrument to a MIDI signal and transmits the signal to an AR headset. The music module will record and compare the played notes against what the user should be playing. The music module will then provide intuitive insight into the performance of the musician. Based on the note or chord that is played, visual cues will be created on the AR headset for the user to play along with. Examples

of cues that the AR system could implement would be the ability to highlight sections of a keyboard denoting a musical key to play in, label notes on a keyboard, provide a Synthesia-like view (Synthesia LLC), and display relevant chords for the user to choose from to play next. This theoretical system is both a new approach for beginners/novices to learn an instrument, as well as a new way to experiment with music theory.

Correlation between education and AR has existed throughout the technology's development (Rampolla, Kipper). Utilizing augmented reality while learning has been a focus for several researchers; specifically in developing an "educational framework with gamification to assist the learning process of children with intellectual disabilities" (Colpani/Homem). The use case that the researchers focused on was object identification and language learning, but this can be extended to include music learning. Another study integrated augmented reality technology and a game-based learning model to teach children English. The group observed in the study exhibited positive results that exceed learning without the use of AR (Chen).

Traditionally when learning a musical instrument an aspiring musician will take music lessons. Learning a new instrument can be done by hiring an instructor, by watching YouTube videos, by playing regularly without assistance, or by buying services like Synthesia (Synthesia LLC). Synthesia is a music playing software that runs on a computer and tells the user what note is pressed on a keyboard. The advancement of technology has promoted new ideas and facilitated innovations which make learning an instrument in new ways possible.

Currently, there are applications that sample what the user is playing through a microphone on the user's cellphone and provides feedback to the user. Yousician (Yousician) allows the user to follow a set chord progression while the app uses the input from the

musician's cell phone microphone to detect the notes that are being played. The notes that are observed by the app are then referenced to the set chord that was determined. The notes that are missing or played incorrectly are then visually addressed on the cellphone screen. Instant Musician is an application that displays visual cues in the form of a Synthesia- like (Synthesia LLC) key UI projected onto a piano through an AR phone device.

The limitations of current systems include requiring prior experience with the instrument to be utilized effectively. This requirement is partly due to the distance between the content displayed on the computer screen and the physical location of the instrument. A user with no prior experience with a musical instrument may struggle to translate note symbol to the physical one. The introduction of augmented reality to this scenario removes the requirement of prior experience by overlaying the note symbol (in the form of a virtual note) on top of the physical instrument. A current teaching style has the user look at the teacher and imitate the progression of notes played by the instructor. With the proposed design, the user will see the highlighted keyboard keys that should be pressed. The user will be able to easily follow along with the music piece after seeing these highlighted keyboard keys. Unlike current music lessons, where the aspiring musician meets with an instructor to learn the fundamentals of music theory, the proposed system will provide the fundamentals of music theory and suggest what notes could potentially be played next that are in the same key.

There are current limitations in technology surrounding chord/note recognition and the technology surrounding displaying chord/notes. These include the inability for current applications to provide recognition of chords/notes in a noisy environment, in scenarios where the instrument is distorted from a clean sound, and in situations where the chord/notes produced

are of low volume. Displaying chords/notes is limited by the device that the application is running on. There are already augmented reality music applications for cell phones like Instant Musician. AR is also not considered to be widespread in usage. (Colpani/Homem).

Limitations of the system would include the data transfer speed and latency between the headset and the frequency sampling device. These specifications would be dependent on the WiFi implementation included in each device. The IEEE 802.11ac WiFi standard is limited to a theoretical 500 megabits/second link. (Narayan) Another limitation of the proposed system is the amount of reference material in regards to developing augmented reality applications. There are very few sources to learn best practices for the development of augmented reality products. These technologies are still in the "stage of practical exploration." (Colpani/Homem)

For processing musical notes played on the keyboard, a Digital signal processing (DSP) chip will be used with an Analog to Digital converter (ADC) to create a Musical Instrument Digital Interface (MIDI) technical standard protocol. MIDI is a widely used standard across the music and audio industry (McGuire). The module will also have MIDI pass though for MIDI out compatible devices.

There is a similarity in how the AR headset will display musical notes in comparison to existing systems. Both existing technologies and the proposed system can be used as a learning tool. The market for linking new technologies to music education is a growing industry, with applications such as Yousician (Yousician) and Synthesia (Synthesia LLC) currently being introduced within the last ten years.

One difference of the proposed system compared to current products is the utilization of an augmented reality headset rather than a phone user interface. Another difference to the

proposed systems design is to process the frequencies of the notes to MIDI, and then display these notes on the musician's headset superimposing the notes on the keys of the keyboard and display the names of the keys while the musician plays. Current technologies only display current and future progressions of notes to be played. The system will dynamically process the notes being played to evaluate the accuracy and suggest additional ways to resolve chord/notes being played. This will potentially enhance improvisation as the musician explores the instrument. The proposed system will dynamically slow down the pace of hard measures in a music composition being played to allow the user to build confidence in the piece at a slower pace. Rather than just playing a preset video stream, the music module will adapt to a musician's weaknesses to improve the performance of selected music pieces. There are augmented reality applications on phones and tablets designed for pianos that parallel the proposed system in certain ways. These applications prompt the user to play a certain note, with the musical notes being represented as falling boxes that are aligned with the musical notes of the keyboard. The difference between the proposed system and these applications is that the applications force users to view content through the screen of a mobile device. Such a method provides a limited use case, for the user must manually hold the phone/tablet between themselves and the keyboard. This allows only one hand free for playing the keyboard. An AR headset would provide a hands-free user-friendly interface for viewing the notes.

There are patented technologies that utilize augmented reality and music performance, although no technologies are currently in the market that accomplishes music learning on an AR headset. A patent titled "Computer implemented method for providing augmented reality (AR) function regarding music track." goes into detail about receiving input information from a music

track and an instrument. The patent describes a process that determines attribute information of the music track based on the received input information. It receives the real-time content of audiovisual input signals using at least one capture device. It then generates visual information corresponding to a view regarding at least one of the user's limbs and an instrument. The data displayed to the user is comprised of AR instructions based on the attribute information of the music track. The patent is relevant to the proposed device because the device will also use (AR) to display musical information to the user.

In another patent titled "System for estimating user's skill in playing a musical instrument and determining virtual exercises thereof" the process for determining what virtual exercises a user should be given was discussed. This included the process concerning the functionality of processing entities and memory entities for processing and storing data. The system is configured to obtain musically notated data, and analyze it to assign the musical piece to which such data pertains a number of characteristics with scalar values to express the difficulty of a music piece. It provides the user with a number of musical pieces with known difficulty characteristics as virtual exercises to be completed by playing an instrument. The user's performance data is obtained for completed virtual exercises. The module will then analyze the user's performance data to determine and assign the user with a weight pertaining to the skill characteristics values in accordance with the difficulty values of the completed musical pieces and can suggest a musical piece for the user as a virtual exercise. This patent is relevant because the proposed device will also be doing live analysis on how the user is playing and correcting/instructing in near real time.

1.4 List of marketing requirements LF BG DK KV

Table 1: List of marketing requirements

1.	The system will identify which note and/or chords are being played while the user is playing.
2.	The system will provide an intuitive way to learn piano by showing notes and music theory concepts.
3.	The system will be able to interface with an 88 key piano.
4.	The system will be small and portable.
5.	The system will interpret audio output from the piano.
6.	The system will implement a scoring algorithm to compare user input to a determined musical composition

2. Engineering Analysis

2.1 Circuits BG KV

Power BG

To gain an understanding of the amount of power that is required for the Visual Music Assistant the complete system module needs to be analyzed individually in terms of power consumption. An approximation of the values for the components that make up the module has been conservatively assumed in deciding the required power to drive the module. The below calculations are the power requirements of each component of the visual music assistant's (VMA) compute module and audio possessing board components. Figure 1 shows the Power block diagram and the necessary steps needed to properly implement power for Visual Music Assistant module.

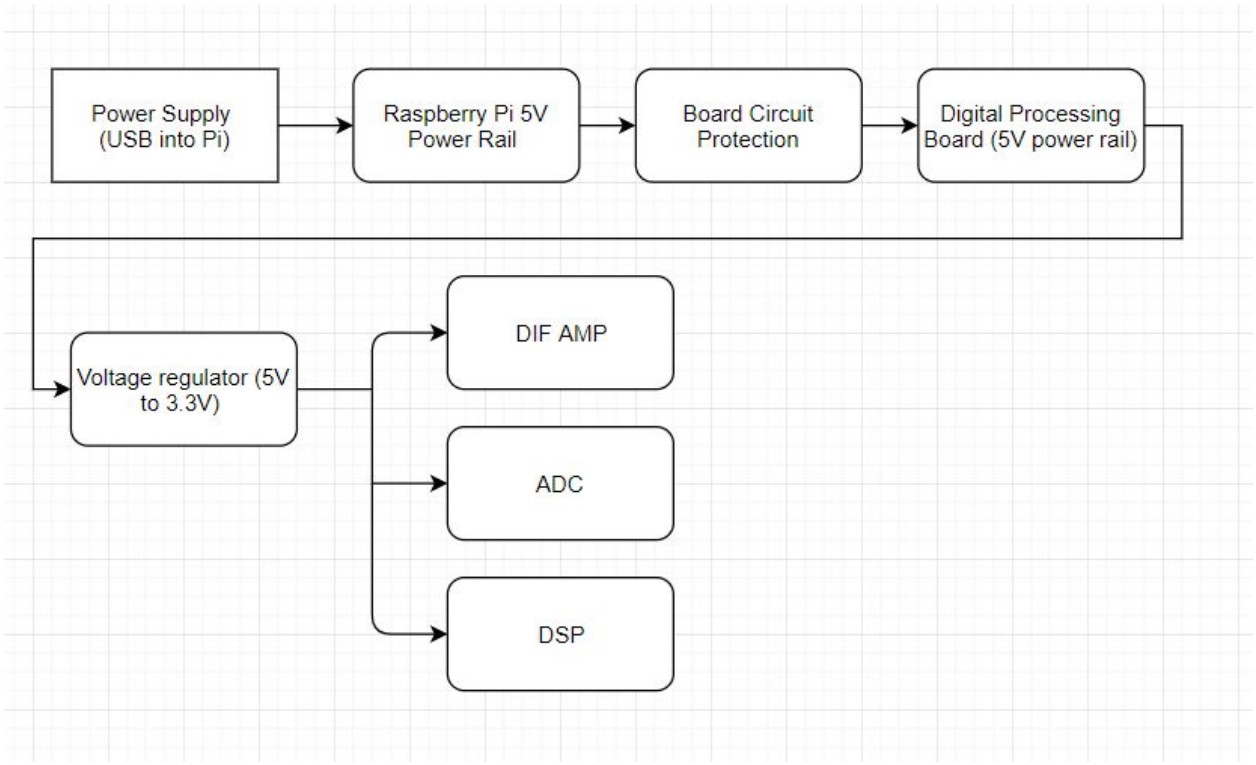

Figure 1: Power Block Diagram

The Power Supply will be a small power supply that has a power output of 1000 mW with a voltage discharge of 5V and can supply at least 2A. The Raspberry Pi 4 needs to be sourced with 5V, and the recommended input current is 2A. The circuit board protection needs to be able to protect against voltage spikes at/higher than 5.5V and needs to filter noise out from the Raspberry Pi 5V rail. The Voltage regulator needs to produce 3.3V from 5V while also introducing little to no noise to the 3.3V power rail. The differential amplifier needs to be able to produce a signal between ~ 0V and 3.3V off of a 0 to 5v reference voltage. The ADC needs to operate at a nominal 3.3V. The DSP needs to operate at a nominal 3.3V. The current draw of Raspberry Pi and components on the audio processing board will be less than 3A.

The Total Power Draw from the Raspberry Pi and Digital signaling board, including the DSP, the ADC, and the differential amplifier circuit are shown below.

- Nominal VDD input: 3.0 to 3.6V; typical voltage: 3.3V; typical current: 55μA
- ADC typical current draw: 200μA
- Typical Raspberry Pi4 current draw: 2.5A

$$Total\ current\ =\ 2.5A\ +\ 200\mu A + 55\mu A\ =\ 2.500255\ A$$

So a power supply with 5V in supplying at less 2.6A will meet the maximum power needs.

2.2 Electronics BG

2.2.1 Touch Screen BG

The user will supply login credentials through the VMA touchscreen to connect to a wifi network. This can be accomplished by creating a UI on the device that queries the user any necessary information.

(this will be powered by the Raspberry Pi)

2.2.2 **ADC** BG KV

The system will require an analog to digital converter in order to read in the analog audio signals and transform the input into the necessary digital bits for DSP calculations. The main properties of interest that affect the scope of the system are sampling rate, resolution, power constraints, and latency. The sampling rate should be able to cover the entire range of human hearing (20 - 20,000 Hz), therefore it should be a value higher than the Nyquist Frequency, or double 20,000 Hz. A common sampling rate for audio signal processing is 44.1 kHz, thus the ADC should be capable of a sampling rate of 44.1 kHz or higher. The resolution of the ADC (determined by the number of quantization levels) should have a minimum of 16 bits (2^16), which is a common value for audio applications (e.g. CD quality audio uses 16-bit samples). Due to the ADC's reliance on power through the Raspberry Pi, the ADC needs to operate at a nominal 3.3V. Concerning latency, the time between back-to-back samples is a common approximation, thus the ADC's latency will be(1/44.1 kHz) sec or less. This latency, although small, contributes to the overall latency of the system.

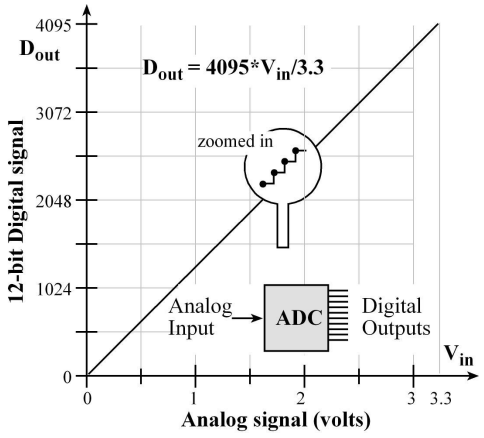

Figure 2: ADC Input voltage/Output Discrete Value

2.2.3 **Mic** DK

The system needs to have the ability to interpret audio output from the piano. In order to accomplish this, the system will need a microphone. The microphone will have to be highly sensitive in order to detect changing frequencies of sound produced by each of the piano's keys. It is known that there is a direct correlation between the impedance of a microphone and the interference associated with a microphone. Among those interferences are electromagnetic and radio interference, which worsens the SNR (signal-to-noise ratio) of the signal, thereby decreasing the audio signal's quality. Therefore, in order to obtain the best input signal, the microphone of the system should have low impedance (generally, low impedance when pertaining to microphones is 600 Ohms and lower).

2.2.4 **Diff amplifier** BG

A differential amplifier stage will be used to take the difference of the microphone signal and produce a signal with very low added distortion, low noise and with good circuit driving capability. Needs to have an input resistance that is 10^12 Ω or high to keep the microphone audio signal from being altered.

2.3 **Signal Processing** KV LF BG

A microphone will need to be used to transduce audio signals into digital representation of the notes that the users has played on the piano. The microphone will be sampled to take the Fourier transform and the resulting notes interrupted. The digital signal processor needs to sample audio signals ranging in frequency from 20Hz to 20KHz in order to cover the entire spectrum of human hearing ability. The Nyquist rate of the input signal will define how fast the

sampling rate F_s needs to be in order to capture the full content of the input signal without distortion.

Nyquist Rate: $F_s \geq 2B$ **where B is bandwidth of input signal**

Here, the Nyquist Rate is $F_s \geq 2*20\text{KHz} \Rightarrow F_s \geq 40\text{KHz}$

The Nyquist frequency is the minimum rate at which a signal can be sampled without introducing errors, which is twice the highest frequency present in the signal.

The average sampling rate of an mp4 file is 44100hz. Matching this encompasses the 40Khz Nyquist Rate required for human hearing. The Discrete Fourier transform is able to convert the time signal into a list of frequencies present at any given point in time. Sensing an 8th note in a song correlates to calculating the Fourier transform described above 8 times a second (8hz). The number of samples needed to take the fourier transform with the following formula:

Sample Rate / FourierTransform Rate = Number of samples

$$\frac{44100\ samples}{second} * \frac{1\ second}{8\ Fourier\ transforms} = 5512.5\ \frac{Samples}{Fourier\ transform}$$

The discrete fourier transform will have a frequency spacing of:

$$\frac{Sample\ Rate}{FFT\ size} = \frac{44100hz}{5512.5} = 8hz$$

Figure 3 shows fourier transform programmatic analysis:

[5]:
```
import math
import numpy
import matplotlib.pyplot
print(math.sin(math.pi))
testARR = []
testARRxAxis = []
sampleRate = 44100


t = numpy.linspace(0.0,1.0,num=4410)
f1=205 #hz
f2=208 #hz
f3=201 #hz
i = 0
while(i < 4410):
    testARR.append(math.sin(2*math.pi*f1*(i/4410))+math.sin(2*math.pi*f2*(i/4410))+math.sin(2*math.pi*f3*(i/4410)))
    testARRxAxis.append(i/sampleRate)
    i += 1
matplotlib.pyplot.figure(1)
matplotlib.pyplot.plot(t,testARR)
matplotlib.pyplot.xlim(0.3/f1)
#print(testARR)



f, fimag,fxaxis = FourierTransform(testARR,testARRxAxis)
matplotlib.pyplot.figure(2)
matplotlib.pyplot.plot(fxaxis,f)
```

```
1.2246467991473532e-16
```

Figure 3: Python code making a summation of sinusoids of various frequencies

Figure 4 shows the Program output of figure 3 outputs the summation of sinusoids and the fourier transform plotted on real and imaginary axis.

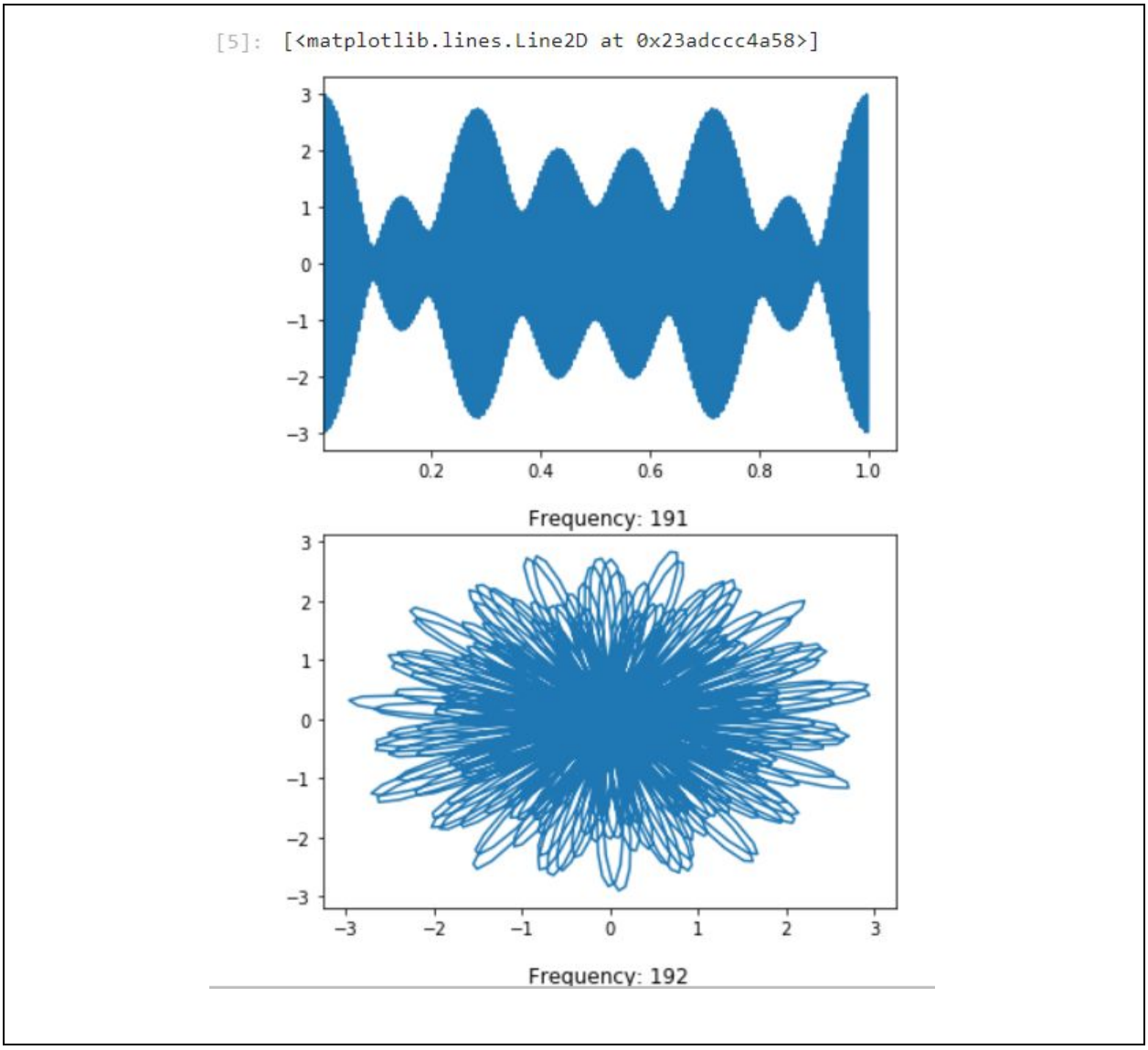

Figure 4: Summation of sinusoids real and imaginary

Figure 5 shows a continuation of the plotting of the fourier transform plotted at different frequencies on the real and imaginary axis.

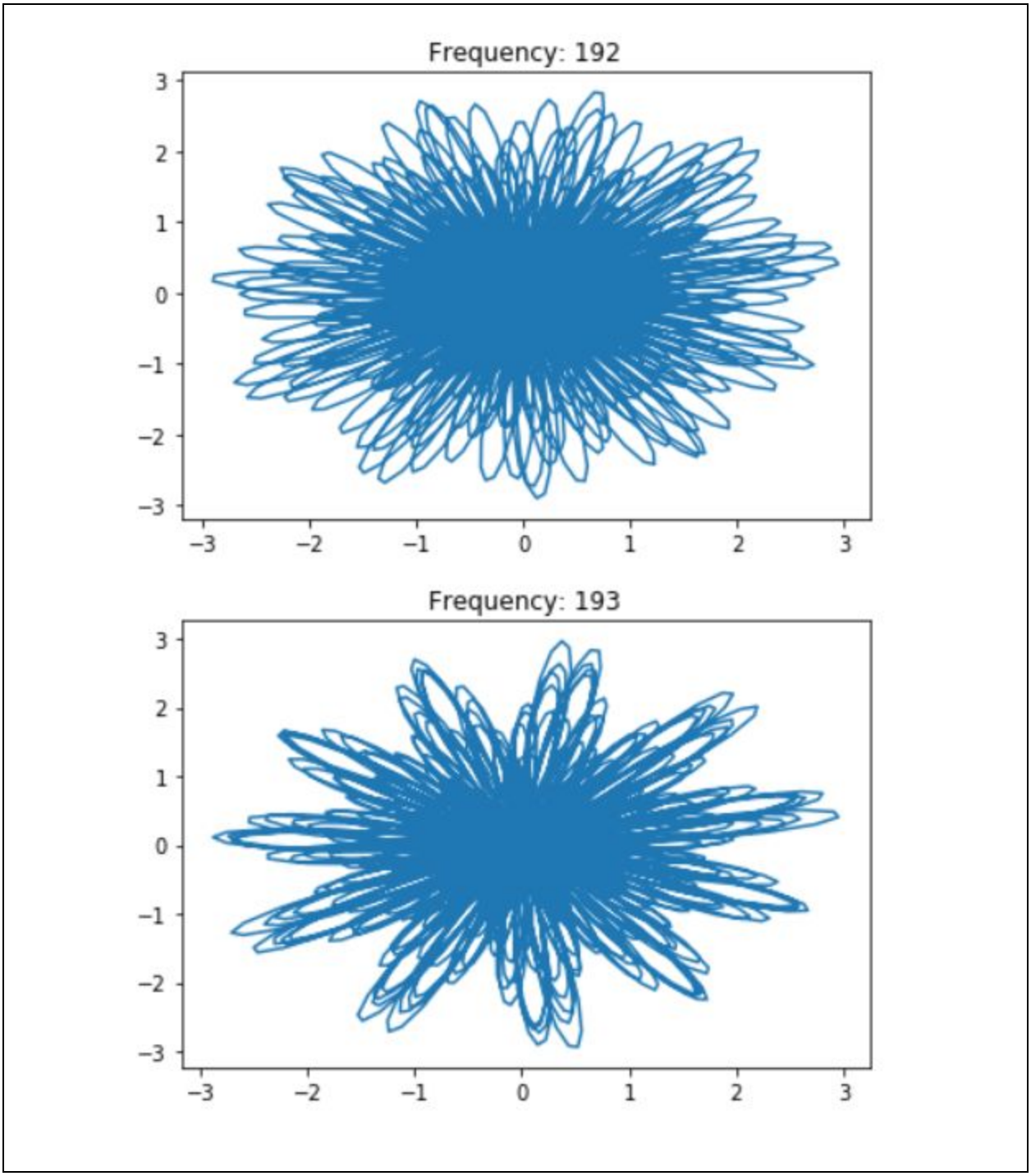

Figure 5: Frequency real and imaginary

Figure 6 shows a continuation of the plotting of the fourier transform plotted at different frequencies on the real and imaginary axis.

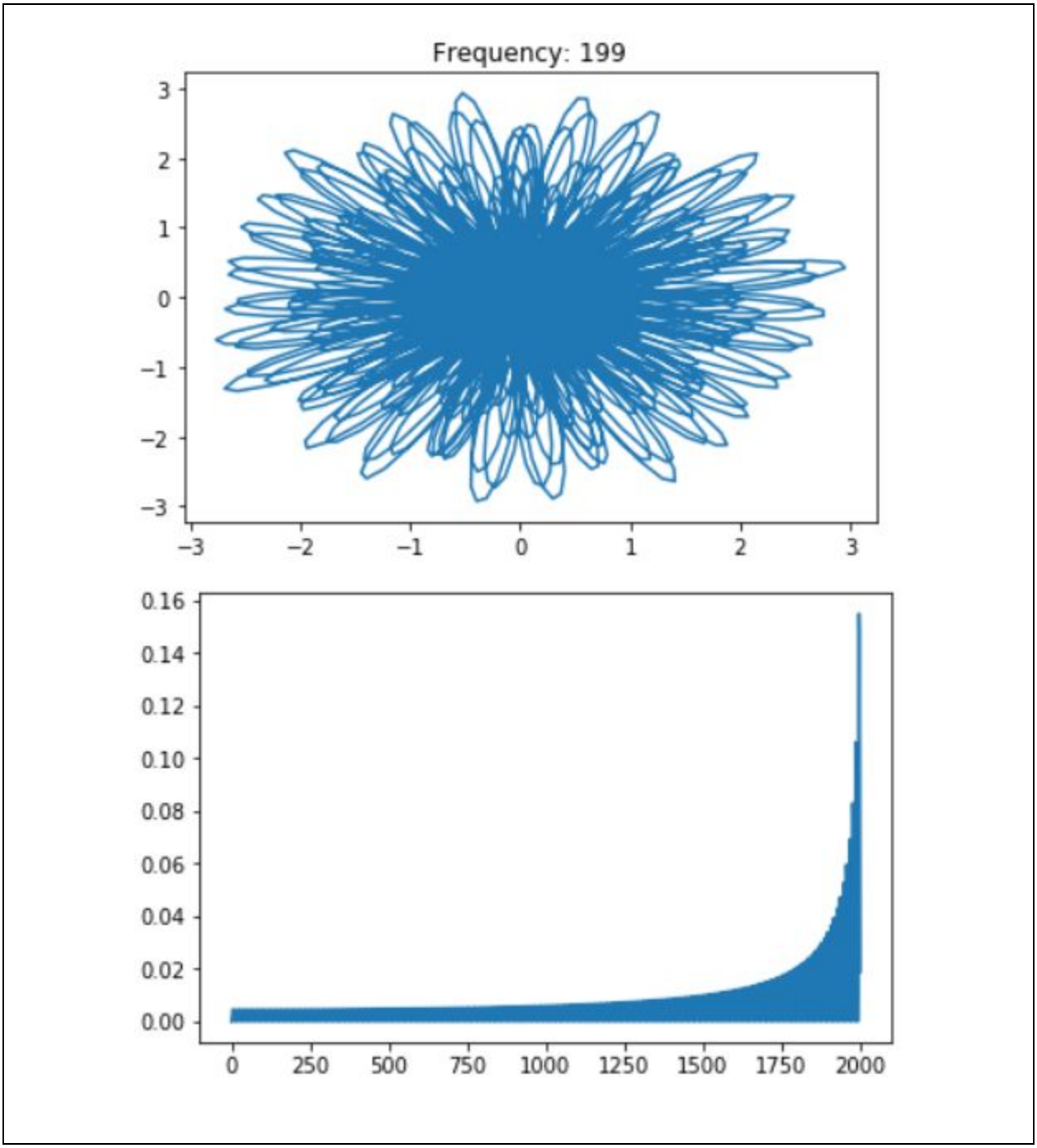

Figure 6: Frequency real and imaginary

Figure 7 shows python code that takes random frequencies and makes a signal that is the summation of their sinusoids with the intent of being able to distinguish the frequencies without knowing what they are.

```
[6]: import math
     import random
     import numpy
     import matplotlib.pyplot
     print(math.sin(math.pi))
     testARR = []
     numberOfTimeSamples = 42000

     t = numpy.linspace(0.0,1.0,num=numberOfTimeSamples)
     f=random.random()*2000
     f2=random.random()*2000
     f3=random.random()*2000
     i = 0
     while(i < numberOfTimeSamples):
         testARR.append(math.sin(2*math.pi*f*(i/4410))+math.sin(2*math.pi*f3*(i/4410))+math.sin(2*math.pi*f2*(i/4410)))
         i += 1
     matplotlib.pyplot.figure(1)
     matplotlib.pyplot.plot(t,testARR)
     matplotlib.pyplot.xlim(0,3/f)
     #print(testARR)

     1.2246467991473532e-16
```

Figure 7 :Code for summation of random frequency sin waves

Figure 8 shows a signal that is the summation of the random frequency sinusoids in the time domain.

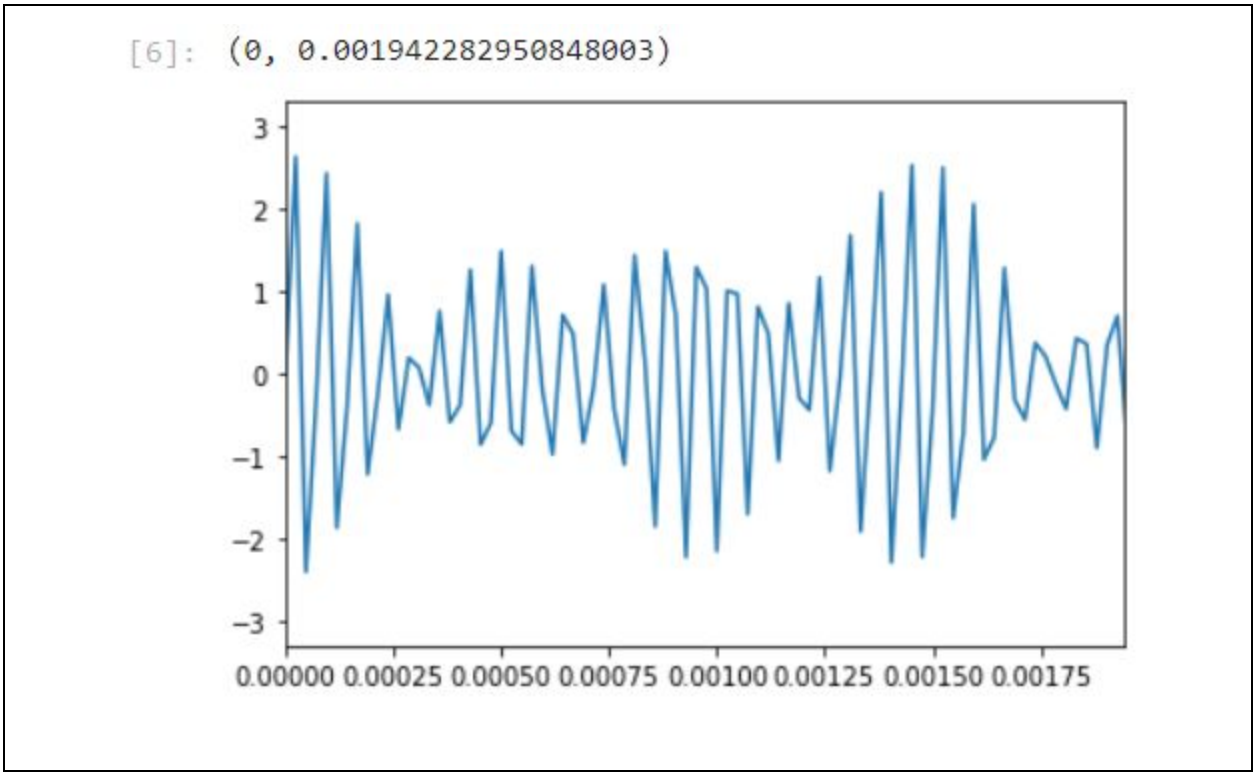

Figure 8: Random frequency sinusoids in the time domain

Figure 9 shows the fourier transform analysis as well as the quesses associated with a frequency that is over the threshold in the fourier transform.

```
[7]: freq = numpy.linspace(0.0,2206,num=numberOfTimeSamples/2 + 1)
freqTestArr = numpy.fft.rfft(testARR)
#matplotlib.pyplot.plot(t,numpy.fft.rfft(testARR))
matplotlib.pyplot.figure(2)
matplotlib.pyplot.plot(freq,abs(freqTestArr))
#matplotlib.pyplot.plot(freq,freqTestArr)


#matplotlib.pyplot.xlim(300,330)
print(freq)
```

Figure 9: Code to output guessed frequencies

Figure 10 shows the guessed outputs with respect to the actual frequencies.

```
Guess:  1327.4867619047618   (721.3402323181011+1087.2983579978886j)
Guess:  1327.5918095238094   (1117.7480480021932+1688.72841634317 46j)
Guess:  1327.696857142857   (2487.522135738234+3766.96269904174 46j)
Guess:  1327.8019047619048   (-10911.580958088525-16562.339570635184j)
Guess:  1327.9069523809524   (-1705.5933430563762-2594.898097262627j)
Guess:  1328.012   (-924.2310104339954-1409.409962620644j)
Guess:  1328.1170476190475   (-633.4428288453075-968.2274502446755j)
Guess:  1463.2082857142857   (558.7723196647335-936.2734442717021j)
Guess:  1463.3133333333333   (815.3988746333704-1364.2250579822085j)
Guess:  1463.4183809523809   (1509.96702240085-2522.500870108508j)
Guess:  1463.5234285714284   (10283.6671365256-17153.773260421054j)
Guess:  1463.628476190476   (-2133.5899354871826+3553.6138566343398j)
Guess:  1463.7335238095238   (-965.667389237836+1605.958417688577j)
Guess:  1463.8385714285714   (-623.693682270715+1035.679402067311j)
Guess:  1544.8302857142855   (686.1635942083146-781.6921677383534j)
Guess:  1544.9353333333333   (899.8687724294111-1024.3787258163693j)
Guess:  1545.040380952381   (1305.2507056014233-1484.735703839232j)
Guess:  1545.1454285714285   (2369.88427490517-2693.7467471027685j)
Guess:  1545.250476190476   (12699.651172103-14424.354362767099j)
Guess:  1545.3555238095237   (-3794.6684879189074+4306.7925684913225j)
Guess:  1545.4605714285713   (-1653.2898938112894+1875.01696402877 56j)
Guess:  1545.565619047619   (-1057.926995517626+1198.9153364070835j)
Guess:  1545.6706666666666   (-778.3891686487518+881.4684520701094j)
1544.574130504619
1327.1805230826292
1462.878053679036
```

Figure 10: Guessed output frequencies and actual frequencies

This program creates a summation of sinusoids of random frequencies and prints them in imaginary, and real spaces. Using this created signal, taking its fourier transform and compare each frequency to a set threshold value. It is possible to then print any frequency that is over the threshold value.

From this analysis it can be seen that the guesses(the printed frequencies over the threshold values) were extremely close to the actual values of the summation of sinusoids. The fourier transform is accurate enough for the application.

Table 2 shows a table of possible frequency spacings given a certain number of Fourier transforms.

Table 2: Frequency spacings given the Fourier transform rate

Fourier transforms per sec.	*Samples per fft*	*Frequency spacing*
1	44100	1*hz*
1.6352	26969.17	1.6352*hz*
2	22050	2*hz*
3	14700	3*hz*
5	8820	5*hz*
8	5512.5	8*hz*
10	4410	10*hz*
20	2205	20*hz*
50	882	50*hz*
100	441	100*hz*

The data above shows that the number of Fourier transforms pers sec determines the sampling resolution. It is also seen the shorter the time in which a Fourier transform is taken the larger the frequency steps.

An average 88 key piano ranges from 27.5Hz to 4186hz, Looking at several changes in frequencies(Δf) from one note to the next sequential note:

$$\Delta f_{1->2} = |f_1 - f_2| = |27.5 - 29.1352| = 1.6352hz$$

$$\Delta f_{43->44} = |f_{43} - f_{44}| = |329.628 - 311.127| = 18.501hz$$

$$\Delta f_{87->88} = |f_{87} - f_{88}| = |3951 - 4186| = 235hz$$

$$\text{Note that } \Delta f_{min} = 1.6352hz$$

An algorithm will be round the Fourier transform to the nearest note associated with a given frequency. The discrete fourier transform will need to have a minimum frequency spacing of Δf_{min}.

This means a sample size of:

$$\frac{Sample\ Rate}{Frequency\ Spacing} = Sample\ Size$$

$$\frac{44100}{1.6352} = 26969.17 \frac{Samples}{Fourier\ Transform}$$

The frequency can be padded with 0's or repeated until the desired samples are reached.

$$\frac{desired\ sample}{the\ sample} = \frac{26929.17}{5512.5} = 4.89\ repeats$$

Alternatively, the DFT of the last 26969 samples and do analysis of when notes came on vs left. However, this will likely cause any one note to persist for 4.89 FFT's

In summary, the sampling of the microphone is 44100hz,Taking the fourier transform of the repeated signal 8 times/second, checking if a frequency is being played by comparing the value to a threshold, and if so assign the frequency to the corresponding note.

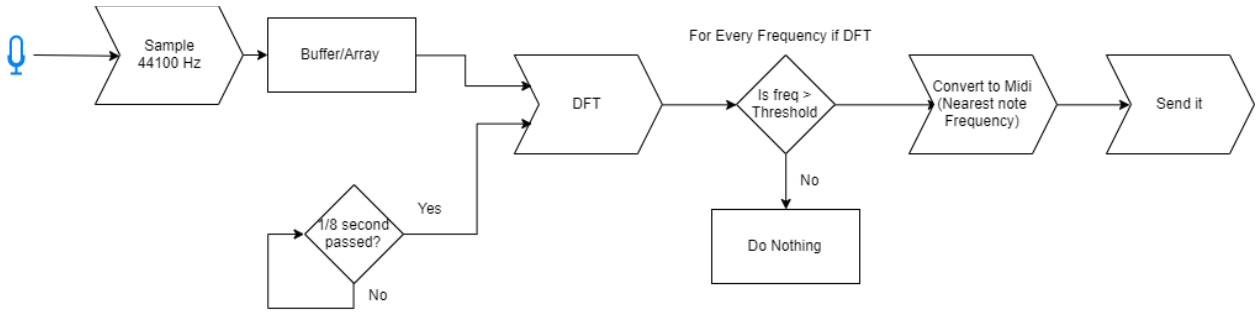

Figure 11: Signal Processing Flow Diagram.

The fourier transform and the microphone sampling affect the latency of the system.

$$t_{Delay} = t_{DSP} + t_{transmit} + t_{SBC} + t_{network} + t_{headset}$$

$$t_{DSP} = t_{sample\ microphone} + t_{fft}$$

Table 3 shows a table of possible latency added by *fft* given a certain number of Fourier Transforms.

Table 3: Repeats Required to Keep Frequency Spacing

Fourier transforms per sec.	*Latency added by fft* ($t_{sample\ microphone} + t_{fft}$)	*Frequency spacing*	*Repeats needed to have good spacing*
1	1000*ms*	1*hz*	0
1.6352	611*ms*	1.6352*hz*	0
2	500*ms*	2*hz*	1.223
3	333.33*ms*	3*hz*	1.834
5	200*ms*	5*hz*	3.057
8	125*ms*	8*hz*	4.892
10	100*ms*	10*hz*	6.115
20	50*ms*	20*hz*	12.230
50	20*ms*	50*hz*	30.577
100	10*ms*	100*hz*	61.154

There is a tradeoff of taking the Fourier transform many times a second to detect 16th and 32nd notes, and the spacing of the Fourier transform.

2.4 Communications KV DK

Music data will be sent from the DSP to the Raspberry pi SBC over UART. The main concerns are the data transfer speed and the size of the data that is transmitting. The size of the data that changes based on several factors. If the piano supports MIDI then it is possible to convey the MIDI events from a piano directly to the raspberry pi through MIDI over USB.

If the piano has only sound, the VMA will sample the sound using the microphone, take the fourier transform, and analyze what frequencies are present in a given sound sample; Also known as music data.

Music data for this purposes is represented in either a list of frequency likelihoods, list of notes currently being played/Change in notes being played, or midi signals. The music data will be transmitted >= 8 times per second. The following table showcases the different file types and sizes thereof.

Table 4: Music Data Types and Sizes

	Type	Size
List of Frequency likelihoods	Constant	8318 bytes
List of Frequencies above the threshold	Variable	0 - 88 bytes
List of notes currently being played	Variable	0 - 88 *3 bytes
List of Change in notes being played	Variable	0 - 88 * 3 bytes
Binary Representation of Frequencies above Threshold	Constant	$Ceil(Log_2(4186)) = 13$ bits
List of Midi Events	Variable	3 bytes * 88 keys = 264 bytes

The size of the music data greatly changes based on the number of notes being played

The general maximum number of notes being played on a piano in a song at a given time is:

On average: 3 notes, Upper average:15 notes , Maximum: 88 notes.

In the worst case where the user plays all 88 notes simultaneously, the VMA will need to transmit the equivalent amount of data over the wire.

List of frequencies played on a piano range from 27.5hz to 4186hz.

Number of integer frequencies stored in freqArr = ceiling(MaxPianoFreq-MinPianoFreq)

ceiling(4186-27.5) = 4159 integers

Total size of freqArr = size of integer * number of integers

2 bytes * 4159 = 8318 bytes

This is an array of 4159 integers.

Note that this contains information on how likely each frequency is currently playing. This is $ceiling(log_2(4159))$ bits. Sending the midi signal equivalent of this is:

(Midi Event Size) x (Number Of Midi Events)

(3 bytes) x 88 keys = 264 bytes

Note that the average song does not involve the user pressing/releasing all of the notes simultaneously.

Max throughput array of freq = 8318 bytes, Max throughput midi files = 264bytes

As such this communication should support the worst-case throughput.

UART baud rate for 115200 Baud (by definition), will give you 115200 * 5 = 576000 bits per second.

UART Calculations:

In order to incorporate UART onto DSP Microchip products, two known factors are needed: (1) Desired Baudrate, (2) the clock cycle speed of the device. With these given, the baud rate generator register can be populated with the right value. The following calculation found from Equation 3-1 in the UART chapter of the dsPIC33/PIC24 Family Reference Manual describes this process:

$$Baud\ Rate = \frac{F_P}{16 \times (UxBRG + 1)} \quad \text{..... (1)}$$

$$UxBRG = \frac{F_P}{16 \times Baud\ Rate} - 1 \quad \text{..... (2)}$$

For the system, the dsPIC33 microcontroller is currently running at 40 MIPS (million instructions per second), thus F_P = 40,000,000. For a commonly desired baud rate of 115200 (bytes/sec), the UxBRG register (in this case, the VMA is using UART module 2, so U2BRG) can then be calculated with the following:

$U2BRG = \frac{F_P}{16 \times Baud\ Rate} - 1$ = $\frac{40{,}000{,}000}{16 \times 115200} - 1$ = 20.7014... ≈ 21 (round up due to register restrictions)

2.5 Computer Networks KV

The networking goals include sending data in near real time from a compute module to the augmented reality headset. Several techniques have been compared including UDP, TCP, and HTTP with several network communication configurations. Three configurations are shown below:

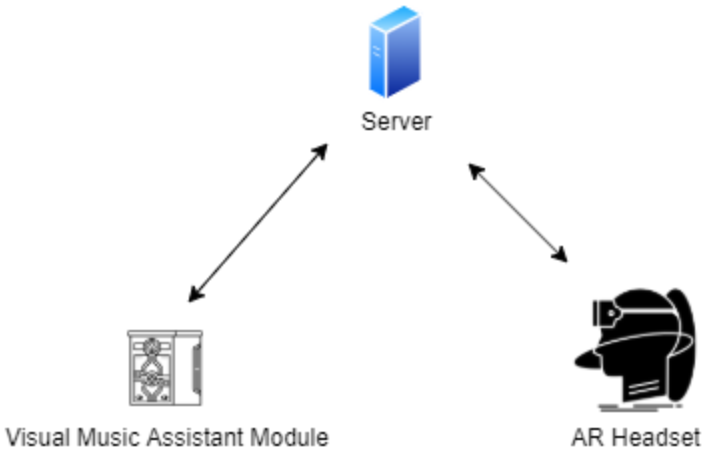

Figure 12: Server client network configuration

Figure 12 showcases the server-client network configuration model. Having a separate, dedicated server that both devices talk to will always be connected to the internet. This would enable issuing over the air updates and be connected to the internet, giving access to any api's that the VMA would use as well as enable cross communication with internal servers that can be used to tweak user experience settings on the fly. A Server client network also increases the latency of the information traveling between the visual music assistant and the AR headset.

An alternative to the server client network configuration is the peer to peer network configuration. Due to the visual music assistant being connected to the AR headset on a local network, the network latency is significantly reduced. One challenge introduced with the peer to peer networking solution is it introduces IP configuration issues where the client and server do not know where each other are on the network and will have to actively search for one another when connecting to the network. Additionally, it can not be guaranteed that the system is connected to the internet, making several quality of life features inaccessible.

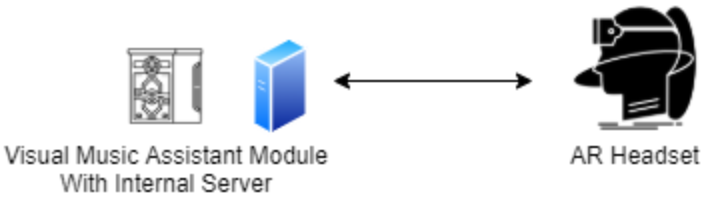

Figure 13: Peer to peer configuration 1

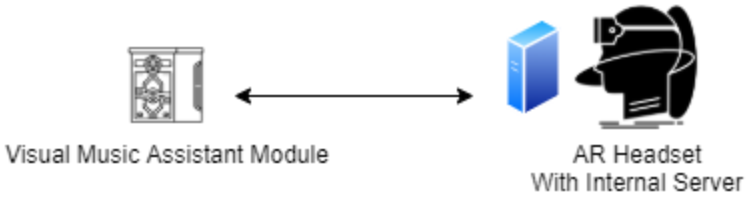

Figure 14: Peer to peer configuration 2

Figure 13 and 14 show that the Server can be located on either the visual music assistant module or the AR headset.

Additionally, sending information from one client to another client using UDP or TCP sockets is possible. TCP sockets automatically have error checking and guarantees that all the packets will arrive in order. UDP does not have error checking, and is much faster to transmit. This is normally used in streaming applications. The project has needs in both technologies. It is also possible to utilize HTTP requests to communicate information about state.
Network latency contributes heavily to the overall latency.

The overall latency is related to the perceived visual sound function of the human ear: An average human can start to detect disconnect in sound when the latency of the expected sound

exceeds approximately 100ms. As such this is the target latency of both displaying information to the user as well as playing sound for the user.

The total latency is

$$t_{Delay} = t_{DSP} + t_{transmit} + t_{SBC} + t_{network} + t_{headset}$$

$$t_{RPi} = t_{midi\ conversion} + t_{json\ conversion} + t_{encoding}$$

$$t_{headset} = t_{audial/visual\ sync}$$

$$t_{Delay} = t_{sample\ microphone} + t_{fft} + t_{midi\ conversion} + t_{transmit} + t_{RPi} + t_{network} + t_{headset}$$

$$t_{Delay\ Goal} \leq 100ms$$

MIDI Message transfer protocol:

Each message sent should be a json midi object terminated by an end line character. This allows for multiple packets to be packaged together and be sent over a network and still be distinguishable.

2.6 Embedded Systems DK

The need for high throughput, ensured worst-case latency, and stability are all characteristics of real-time systems, of which audio signal processing is a prime example. Digital signal processors (DSP's) are specifically designed and built to meet real-time system needs on an embedded hardware level. The hardware of a DSP is optimized for high-speed computation in several ways, generally including the following: multiply-accumulate hardware that is integrated into the main data path of the DSP processor (allowing a single cycle for a multiply-accumulate operation); two or more multiply-accumulate units (allowing several multiply-accumulate operations to run in parallel); ability for parallel direct memory access (allowing multiple direct accesses to memory in a single clock cycle); one or more address generation units, commonly referred to as AGU's (these units operate in the background, forming the addresses required for operand accesses without using the main data path of the processor, thereby running in parallel with the execution of arithmetic instructions); one of the address modes for an AGU generally found is bit-reversed addressing (this increases the speed of certain FFT's); repeat loop register bits (bypassing the need to update a loop counter and wasting an instruction cycle, thereby providing an increased for-next loop execution time). These hardware design optimizations make a DSP a "great fit" (change later) for audio signal processing, which involve heavy use of the Fast Fourier Transform and other fast operations.

The digital signal processor needs to be able to sample audio signals ranging in frequency from 20Hz to 20KHz in order to cover the entire spectrum of human hearing ability. The Nyquist rate of the input signal will define how fast the sampling rate F_s needs to be in order to capture the full content of the input signal without distortion.

Nyquist Rate: $F_s \geq 2B$ **where B is bandwidth of input signal**

Here, the Nyquist Rate is $F_s \geq$ 2*20KHz => $F_s \geq$ 40KHz

The Nyquist frequency is the minimum rate at which a signal can be sampled without introducing errors, which is twice the highest frequency present in the signal. Thus, all calculations on the DSP need to be calculated within the timeframe of each sampling rate. In other words, the system must be able to operate all calculations on a single sample in 0.000025 seconds (1/40k).

2.7 Raspberry Pi LF

The networking module on the Raspberry Pi will require at least three functions. The first function polls a MIDI keyboard over USB and reads in MIDI events. This function will require an object to represent the MIDI input stream, and an array to hold the data retrieved from the stream. The second function converts an array containing MIDI data to a byte array containing the JSON representation of the array with an endline character appended to it. It will require a string to hold the JSON representation of the MIDI data, and a byte array to hold both the MIDI data and the endline character. The third function will open up a socket connection to a server, relay any MIDI data passed into it, and store any information received from the server. It will require a socket object to represent the socket connection, a string to represent the server IP address, an integer to represent the server port, an integer to represent the buffer size, and a byte array to represent the data received from the server.

2.8 Software Memory Usage LF KV

Python Type Sizes

Table 5: Sizes of Python Data Types

Data Type	**Size (bytes)**
String	49
Integer	28
1D Array	60 + (2 * Number of Elements)
2D+ Array	72 + (2 * Number of Elements)
Byte Array	56 + (2 * Number of Elements)
MIDI/Socket Object	104 bytes
Socket Received Data	1024 bytes

The memory usage of a program can be estimated by summing up the size of the variables used in each function of the program. Below are the calculations for the sizes of functions 1, 2 and 3 respectively (S_{f1} , S_{f2} , S_{f3})

$$S_{f1} = (1 * MIDI\ Object\ Size) + (1 * (\ 2D\ Array\ Minimum\ Size\ +\ 2 * Array\ Length))$$

$$= (1 * 104\ bytes) + (1 * (72\ bytes + 2 * 4\ bytes))$$

$$S_{f1} = \mathbf{184\ bytes}$$

$$S_{f2} = (1 * String\ Size) + (1 * (Byte\ Array\ Minimum\ Size + 2 * Byte\ Array\ Length))$$

$$= (1 * 49\ bytes) + (1 * (56\ bytes + 2 * 5\ bytes))$$

$$S_{f2} = \mathbf{115\ bytes}$$

$$S_{f3} = (1 * Socket\ Size) + (1 * String\ Size) + (1 * Integer\ Size) +$$

$$(1 * (Byte\ Array\ Minimum\ Size + 2 * Byte\ Array\ Length))$$

$$= (1 * 104\ bytes) + (1 * 49\ bytes) + (1 * 28\ bytes) +$$

$$(1 * (56\ bytes + 2 * 1024\ bytes))$$

$$S_{f3} = \mathbf{2285\ bytes}$$

Total memory usage of Python program = $S_{f1} + S_{f2} + S_{f1}$

= 184 bytes + 115 bytes + 2285 bytes

= 2584 bytes = **2.584 KB**

2.9 Software Speed Analysis DK KV LF

2.9.1 DSP DK

The digital signal processor needs a portion of the target latency time to collect samples from the microphone, calculate the Fourier transform of the samples collected, calculate the significant frequencies from the Fourier transform, and to convert the significant frequencies to MIDI data.

$$t_{DSP} = t_{sample\ microphone} + t_{fft}$$

The number of operations of the Fast Fourier Transform can be estimated by $Nlog_2 N$, with N being the number of samples. The value of N for a 16-bit ADC (common in audio applications would be 2^{16} or 65536 samples. Thus, the number of operations for an FFT algorithm operating on 65536 samples is $65536 * log_2 65536$ or 1.048 million instructions. In order to calculate the speed of the FFT on a DSP using 40 MIPS (million instructions per second), the execution time formula is used:

$$\text{Execution time}(T) = \text{CPI} \times \text{Instruction count} \times \text{clock time}$$

The CPI of the Fourier Transform calculation is the average cycles per instruction. To find this, the FFT equation from Figure 15 was analyzed:

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn} \\ &= \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N} kn\right) - i \cdot \sin\left(\frac{2\pi}{N} kn\right) \right], \end{aligned}$$

Figure 15: Discrete Fourier transform (Wikipedia, 2019)

This formula includes the following operations: 1 multiply, 1 cosine, 2 divides, 1 subtraction, and 1 sine operation. Assuming each sine and cosine operation is calculated using a 4th degree Taylor series (Texas Instruments, 2002), a sine or cosine operation includes 4 additions, 4 divides, and 4 multiplications. Thus, there are a total of 9 multiplications, 10 divisions, and 9 additions/subtractions per FFT summation. Referencing the instruction architecture of a dSPIC33 (Microchip Technology, Inc, 2008), an addition/subtraction takes 1 clock cycle, a division takes 18 cycles, and a multiplication takes only 1 cycle (via MAC unit). The average CPI is thus:

$$\frac{9*1 + 10*18 + 9*1 \; cycles}{9 + 10 + 9 \; instructions} = 7.07143 \text{ CPI (cycles per instruction).}$$

Plugging the average CPI calculated, the 1.048 million instructions per FFT, and a clock time of 40 MHz into the execution time formula gives a value of:

$$Execution\ time = \frac{1\ second}{40\ MHz} * (7.07143 \text{ CPI}) * 1.048 \text{ million (instruction count)} = 0.185271 \text{ sec}$$

This time exceeds the time delay goal of 100 ms, thus in order to decrease the FFT calculation time, the digital signal processor's clock speed must be higher than 40 MHz and/or the sampling count must be lower.

2.9.2 Raspberry Pi LF

The Raspberry Pi needs a portion of the target latency time to convert incoming frequency data to the MIDI format, convert MIDI data to JavaScript Object Notation (JSON), and to convert the JSON data to a byte array that is suitable for transfer over TCP/UDP.

$$t_{RPi} = t_{midi\ conversion} + t_{json\ conversion} + t_{encoding}$$

To correctly pick out the frequencies being played, the program running on the Raspberry Pi will need to iterate over an array that is at worst *N,* the length of the maximum playable frequency of the piano. To map the frequencies being played to the correct musical note, the

program will have to do a lookup in a binary search tree that contains each musical note and its corresponding frequency. This will take, at worst, O(log(N)) time. In most cases the time to match all notes being played will be significantly less because the user will be playing at most 10 notes at a time. The number of operations required for picking and matching frequencies can then be found as:

$$N + N * Log(N) = 4000 + 4000 * Log(4000) = 18408 \; operations$$

The amount of time the program will take to process 88 notes being pressed can then be calculated using the instructions per second measurement of the Raspberry Pi, which is around 4744 MIPS.

$$t_{match} = \frac{18408 \; instructions}{4744M \; instructions/second} = 3.8 \; uS$$

2.9.3 HoloLens KV

The HoloLens rendering engine should be kept to a minimum of 60 FPS as per the microsoft documentation. This is primarily handled internally by the unity rendering engine, which can be seen through the use of various tools provided such as the GPUView, Visual Studio Graphics Debugger, and the profilers built into 3D engines such as the Frame Debugger in Unity.

2.10 Storage Capacity LF

The MIDI format is relatively small in terms of storage size because it is a representation of musical notes that make up a song rather than a full audio signal. Therefore the typical size of a MIDI file is on the order of kilobytes (KB) rather than megabytes (MB). Below is the calculation for the storage size of a library of MIDI files, assuming a max MIDI file size of 100KB (an overestimate, as a 4 minute song is ~55KB) and a MIDI file count of 100 songs:

$$Storage \; size = 100 \frac{KB}{song} * 100 \; songs = 10000KB = 10GB$$

The required storage size for a MIDI file can potentially be reduced for this application, as only the Note On/Note Off MIDI events and their timestamps need to be recorded. All other events can potentially be ignored.

2.11 MIDI File Parsing LF

The MIDI file format is a sequence of bytes that is structured into a header chunk, and then one or more track chunks following the header chunk. The header chunk of a MIDI file contains the MIDI file format, the number of tracks following the header, and the time division of the file; each header field is always 2 bytes long. The track chunks following the header chunk contain a sequence of events that are either Meta, Sysex, or MIDI events. Meta events consist of a sentinel byte “FF” followed by a meta event identifier byte, followed by a variable length field, followed by the data for the Meta event. Meta events contain data for information like track names, lyrics, and instruments used for each track. Sysex events consist of a sentinel byte “F0” or “F7” followed by a variable length field, followed by the data for the Sysex event. MIDI events can have a wide range of identifier bytes, from “8x” to “Ex.” Most MIDI messages contain information regarding the playback of a song or the notes pressed on a keyboard. The two that are most important to this project are the Note On and Note Off MIDI events, which are indicated with the identifier bytes “9x” and “8x” respectively.

3. Engineering Requirements Specification BG

The following lists the design requirements for the Visual Music Assistant, and the marketing requirements to which they pertain. The list calls out the marketing requirement that the design requirement statement addresses. The design requirement statement is then given a justification pertaining to the given marketing requirements that are intended to be met.

Table 6: Design Requirement Specifications

Marketing Requirements	Design Requirements	Justification
1,5	Digital Signal Processing circuit will need to sample audio signal at a rate above 44100hz.	Identifying the notes and chords will require a sampling rate above nyquist sampling rate
2	Visual Music Assistant will be able to teach the melody of"*Twinkle, Twinkle, Little Star*" to the user in under 5mins.	Will be able to evaluate whether VMA can provide a way to learn keyboard notes and music concepts
1,3	Visual Music Assistant will be able to detect notes played in the range of 27.5hz to 4186hz	The frequency range of a 88key piano is 27.5 to 4186hz. VMA must be able to detect these frequencies
4	Visual Music Assistant will be able to fit in a backpack and weighs less than 3lbs.	The criteria of small and portable are constrained to something that is easily carried by one person
1,5	DSP needs to have a Latency added to the system of less than 30ms	Systems needs to provide feedback of the interpreted note while the user is playing
1,5	Overall latency for audio input to user interface output should be less than 100ms	Systems needs to provide feedback of the interpreted note while the user is playing
2	Visual Music Assistant will have a user interface that	Intuitive feedback that is relevant to the user will need

	gives real-time feedback of the users performance in less than 200ms	to be provided while the user is playing
2	Visual Music Assistant will have a scoring algorithm that will provide visual feedback in Augmented Reality	Addresses the need for an intuitive way to learn instruments by showing notes and music theory concepts.
2,3	Visual Music Assistant will be able to display upto 88 augmented reality keys.	For the VMA system to interface with the 88 keys on a keyboard, AR keys will need to be mapped to each key and be distinguishable by the user
6	Visual Music Assistant will be able interface with midi protocol music files and then translate the midi music files into a 3D interpretation.	Addresses the need for the scoring algorithm to compare a musical composition with the user's input.
1,2	Visual Music Assistant will superimpose an audiovisual indication if a note is misplayed and/or played correctly	The notes will be identified as a correct note or incorrect note by VMA system to promote learning the instrument intuitively
1,5,6	Visual Music Assistant will be able to tranduces audio signals into midi protocol	Addresses the need for compatibility between user input when compared with the corresponding musical composition

1. The system will identify what notes and/or chords the user is playing.
2. The system will provide an intuitive way to learn instruments by showing notes and music theory concepts.
3. The system will be able to interface with 88 key piano.
4. The system will be small and portable.
5. The system will interpret audio output from the piano
6. The system will implement a scoring algorithm to compare user input to a determined

musical composition

4. Engineering Standards Specification BG, DK

Table 7: Safety standards and protocols

	Standard	Use
Safety		
Communications	TCP, UART, WIFI (IEEE 802.11)	Networking between the Raspberry Pi and the Microsoft Hololens
Data Formats	MIDI, JSON (**IEEE**-1394)	Serial communication for datafiles,and Raspberry Pi to Hololens
Design Methods	TCP standards (**IEEE** 802.2)	Transmission Control Protocol between Raspberry and Hololens Pi
Programming Languages	C, C#, Python, JavaScript (**ISO/IEC** 9899:1999) (**ISO/IEC** 23270:2003)	Used in the production of Visual Music Assistant code
Connector Standards	USB C, USB A (**IEEE**-1394)	Use on Raspberry Pi and on Digital signal processing board

Table 7 shows the protocols and safety standards that are considered in the creation of Visual Music Assistent

5. Accepted Technical Design LF BG DK KV

5.1 Hardware Design: BG

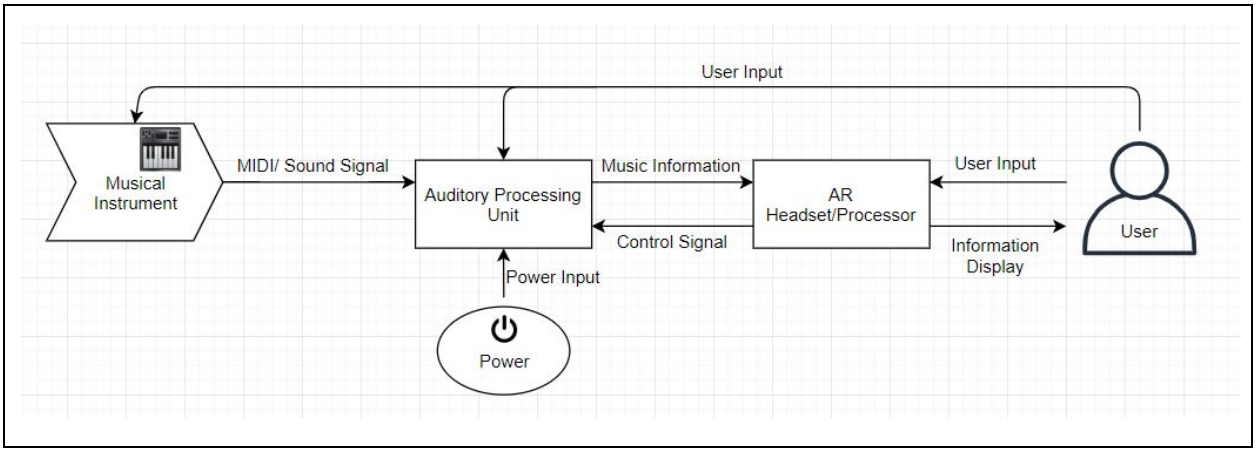

Figure 16: Level 0 Hardware Diagram

Figure 16 shows the Level 0 Hardware Diagram. The system will receive a signal from the musical instrument. The signal received will be processed by the Auditory processing unit. The Auditory processing unit will send the distinguished frequency to the AR Headset. The User will interact with the AR headset and the auditory processing unit. The user will configure the Auditory processing unit to be able to connect to AR headset. The system will be powered via 5V USB.

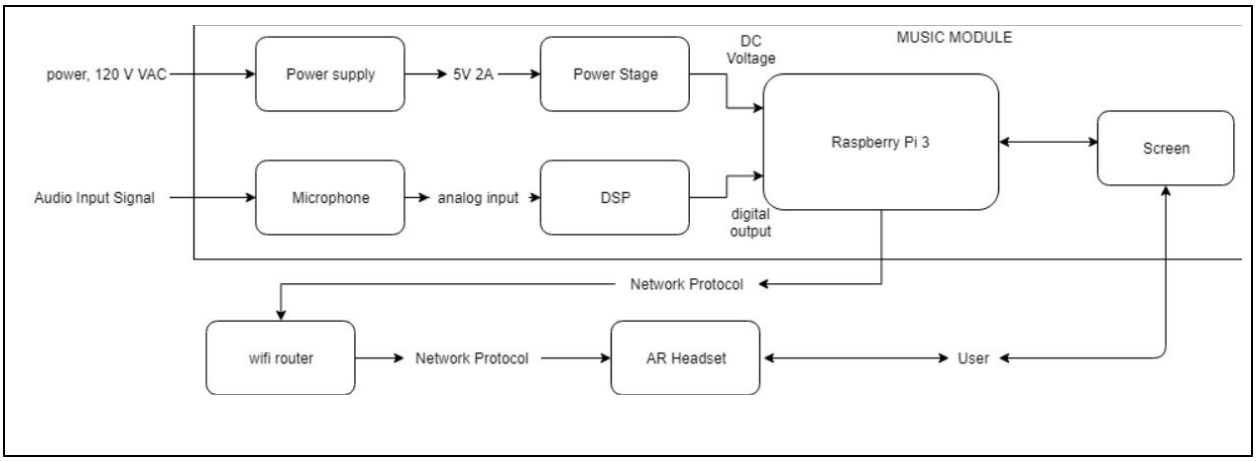

Figure 17: Level 1 Diagram

Figure 17 shows the Level 1 Hardware Diagram. The system will receive power from an outlet and will be rectified into 5V @ 2A power. This power will be delivered to the Raspberry Pi. The Raspberry Pi will receive the distinguished frequency from the DSP chip. The DSP chip will receive an analog input from the microphone circuitry. The microphone circuitry will take the transduces audio signal from the microphone and create a signal for the DSP to use. The distinguished frequency that is sent to the Raspberry Pi will then be made into a protectal that will be sent to the AR headset via WIFI or Bluetooth. The user will interact with the AR headset and the Screen. The screen will be used to configure the Music Module.

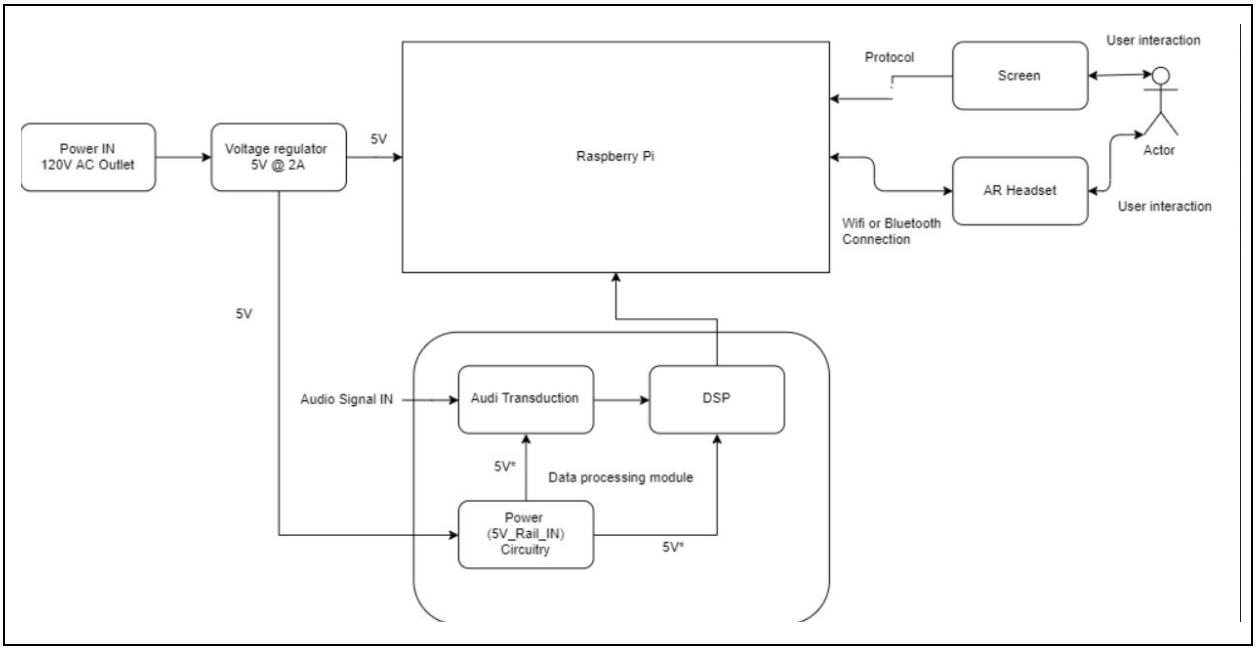

Figure 18: Level 2 Hardware Diagram

Figure 18 shows the Level 2 Hardware Diagram. The system will receive power from an outlet and be rectified into 5V @ 2A power. This power will be delivered to the Raspberry Pi and to the Data Processing Module. The data processing module will receive an Audio Signal from the keyboard. The Auditory Transduction circuitry will then produce a digitized signal for the DSP

chip to run a fast fourier transform on. The determined frequency that was interpreted by the DSP will then be sent to the Raspberry Pi to be converted to MIDI. The MIDI signal will be sent over WIFI or Bluetooth wireless protocol to the AR headset. The frequencies will then be displayed on the headset for the User to interpret and interact with. The User will interact with the Ar headset and the screen. The screen will be used to configure the module and will be used to display redundant information that the AR headset displays while in operation.

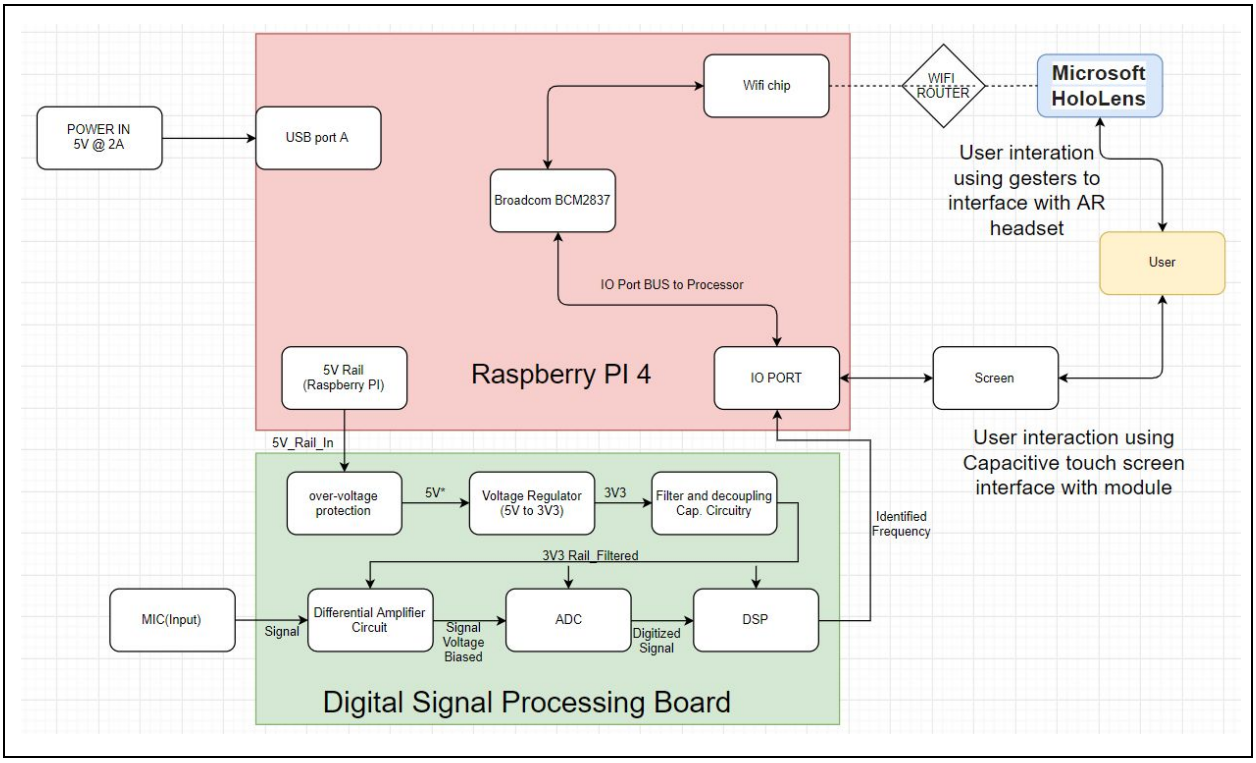

Figure 19: Hardware level 3 Diagram

Figure 19 shows the Level 3 Hardware Diagram. The system will receive power from an outlet and be rectified into 5V @ 2A power using a cellphone power supply. This power will be delivered to the Raspberry Pi 4. The data processing board will be powered via the 5V rail on the raspberry Pi 4. The 5V rail from the Raspberry Pi 4 will go through an over voltage protection

circuit to protect the voltage regulator from being damaged. The voltage regulator will step the 5V voltage down to 3.3V. The 3.3V rail will then be filtered to ensure a clean and stable 3.3V rail. The filtered 3.3V rail will then be used to drive the differential amplifier circuitry, the ADC chip,and the DSP chip. The data processing module will receive an Audio Signal from the keyboard. The differential amplifier circuitry will take in the microphone transduced signal and produce a signal from 0 to 3.3. The ADC will take the biased signal and produce a digitized signal for the DSP chip to run a fast fourier transform on. The determined frequency that was calculated by the DSP will then be sent to the Raspberry Pi to be converted to MIDI. The MIDI signal will be sent over WIFI wireless protocol to the AR headset. The frequencies will then be displayed on the headset for the User to interpret and interact with. The User will interact with the AR headset and the screen. The screen will be used to configure the Visual Music Assistant module to connect to the WIFI router and will also be used to display redundant information that the AR headset displays while in operation.

OpAmp Circuit

Figure 20 shows the operational amplifier circuit simulated to take an audio signal of less than +/- 100mV and produce an output that has a positive value between 3.3 and 0 volts. Ltspice symbol schematic was created with equivalent parts used to created a circuit prototype.

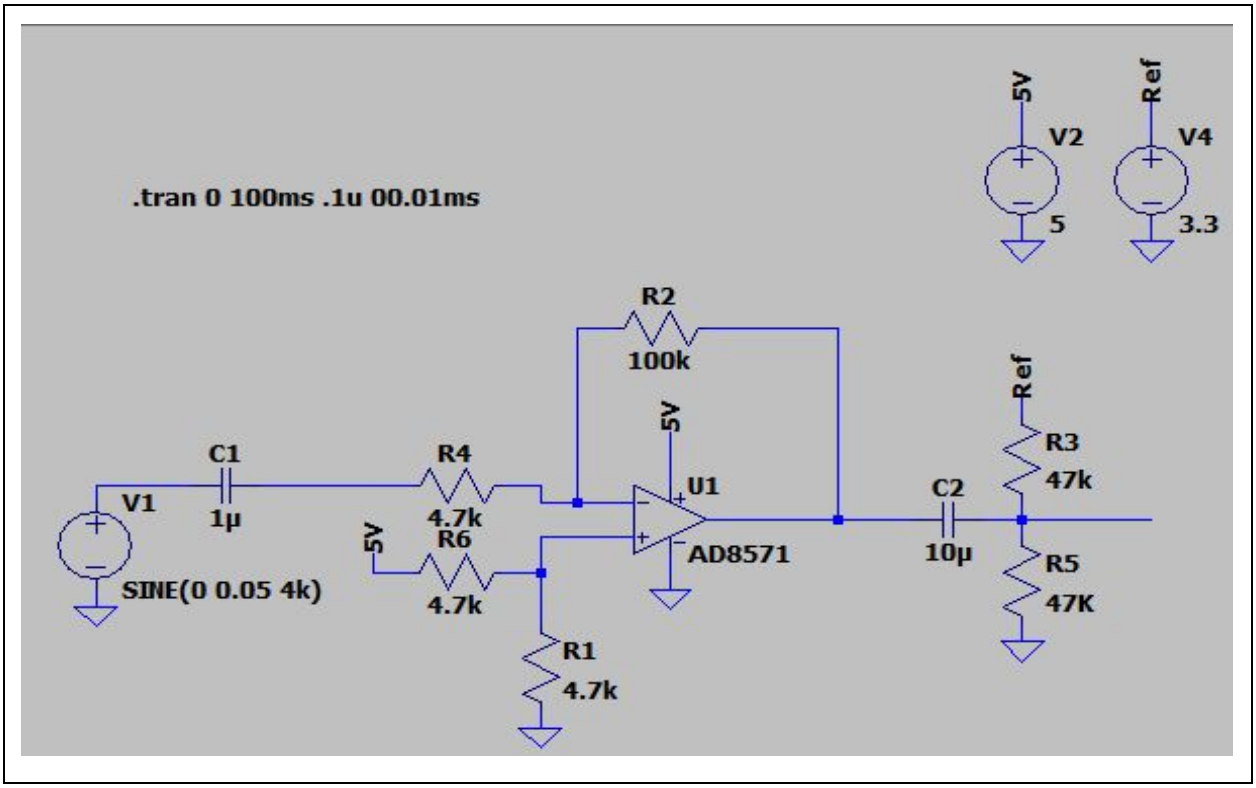

Figure 20: Operational Amplifier Circuit and Waveform

V_1 represents an audio input after a microphone transduces an audio signal into a voltage. The input of V_1 was observed to be less than +/- 50mV in standard operation. The circuit was designed to still work (with no climping) with an input from V_1 of +/- 100mV. C_1 creates a DC buffer for the circuit. C_1 also creates a pass though for the AC component of the V_1 input. The resistance values of R4 and R2 where chosen to create a gain over 20 for the circuit. R6 was chosen to be 4.7KΩ and R1 was chosen to be 4.7KΩ. R6 and R1 were chosen to set a voltage reference point for the input of the Op-Amp. The voltage value chosen for the reference between the positive input terminal node of the Op-Amp was 2.5V. The negative input terminal node of the Op-Amp is also 2.5V. When V_1 excites the circuit with a voltage higher than 0mV then current through R4 is decrease. The decrease in the current through R4 also leads to a decrease in the current through R2. The opposite is true for a voltage lower than 0mV. The circuit is inverting the signal. The Op-Amp chosen to represent the physical Op-Amp was AD8571. AD8571 has zero drift, 2.7/5V operation, and RRIO capability. The actual Op-amp used TLV2252AIP has increased output dynamic range, lower noise voltage, and lower input offset voltage with operation from 2.7/8V and RRIO capability. C2 is a DC buffer and AC pass through used to separate the output signal of the circuit from the AD8571 Op-Amp signal. The AD8571 Op-Amp signal voltage generated is between 1.5V and 3.6V which exceeds the allowable voltage input range 0 to 3V for the dsPIC33FJ256MC710A. The isolated signal generated after passing through C2 is shifted to operate on a DC value of 1.65V. The shift of the output signal is produced using resistors R3 and R5. R3 was chosen to be 47KΩ and R5 was chosen to be 47KΩ.

Figure 21 shows the operational amplifier waveform generated by taking an audio signal of +/- 100mV and produce an output that has a positive output between 3.3 and 0 volts

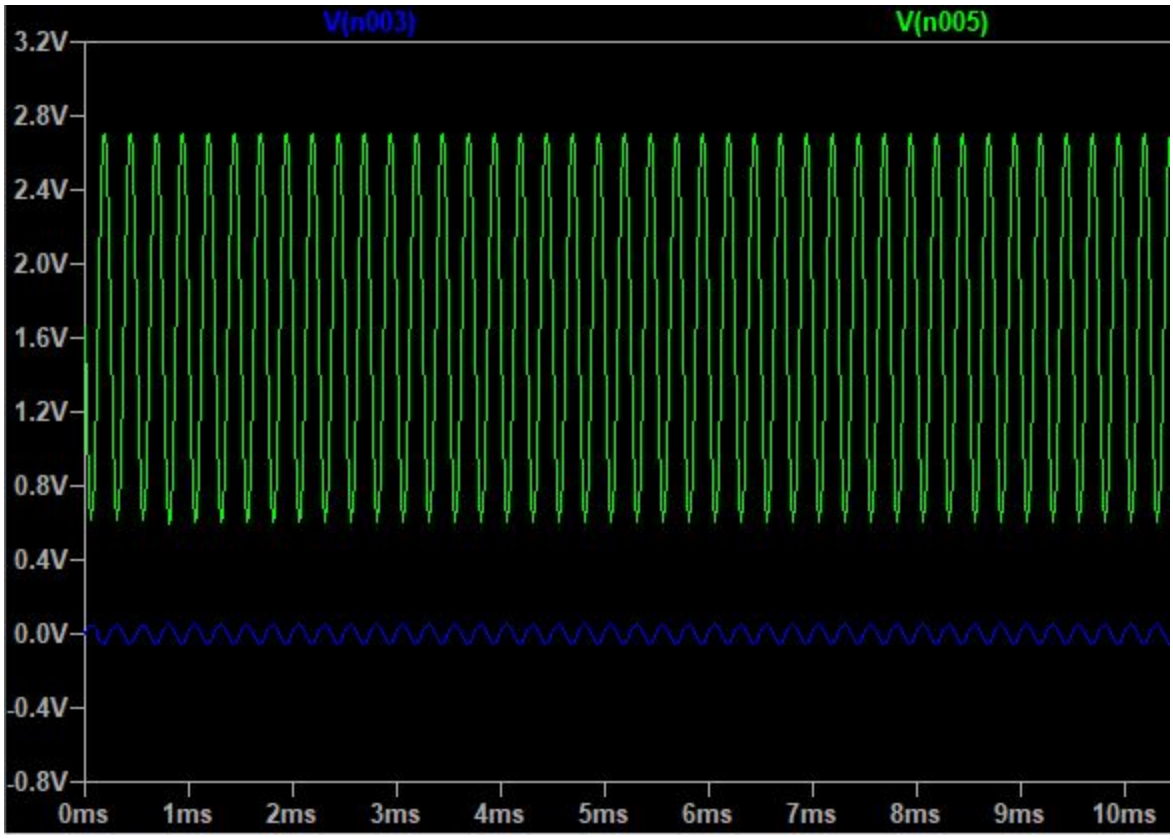

Figure 21: Operational Amplifier Simulated Waveform

The small (blue) signal is the input signal from the microphone V_1. The input signal is generated using a sinusoidal wave at 4KHz between +/- 50mV. The large signal (yellow) output signal is 4KHz ranging from 600mV to 2.7V. The gain of the system as seen from the waveform is 21.

Figure 22 shows the operational amplifier symbol schematic modeled in Kicad to take an audio signal of less than +/- 100mV and produce an output that has a positive value between 3.3 and 0 volts.

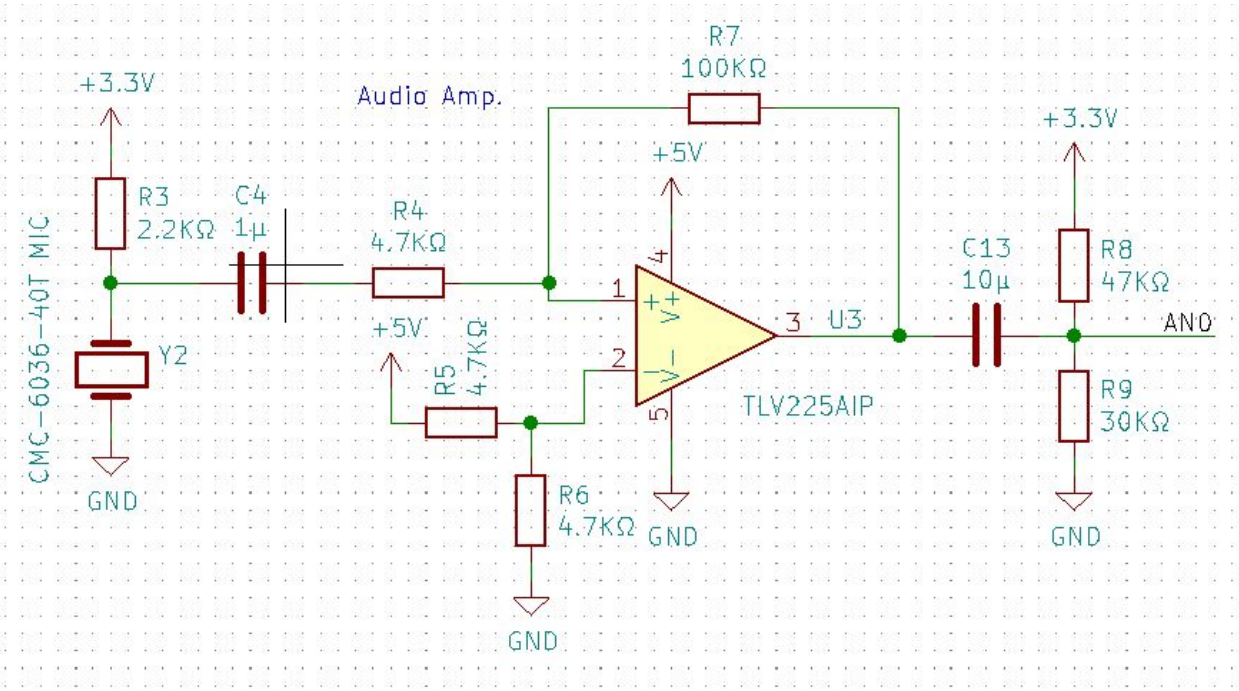

Figure 22: Operational Amplifier Symbol Schematic

The Symbol schematic include the actual parts that will be used to implement Visual music assistant digital signaling board.

Voltage Regulator Circuit

Figure 23 shows the voltage regulator circuit used to filter noise from the Raspberry Pi 5v rail

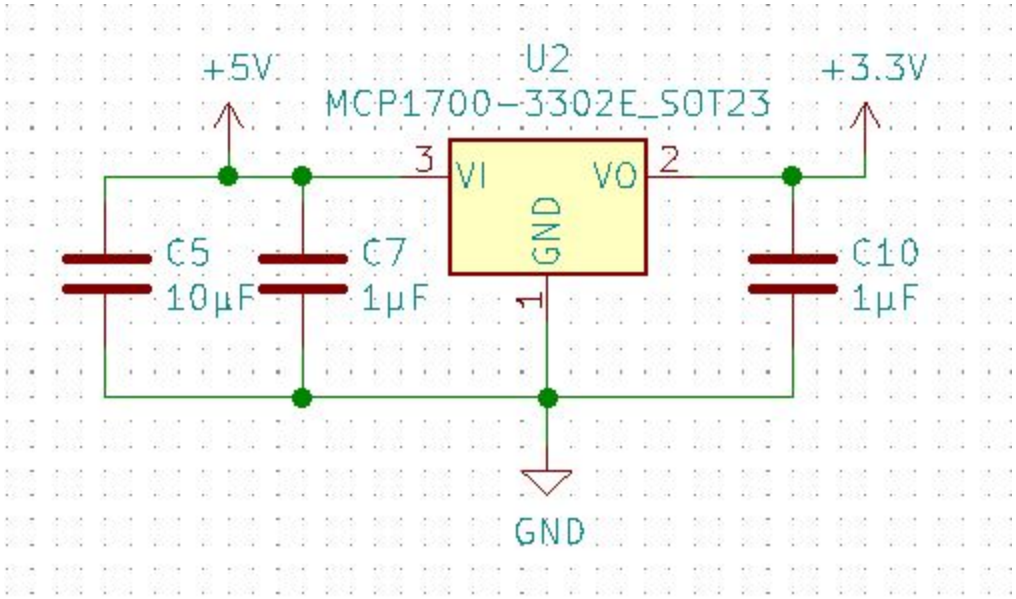

Figure 23: Voltage Regulator Circuit

The MCP1700-3302E is a SOT23 surface mount package linear regulator that takes 5V input and produces 3.3V output. The C7 and C10 are decoupling capacitors used by the linear regulator for filtering. C5 values was chosen to be 10µF to filter noise from the raspberry pi 5V rail.

5.2 Software Design: LF KV DK

Software Level 0 thru N block diagrams w/ Functional Requirements (FR) Tables

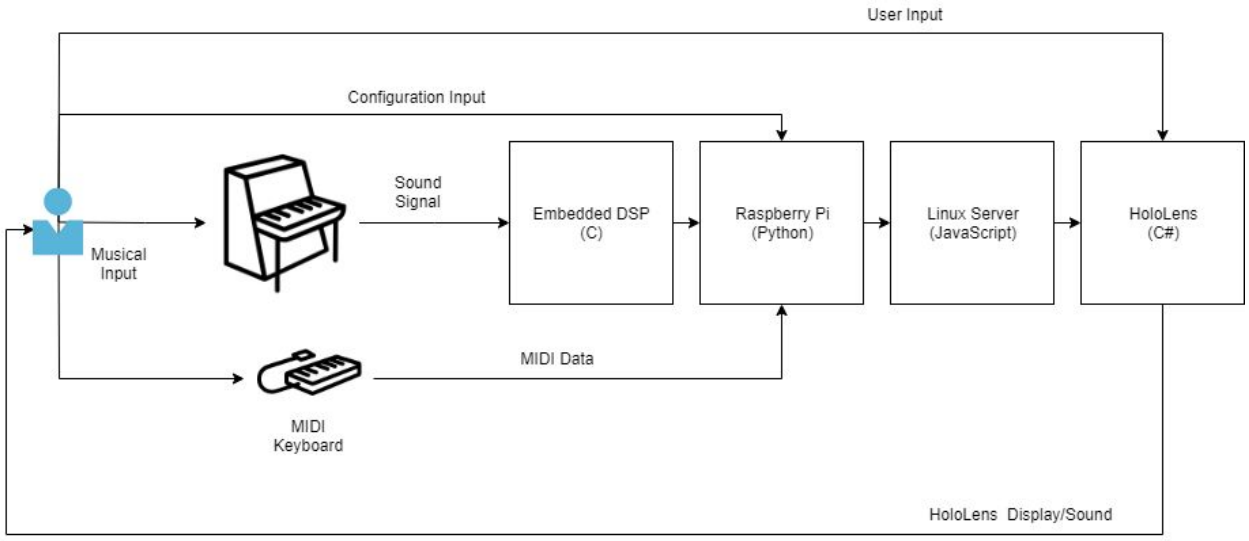

Figure 24: Software Level 0 Diagram

The diagram above shows the top level view of the project from a data transfer perspective. It shows a user (the blue icon) interacting with an analog or digital piano; if the user interacts with an analog piano, the soundwaves of the piano are passed into a microphone connected to the digital signal processor. If the user interacts with a digital piano, the digital music events from the piano are sent directly to a Raspberry Pi for processing. Any analog soundwaves read by the digital signal processor are translated into frequency arrays and passed to the Raspberry Pi. The Raspberry Pi processes the frequency arrays sent from the digital signal processor and calculates which MIDI note the frequency arrays represent. Once the Raspberry Pi has retrieved a MIDI note from a frequency array or directly from a digital piano, it packages the MIDI note as a JavaScript Object Notation (JSON) string and then encodes it as a byte array to

be sent to the server via TCP/UDP. When the server receives the byte array containing the MIDI data, it forwards it to the HoloLens via TCP/UDP. Once the HoloLens has received the MIDI data from the server, it compares the “note played” from the MIDI data to the note of the song currently being played. The HoloLens then calculates a score based on the difference in timing between the note played and the target note, and displays to the user the note played as well as an indicator of the score.

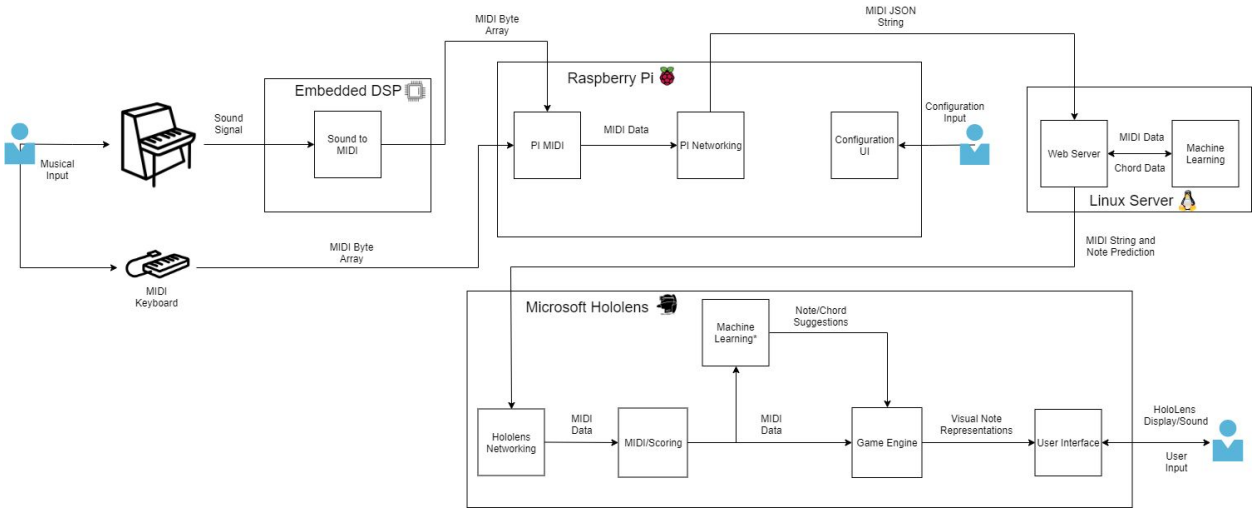

Figure 25: Software Level 1 Diagram

The diagram above(Figure 25) expands on the previous data flow diagram in that it further breaks down each module. The Raspberry Pi now has three modules. The first module, the Pi MIDI module, processes the frequency arrays sent from the digital signal processor and calculates which MIDI note the frequency arrays represent. Once the Pi MIDI module has retrieved a MIDI note from a frequency array or directly from a digital piano, sends the MIDI note to the Pi Networking module. The second module, the Pi Networking module, sets up a

socket connection to the server based on the IP address obtained by the configuration module. The Pi Networking module relays to the server via TCP/UDP any MIDI note passed to it. The module converts MIDI notes to byte arrays containing JSON data before sending them. The third module, the Pi Configuration Module, allows users to interact with a screen connected to the Raspberry Pi. Users will enter their WiFi information using the touch screen, and the information will be used to configure the socket connection between the Raspberry Pi and the server.

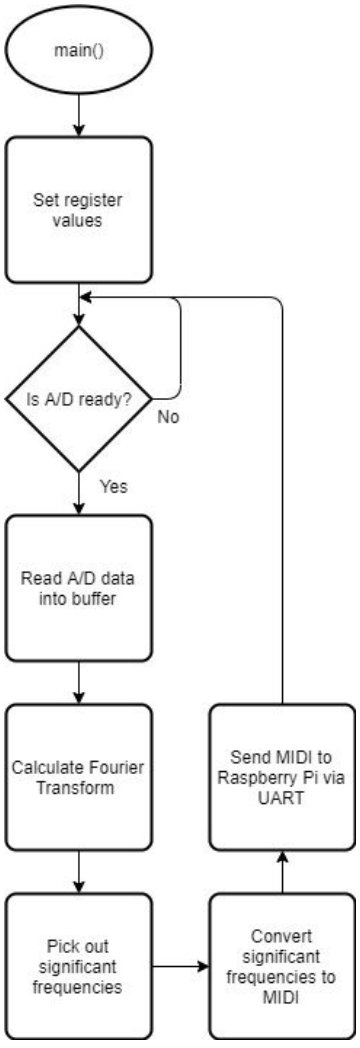

Figure 26: DSP Software Level 2 Diagram

The above figure 26 describes the overall digital signal processing software flow. The DSP will be first configured in a setup block before entering the main while loop. Inside the main while loop, values from the ADC will be read into a buffer. An FFT (Fast Fourier Transform) algorithm will then be operated on the values inside the buffer, filtering out the relevant frequencies. These frequencies will then be converted to their corresponding MIDI notes values and be sent to the Raspberry Pi via the UART communication protocol.

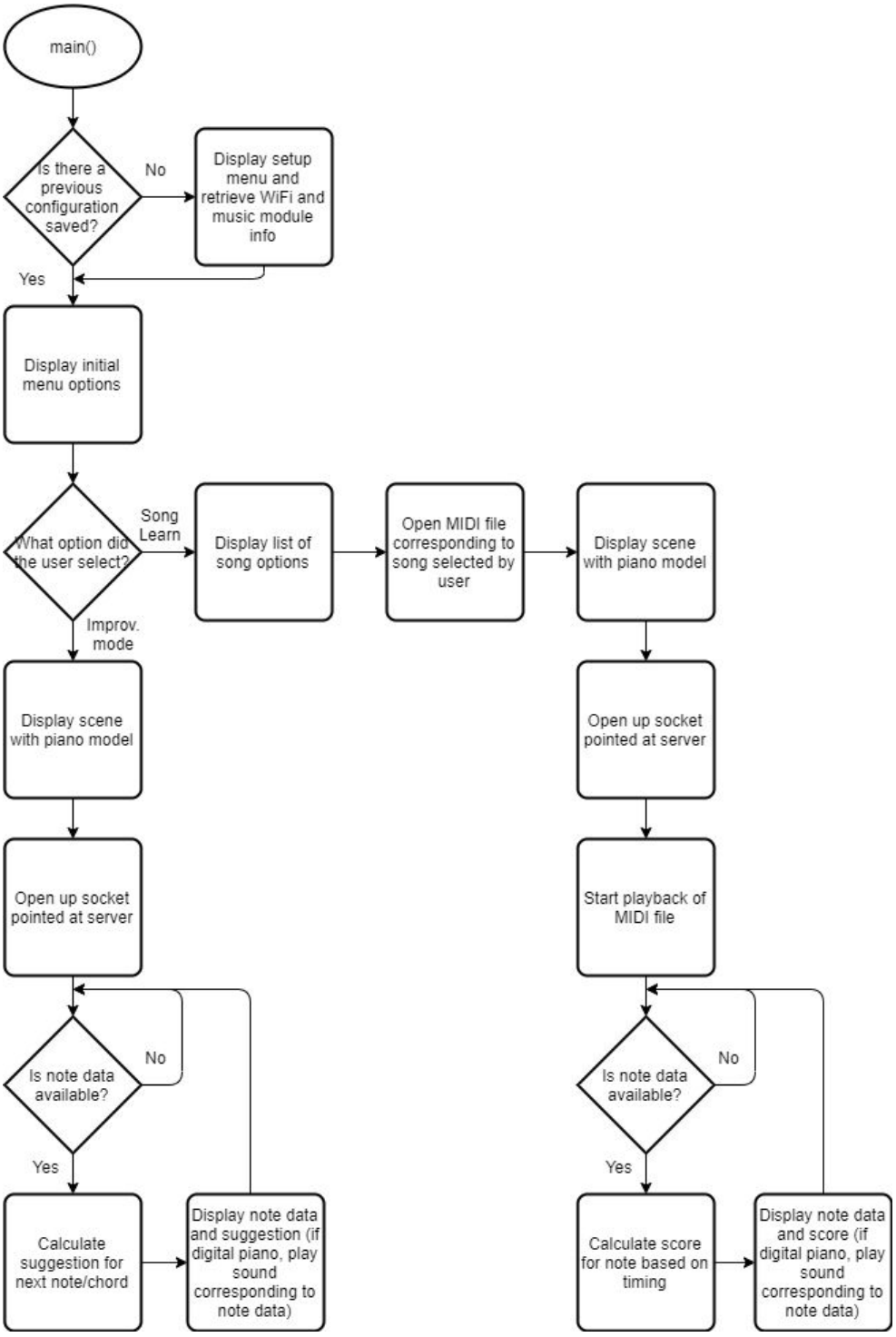

Figure 27: HoloLens Software Level 2 Diagram

Figure 27 describes the control flow of the program that will be running on the HoloLens. It will initially check to see if there is a configuration saved for WiFi credentials and the IP address of the music module. If a configuration is not found, a menu will be displayed that will retrieve the WiFi credentials and IP address of the music module. If a configuration is found, the program will display a menu offering two different options for music learning.

If the user selects the Song Learning menu option, the program will display a list of songs available for playback. Once the user picks a song the program will open a file stream corresponding to the song picked by the user. After the file stream is opened, the program will load in the 3D model that represents the piano and the notes currently being played. Once the model is loaded the program will open a socket connection to the server. After the socket connection is opened the program will listen on the socket for incoming note data. When note data is received, the program will calculate a score for the note based on when it was played vs. when it should have been played. Once the score calculation is complete, the score and the corresponding note will be displayed on the 3D piano model.

If the user selects the Improvisation menu option, the program will load in the 3D model that represents the piano and the notes currently being played. Once the model is loaded the program will open a socket connection to the server. After the socket connection is opened the program will listen on the socket for incoming note data. When note data is received, the program will calculate relevant note/chord suggestions based on music theory. Once the note/chord suggestion is calculated, it will be displayed along with the note played on the 3D piano model.

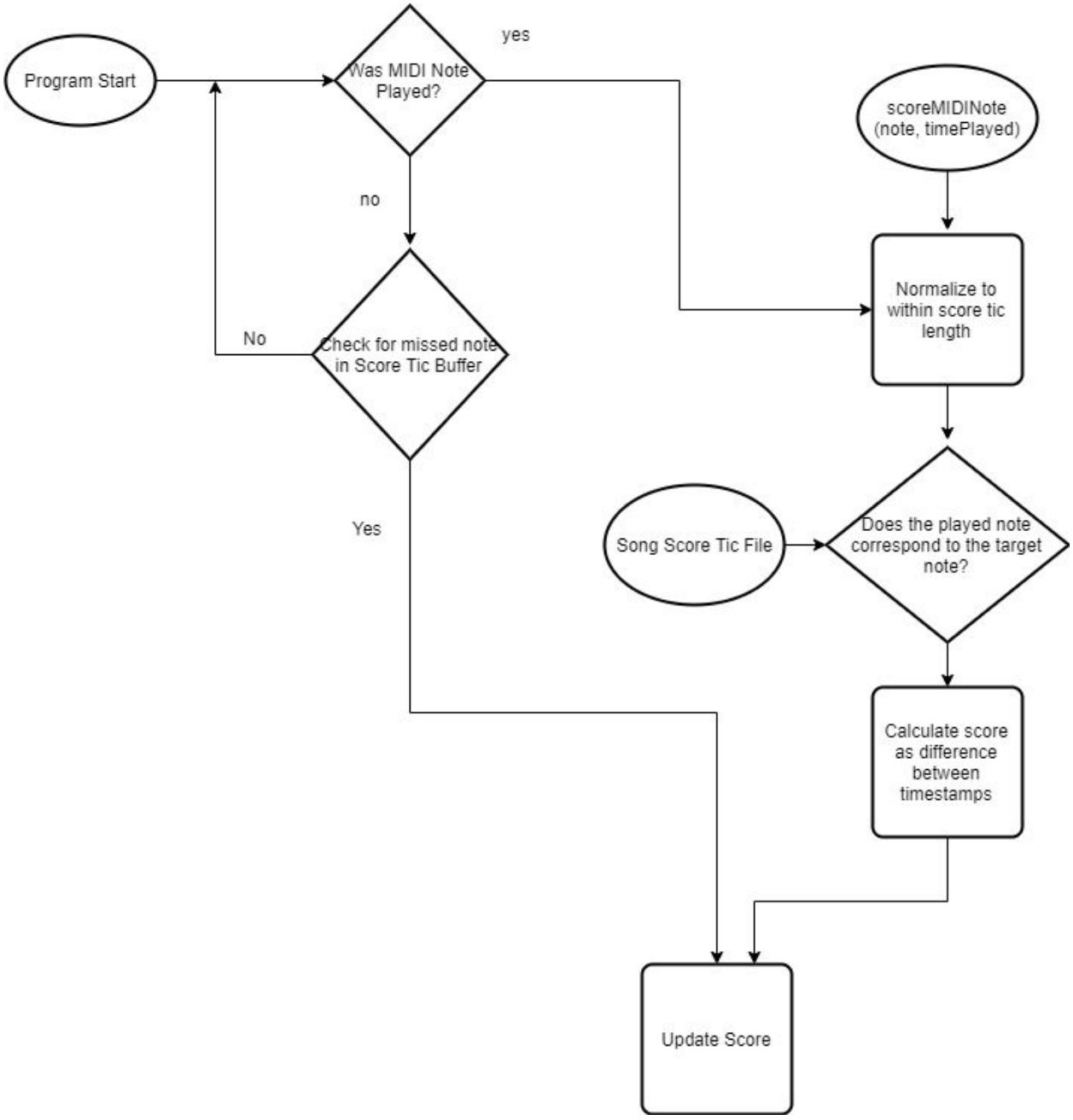

Figure 28: HoloLens Score Loop Software Level 3 Diagram

The diagram above(Figure 28) describes how the program running on the HoloLens determines the score for each note played by the user. It first checks to see if a MIDI note has been played.

If a MIDI note has been played then it is normalized to be compared to a sliding-window file structure containing the notes that should be played and when they should be played. Once the comparison between the note played and the file structure has been made, the score for the note is reflected in the user's total score.

If a MIDI note has not been played, the program checks for any missed notes in the sliding-window file structure. If a missed note is found, the user's score is updated appropriately. If no missed note is found, the program continues to check for played MIDI notes.

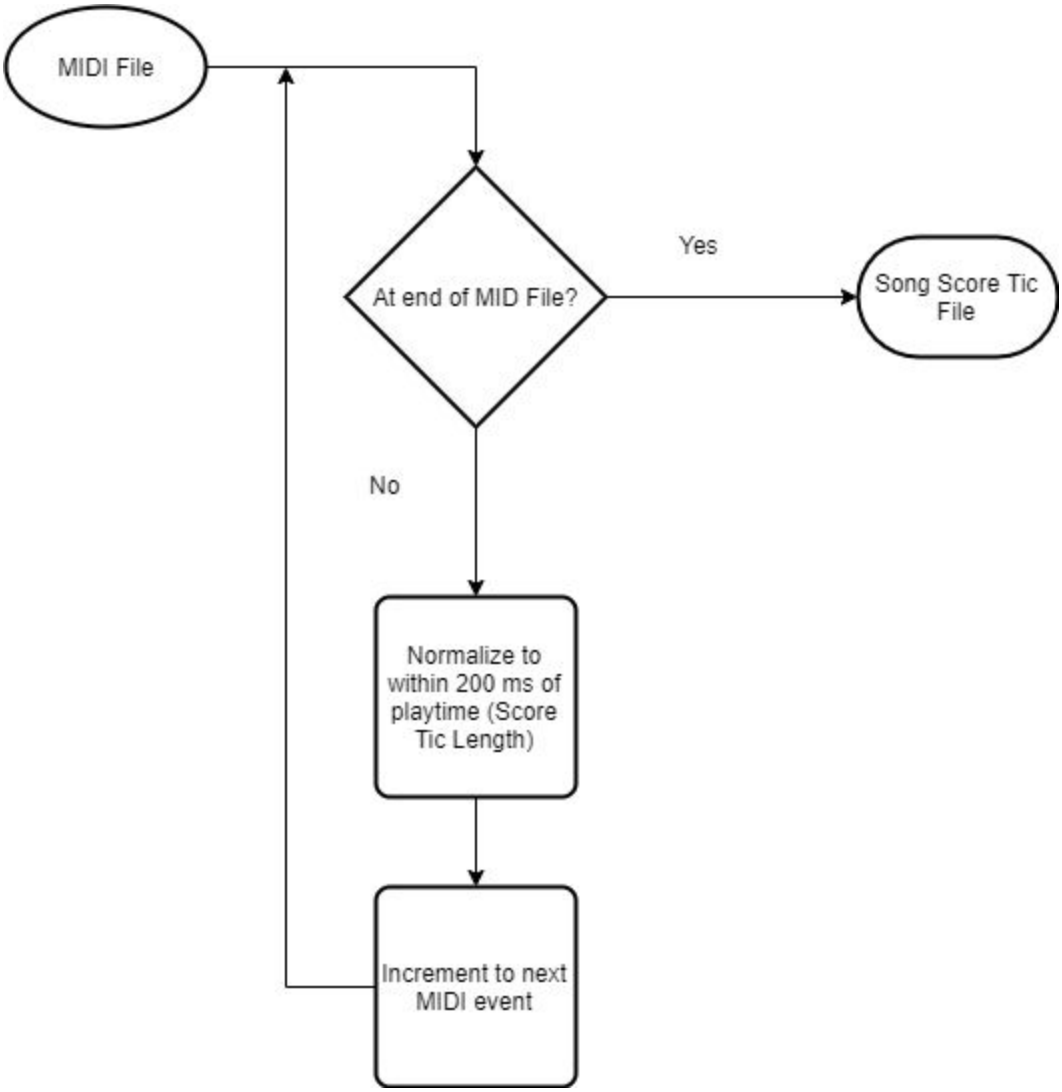

Figure 29: HoloLens MIDI File Level 3 Diagram

Figure 29 describes how the program running on the HoloLens implements the sliding-window structure for parsing MIDI files. It checks to see if the end of the MIDI file has been reached. If the end of the file has been reached, the file is exported. If the end of the MIDI

file has not been reached, the next MIDI note event is read and then normalized to be evenly spaced with previous and subsequent notes.

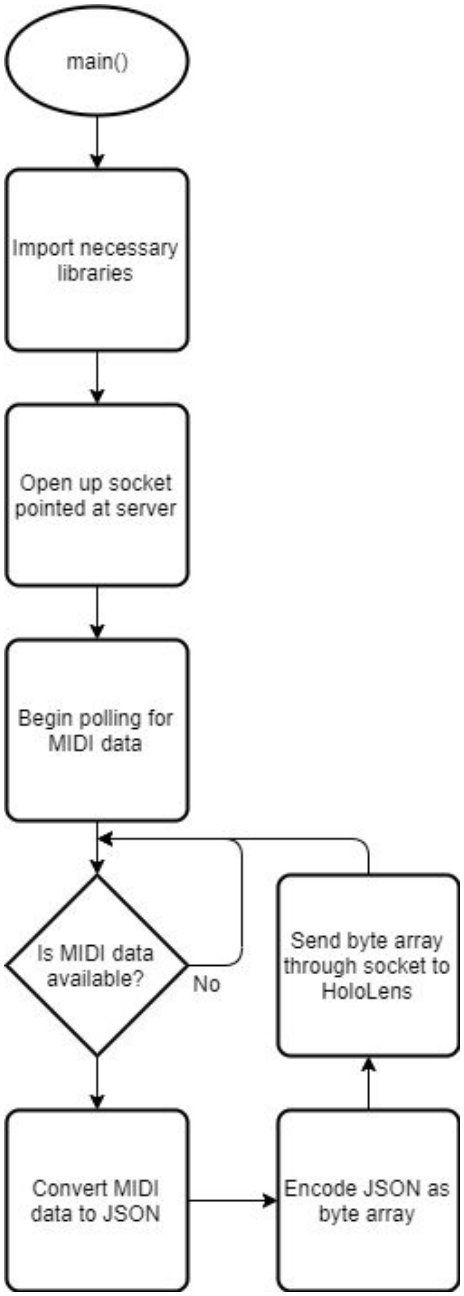

Figure 30: Raspberry Pi Level 2 Diagram

Figure 30 describes the control flow of the program running on the Raspberry Pi. The program first imports all libraries necessary for polling MIDI keyboards and establishing socket connections. Once all libraries are imported the program opens up a TCP/UDP socket connection using the IP address entered by the user into the Pi Configuration module. Once the socket connection is opened the program begins polling for MIDI data. If MIDI data is received, it is converted to JSON and then added to a byte array. The resulting byte array is sent via TCP/UDP to the server (and subsequently the HoloLens)

Pseudocode for DSP frequency to MIDI:

Input: Frequency (obtained from the DSP calculations to the RPi through UART)
Output: MIDI note

Functions within the scope of determining the MIDI note:

- search_avl_tree (Lookup algorithm will be an AVL tree, self-balancing binary search tree data structure); The function will return the MIDI note
- parse_into_midi_data_structure
- get_midi_note

```
get_midi_note( frequency value )
{
        note = search_avl_tree(frequency value);
        midi_data = parse_into_midi_data_structure(note);
        return midi_data;
}
```

Microchip embedded C Demo Code (dsPIC33FJ256GP710A):

Header file config.h:

Figure 31 shows the Header file config.h code.

11/22/2019 config.h

```
1  // ******************************************************************************
2  // ******************************************************************************
3  // Section: Configuration Bits
4  // ******************************************************************************
5  // ******************************************************************************
6  // FBS
7  #pragma config BWRP = WRPROTECT_OFF     // Boot Segment Write Protect (Boot Segment
   may be written)
8  #pragma config BSS = NO_FLASH           // Boot Segment Program Flash Code Protection
   (No Boot program Flash segment)
9  #pragma config RBS = NO_RAM             // Boot Segment RAM Protection (No Boot RAM)
10
11 // FSS
12 #pragma config SWRP = WRPROTECT_OFF     // Secure Segment Program Write Protect
   (Secure Segment may be written)
13 #pragma config SSS = NO_FLASH           // Secure Segment Program Flash Code
   Protection (No Secure Segment)
14 #pragma config RSS = NO_RAM             // Secure Segment Data RAM Protection (No
   Secure RAM)
15
16 // FGS
17 #pragma config GWRP = OFF               // General Code Segment Write Protect (User
   program memory is not write-protected)
18 //#pragma config GSS = OFF                // General Segment Code Protection (User
   program memory is not code-protected)
19 #pragma config GSS = GCP_OFF            // match what my settings work for uart
20
21 // FOSCSEL
22 #pragma config FNOSC = FRC              // Oscillator Mode (Internal Fast RC (FRC))
23 //#pragma config FNOSC = FRCPLL           // match what my settings work for uart
24 #pragma config IESO = OFF               // Two-speed Oscillator Start-Up Enable
   (Start up with user-selected oscillator)
25
26 // FOSC
27 #pragma config POSCMD = XT
28 #pragma config OSCIOFNC = OFF
29 #pragma config FCKSM = CSECMD
30
31 // FWDT
32 #pragma config WDTPOST = PS32768        // Watchdog Timer Postscaler (1:32,768)
33 #pragma config WDTPRE = PR128           // WDT Prescaler (1:128)
34 #pragma config PLLKEN = ON              // PLL Lock Enable bit (Clock switch to PLL
```

Figure 31: config.h

Figure 32 shows a continuation of Header file config.h code.

```
source will wait until the PLL lock signal is valid.)
#pragma config WINDIS = OFF             // Watchdog Timer Window (Watchdog Timer in
Non-Window mode)
#pragma config FWDTEN = OFF             // Watchdog Timer Enable (Watchdog timer
enabled/disabled by user software)

//FPWRT
#pragma config FPWRT = PWR1             // POR timer value disabled (turn off power
up timer)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.
```

Figure 32: Header file config.h continued

Main Program file: main.c

Figure 33 shows Main Program file: main.c

11/22/2019 main.c

```
1  /*
2   * File:   main.c
3   * Author: david
4   *
5   * Created on November 5, 2019, 12:47 PM
6   */
7  #include <xc.h>
8  #include "fft.h"
9  #include <dsp.h>
10 #include <stdio.h>
11 #include <string.h>
12 #include "config.h"
13 #define FP 40000000
14 #define BAUDRATE 9600
15 #define BRGVAL ((FP/BAUDRATE)/16)-1
16 unsigned int i;
17 #define DELAY_105uS asm volatile ("REPEAT, #4201"); Nop();  // 105uS delay
18 //#define SAMPLING_RATE 262
19 #define SAMPLING_RATE 2846
20
21 void ms_delay(int N) {
22     T4CON = 0x8030;
23     TMR4 = 0;
24     while (TMR4 < 156*N) {}
25 }
26
27 uint16_t ADC_Read10bit(int channel)
28 {
29     uint16_t i;
30
31     AD1CHS0 = channel;
32
33     // Get an ADC sample
34     AD1CON1bits.SAMP = 1;           //Start sampling
35     for(i=0;i<1000;i++)
36     {
37         Nop(); //Sample delay, conversion start automatically
38     }
```

Figure 33: main.c

Figure 34 shows a continuation of Main Program file: main.c

```
39
40      AD1CON1bits.SAMP = 0;           //Start sampling
41      for(i=0;i<1000;i++)
42      {
43          Nop(); //Sample delay, conversion start automatically
44      }
45
46      while(!AD1CON1bits.DONE);       //Wait for conversion to complete
47
48      return ADC1BUF0;
49  }
50
51   // ******************************************************************************
52  // ******************************************************************************
53  // Section: File Scope or Global Constants
54  // ******************************************************************************
55  // ******************************************************************************
56
57  fractcomplex sigCmpx[FFT_BLOCK_LENGTH]
58  __attribute__ ((eds, space(ymemory), aligned (FFT_BLOCK_LENGTH * 2 *2)));
59
60   /* Declare Twiddle Factor array in X-space*/
```

Figure 34: Continuation of Main Program file: main.c

Figure 35 shows a continuation of Main Program file: main.c 2

```
61  #ifndef FFTTWIDCOEFFS_IN_PROGMEM
62  fractcomplex twiddleFactors[FFT_BLOCK_LENGTH/2]
63  __attribute__ ((section (".xbss, bss, xmemory"), aligned (FFT_BLOCK_LENGTH*2)));
64  #else
65  extern const fractcomplex twiddleFactors[FFT_BLOCK_LENGTH/2] /* Twiddle Factor
    array in Program memory */
66  __attribute__ ((space(prog), aligned (FFT_BLOCK_LENGTH*2)));
67  #endif
68
69 fractional output[FFT_BLOCK_LENGTH/2];
70 int16_t peakFrequencyBin = 0; // Declare post-FFT variables to compute the
71 uint32_t peakFrequency = 0; // frequency of the largest spectral component
72 volatile fractcomplex adcBuff[FFT_BLOCK_LENGTH];
73 volatile uint16_t bufferIndex = 0;
74 volatile uint16_t bufferFull = 0;
75
76 int main(void) {
77
78
79 // ******************************************************************************
80 // Section: Configure Oscillator to operate the device at 40 Mhz
81 // Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
82 // Fosc= 8M*40/(2*2)=80Mhz for 8M input clock
83 // ******************************************************************************
84     PLLFBD = 38; // M=38
85     CLKDIVbits.PLLPOST = 0; // N1=2
86     CLKDIVbits.PLLPRE = 0; // N2=2
87      // Initiate Clock Switch to Primary Oscillator with PLL (NOSC=0b011)
88     __builtin_write_OSCCONH(0x03);
89     __builtin_write_OSCCONL(OSCCON | 0x01);
90     __builtin_write_OSCCONL(0x01); // Start clock switching
91
92     while (OSCCONbits.COSC!= 0b011);// Wait for Clock switch to occur
93     while (OSCCONbits.LOCK!= 1);// Wait for PLL to lock
```

Figure 35: Continuation of Main Program file: main.c 2

Figure 36 shows a continuation of Main Program file: main.c 3

```
94
95  // ******************************************************************************
96  // Section: ADC initialization
97  // ******************************************************************************
98
99      AD1CON2bits.VCFG = 0x0;//VRefH = AVDD, VREFL = AVSS
100     AD1CON3bits.ADCS = 0xFF;//ADC Conversion Clock Select bits reserved
101
102     AD1CON1bits.SSRC = 0x0;// Clearing sample bit ends sampling and starts
    conversion
103     AD1CON3bits.SAMC = 0b10000;//Auto Sample Time bits
104     AD1CON1bits.FORM = 0b00;//measure as integer
105     AD1CON2bits.SMPI = 0x0;// ADC interrupt is generated at the completion of every
    sample/conversion operation
106     AD1CON1bits.ADON = 1;//ADC module operating
107     AD1PCFGLbits.PCFG1 = 0 ; //set channel
108
109 // ******************************************************************************
110 // Section: UART Initialization
111 // ******************************************************************************
112
113     U2MODEbits.STSEL = 0;   //1-stop bit
114     U2MODEbits.PDSEL = 0;   //No parity, 8-data bits
115     U2MODEbits.ABAUD = 0;   //Auto-Baud disabled
116     U2MODEbits.BRGH = 0;    //Standard-speed mode
117
118     U2BRG = BRGVAL;         // Baud Rate setting for 9600
```

Figure 36: Continuation of Main Program file: main.c 3

Figure 37 shows a continuation of Main Program file: main.c 4

```
119
120     U2STAbits.UTXISEL0 = 0;      //Interrupt after one TX character is transmitted
121     U2STAbits.UTXISEL1 = 0;
122
123     U2MODEbits.UARTEN = 1;       //Enable UART
124     U2STAbits.UTXEN = 1;         //Enable UART TX (needs to be set after UARTEN)
125
126     int i = 0;
127
128     //TRISA = 0x00;
129     while(1)
130     {
131         // Obtain ADC output
132         while(i < 512)
133         {
134             sigCmpx[i].real = ADC_Read10bit(1) >> 1;
135             sigCmpx[i].imag = 0;
136             i++;
137         }
138         i = 0;
139
140         /* Perform FFT operation */
141
142         FFTComplexIP (LOG2_BLOCK_LENGTH, &sigCmpx[0], (fractcomplex *)
    __builtin_psvoffset(&twiddleFactors[0]), (int)
    __builtin_psvpage(&twiddleFactors[0]));
143
144         /* Store output samples in bit-reversed order of their addresses */
145         BitReverseComplex (LOG2_BLOCK_LENGTH, &sigCmpx[0]);
146
```

Figure 37: Continuation of Main Program file: main.c 4

Figure 38 shows a continuation of Main Program file: main.c 5

```
147         /* Compute the square magnitude of the complex FFT output array so we have a
    Real output vector */
148         SquareMagnitudeCplx(FFT_BLOCK_LENGTH/2, &sigCmpx[0], output);
149
150         /* Find the frequency Bin ( = index into the SigCmpx[] array) that has the
    largest energy*/
151         /* i.e., the largest spectral component */
152         VectorMax(FFT_BLOCK_LENGTH/2, output, &peakFrequencyBin);
153
154         /* Compute the frequency (in Hz) of the largest spectral component */
155         peakFrequency = ((uint32_t)peakFrequencyBin * SAMPLING_RATE
    /FFT_BLOCK_LENGTH);
156         U2TXREG = (peakFrequency);
157     }
158 }
```

Figure 38: Continuation of Main Program file: main.c 5

Description of main.c:

This file is the main program of the demo code running on the dsPIC33FJ256GP710A. It includes the following local files: “fft.h,” “config.h,”; and the following libraries: <xc.h>, <dsp.h>, <stdio.h>, and <string.h>. The first lines after the include statements are definitions in order to calculate the proper baudrate for UART communications. After that, lines 21-25 is the function definition for ms_delay, which works to delay the processor for N number of milliseconds passed into the function. The next function is defined from lines 27-49, and is written following Microchip’s sample code for the dsPIC33FJ256GP710A. ADC_Read10bit returns the value of the ADC from the referenced ADC channel (I pass in channel 1, which refers to AN1 or analog pin 24). The next section of code (lines 51-74) follow the Microchip practices for utilizing the DSP library for Fast Fourier Transform calculations. It contains defines for the FFT buffer arrays, with fractcomplex being a struct with two entries (real and imaginary). Other variables are initialized that will later be used for the frequency calculations. Lines 76-158 contains all of the main function. Lines 79-93 are oscillator configurations such that the internal

oscillator runs at 80 Mhz, or 40 MIPS (Million Instructions per Second). Lines 95-107 contain the ADC initialization. Lines 109-124 contain the UART initialization. Lines 126-158 contain the sampling of the ADC and the FFT calculations. This is only a first revision for a proof of concept, and the next steps include modifying the ADC sampling to increase the sampling rate. The following is a snapshot of the variables window during the debugging phase. During debugging, a function generator with a 100 Hz square wave (0-3 V, peak-to-peak) is input into analog pin 24 for the ADC to read. As you can see, the value for peakFrequency is 0x00000063 (HEX), which in decimal is 99. This corresponds closely to the input of 100 Hz. The reason why they are not identical is due to the resolution of the FFT bins.

Figure 39 shows the watch window during program operation as well as the addresses of output variables.

Figure 39: Watch windows during program operation

Header file fft.h:

Figure 40 shows Header file fft.h

11/22/2019 fft.h

```
1  #ifdef __cplusplus  // Provide C++ Compatability
2
3      extern "C" {
4
5  #endif
6
7  // ******************************************************************************
8  // ******************************************************************************
9  // Section: Constants
10 // ******************************************************************************
11 // ******************************************************************************
12 /* Constant Definitions */
13 #define FFT_BLOCK_LENGTH   512 // 512     /* = Number of frequency points in the FFT
   */
14 #define LOG2_BLOCK_LENGTH     9    /* = Number of "Butterfly" Stages in FFT
   processing */
15
16 #define FFTTWIDCOEFFS_IN_PROGMEM     /*<---Comment out this line of the code if
   twiddle factors (coefficients) */
17                                      /*reside in data memory (RAM) as opposed to
   Program Memory */
18                                      /*Then remove the call to "TwidFactorInit()" and
   add the twiddle factor*/
19                                      /*coefficient file into your Project. An example
   file for a 256-pt FFT*/
20                                      /*is provided in this Code example */
21
22 #ifdef __cplusplus  // Provide C++ Compatibility
23
24     }
25
26 #endif
```

Figure 40: Header file fft.h

Description of fft.h:

This file contains constants for the FFT calculations.

C file: twiddle_factors.c

Figure 41 shows C file: twiddle_factors.c 1

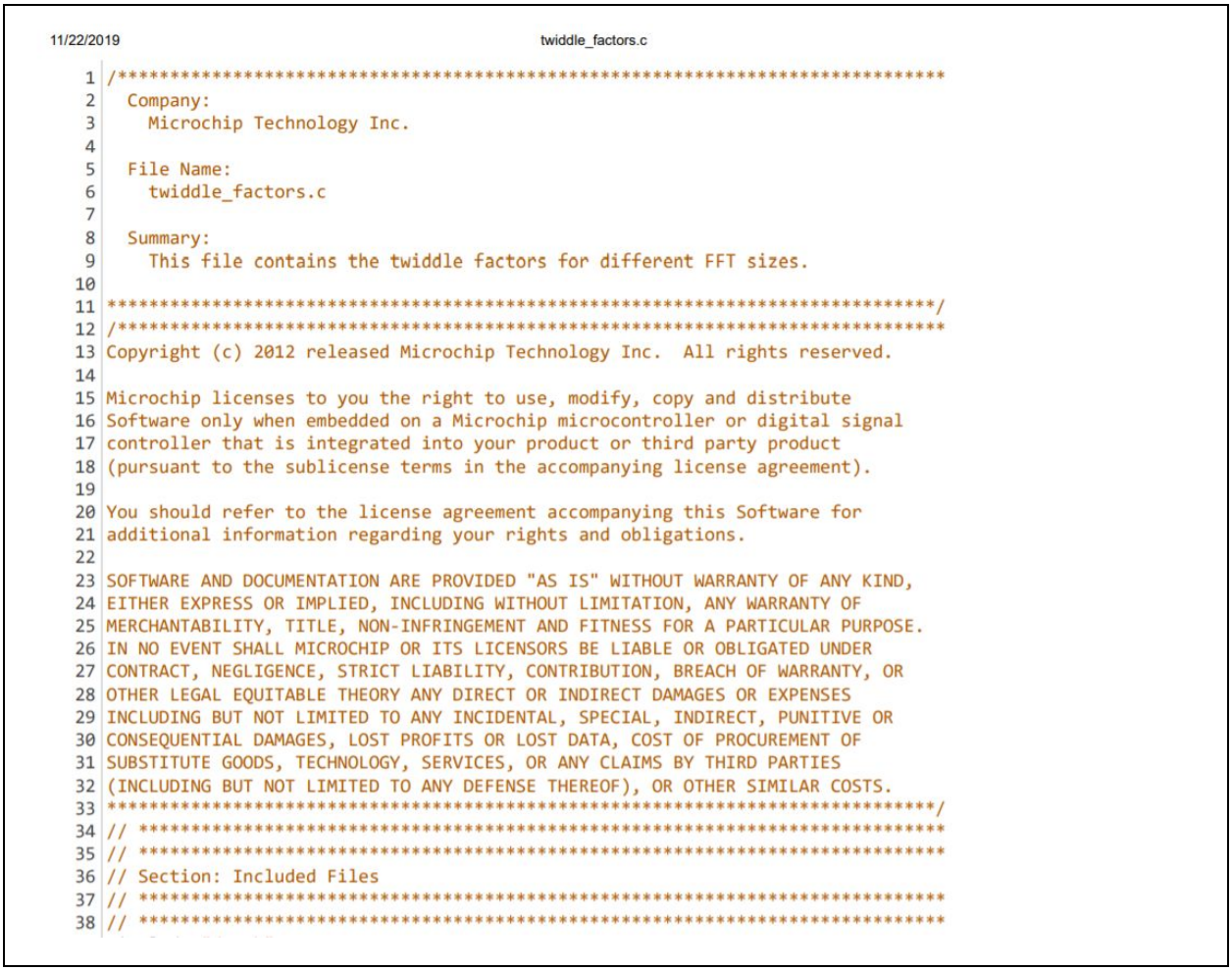

Figure 41: C file: twiddle_factors.c 1

Figure 42 shows a continuation of C file: twiddle_factors.c

```
#include "dsp.h"
#include "fft.h"

#ifdef FFTTWIDCOEFFS_IN_PROGMEM

#if (FFT_BLOCK_LENGTH == 64)
        const fractcomplex twiddleFactors[] __attribute__ ((space(auto_psv), aligned
(FFT_BLOCK_LENGTH*2)))=
        {
        0x7FFF, 0x0000, 0x7F62, 0xF374, 0x7D8A, 0xE707, 0x7A7D, 0xDAD8,
        0x7642, 0xCF04, 0x70E3, 0xC3A9, 0x6A6E, 0xB8E3, 0x62F2, 0xAECC,
        0x5A82, 0xA57E, 0x5134, 0x9D0E, 0x471D, 0x9592, 0x3C57, 0x8F1D,
        0x30FC, 0x89BE, 0x2528, 0x8583, 0x18F9, 0x8276, 0x0C8C, 0x809E,
        0x0000, 0x8000, 0xF374, 0x809E, 0xE707, 0x8276, 0xDAD8, 0x8583,
        0xCF04, 0x89BE, 0xC3A9, 0x8F1D, 0xB8E3, 0x9592, 0xAECC, 0x9D0E,
        0xA57D, 0xA57D, 0x9D0E, 0xAECC, 0x9592, 0xB8E3, 0x8F1D, 0xC3A9,
        0x89BE, 0xCF04, 0x8583, 0xDAD8, 0x8276, 0xE707, 0x809E, 0xF374
        } ;
#endif
#if (FFT_BLOCK_LENGTH == 128)
        const fractcomplex twiddleFactors[] __attribute__ ((space(auto_psv), aligned
(FFT_BLOCK_LENGTH*2)))=
```

Figure 42: Continuation of C file: twiddle_factors.c

Figure 43 shows a continuation of C file: twiddle_factors.c 2

```
59          {
60          0x7FFF, 0x0000, 0x7FD9, 0xF9B8, 0x7F62, 0xF374, 0x7E9D, 0xED38,
61          0x7D8A, 0xE707, 0x7C2A, 0xE0E6, 0x7A7D, 0xDAD8, 0x7885, 0xD4E1,
62          0x7642, 0xCF04, 0x73B6, 0xC946, 0x70E3, 0xC3A9, 0x6DCA, 0xBE32,
63          0x6A6E, 0xB8E3, 0x66D0, 0xB3C0, 0x62F2, 0xAECC, 0x5ED7, 0xAA0A,
64          0x5A82, 0xA57E, 0x55F6, 0xA129, 0x5134, 0x9D0E, 0x4C40, 0x9930,
65          0x471D, 0x9592, 0x41CE, 0x9236, 0x3C57, 0x8F1D, 0x36BA, 0x8C4A,
66          0x30FC, 0x89BE, 0x2B1F, 0x877B, 0x2528, 0x8583, 0x1F1A, 0x83D6,
67          0x18F9, 0x8276, 0x12C8, 0x8163, 0x0C8C, 0x809E, 0x0648, 0x8027,
68          0x0000, 0x8000, 0xF9B8, 0x8027, 0xF374, 0x809E, 0xED38, 0x8163,
69          0xE707, 0x8276, 0xE0E6, 0x83D6, 0xDAD8, 0x8583, 0xD4E1, 0x877C,
70          0xCF04, 0x89BE, 0xC946, 0x8C4A, 0xC3A9, 0x8F1D, 0xBE32, 0x9236,
71          0xB8E3, 0x9592, 0xB3C0, 0x9931, 0xAECC, 0x9D0E, 0xAA0A, 0xA129,
72          0xA57E, 0xA57E, 0xA129, 0xAA0A, 0x9D0E, 0xAECC, 0x9931, 0xB3C0,
73          0x9592, 0xB8E3, 0x9236, 0xBE32, 0x8F1D, 0xC3A9, 0x8C4A, 0xC946,
74          0x89BE, 0xCF04, 0x877C, 0xD4E1, 0x8583, 0xDAD8, 0x83D6, 0xE0E6,
75          0x8276, 0xE707, 0x8163, 0xED38, 0x809E, 0xF374, 0x8027, 0xF9B8
76          } ;
77  #endif
78  #if (FFT_BLOCK_LENGTH == 256)
79          const fractcomplex twiddleFactors[] __attribute__ ((space(auto_psv), aligned
    (FFT_BLOCK_LENGTH*2))) =
80          {
81          0x7FFF, 0x0000, 0x7FF6, 0xFCDC, 0x7FD9, 0xF9B8, 0x7FA7, 0xF695,
82          0x7F62, 0xF374, 0x7F0A, 0xF055, 0x7E9D, 0xED38, 0x7E1E, 0xEA1E,
83          0x7D8A, 0xE707, 0x7CE4, 0xE3F4, 0x7C2A, 0xE0E6, 0x7B5D, 0xDDDC,
84          0x7A7D, 0xDAD8, 0x798A, 0xD7D9, 0x7884, 0xD4E1, 0x776C, 0xD1EF,
85          0x7642, 0xCF04, 0x7505, 0xCC21, 0x73B6, 0xC946, 0x7255, 0xC673,
86          0x70E3, 0xC3A9, 0x6F5F, 0xC0E9, 0x6DCA, 0xBE32, 0x6C24, 0xBB85,
87          0x6A6E, 0xB8E3, 0x68A7, 0xB64C, 0x66CF, 0xB3C0, 0x64E8, 0xB140,
88          0x62F2, 0xAECC, 0x60EC, 0xAC65, 0x5ED7, 0xAA0A, 0x5CB4, 0xA7BD,
89          0x5A82, 0xA57E, 0x5843, 0xA34C, 0x55F6, 0xA129, 0x539B, 0x9F14,
90          0x5134, 0x9D0E, 0x4EC0, 0x9B18, 0x4C40, 0x9931, 0x49B4, 0x9759,
91          0x471D, 0x9592, 0x447B, 0x93DC, 0x41CE, 0x9236, 0x3F17, 0x90A1,
92          0x3C57, 0x8F1D, 0x398D, 0x8DAB, 0x36BA, 0x8C4A, 0x33DF, 0x8AFB,
93          0x30FC, 0x89BE, 0x2E11, 0x8894, 0x2B1F, 0x877C, 0x2827, 0x8676,
94          0x2528, 0x8583, 0x2224, 0x84A3, 0x1F1A, 0x83D6, 0x1C0B, 0x831C,
95          0x18F9, 0x8276, 0x15E2, 0x81E3, 0x12C8, 0x8163, 0x0FAB, 0x80F7,
96          0x0C8C, 0x809E, 0x096B, 0x8059, 0x0648, 0x8028, 0x0324, 0x800A,
97          0x0000, 0x8000, 0xFCDC, 0x800A, 0xF9B8, 0x8028, 0xF695, 0x8059,
98          0xF374, 0x809E, 0xF055, 0x80F7, 0xED38, 0x8163, 0xEA1E, 0x81E3,
99          0xE707, 0x8276, 0xE3F5, 0x831C, 0xE0E6, 0x83D6, 0xDDDC, 0x84A3,
```

Figure 43: Continuation of C file: twiddle_factors.c 2

Figure 44 shows a continuation of C file: twiddle_factors.c 3

```
100         0xDAD8, 0x8583, 0xD7D9, 0x8676, 0xD4E1, 0x877C, 0xD1EF, 0x8894,
101         0xCF04, 0x89BE, 0xCC21, 0x8AFB, 0xC946, 0x8C4A, 0xC673, 0x8DAB,
102         0xC3A9, 0x8F1D, 0xC0E9, 0x90A1, 0xBE32, 0x9236, 0xBB85, 0x93DC,
103         0xB8E3, 0x9593, 0xB64C, 0x975A, 0xB3C0, 0x9931, 0xB140, 0x9B18,
104         0xAECC, 0x9D0E, 0xAC65, 0x9F14, 0xAA0A, 0xA129, 0xA7BD, 0xA34C,
105         0xA57E, 0xA57E, 0xA34C, 0xA7BD, 0xA129, 0xAA0A, 0x9F14, 0xAC65,
106         0x9D0E, 0xAECC, 0x9B18, 0xB140, 0x9931, 0xB3C0, 0x975A, 0xB64C,
107         0x9593, 0xB8E3, 0x93DC, 0xBB85, 0x9236, 0xBE32, 0x90A1, 0xC0E9,
108         0x8F1D, 0xC3A9, 0x8DAB, 0xC673, 0x8C4A, 0xC946, 0x8AFB, 0xCC21,
109         0x89BF, 0xCF04, 0x8894, 0xD1EF, 0x877C, 0xD4E1, 0x8676, 0xD7D9,
110         0x8583, 0xDAD8, 0x84A3, 0xDDDC, 0x83D6, 0xE0E6, 0x831C, 0xE3F5,
111         0x8276, 0xE707, 0x81E3, 0xEA1E, 0x8163, 0xED38, 0x80F7, 0xF055,
112         0x809E, 0xF374, 0x8059, 0xF695, 0x8028, 0xF9B8, 0x800A, 0xFCDC
113         } ;
114 #endif
115 #if (FFT_BLOCK_LENGTH == 512 )
116 const fractcomplex twiddleFactors[]__attribute__((space(auto_psv),aligned(1024)))
117 ={  {0x7FFF, 0x0000}, {0x7FFE, 0xFE6E}, {0x7FF6, 0xFCDC}, {0x7FEA, 0xFB4A},
118     {0x7FD9, 0xF9B8}, {0x7FC2, 0xF827}, {0x7FA7, 0xF695}, {0x7F87, 0xF505},
```

Figure 44: Continuation of C file: twiddle_factors.c 3

Figure 45 shows a continuation of C file: twiddle_factors.c 4

```
119     {0x7F62, 0xF374}, {0x7F38, 0xF1E4}, {0x7F0A, 0xF055},{ 0x7ED6, 0xEEC6},
120     {0x7E9D, 0xED38}, {0x7E60, 0xEBAB}, {0x7E1E, 0xEA1E},{ 0x7DD6, 0xE892},
121     {0x7D8A, 0xE707}, {0x7D3A, 0xE57D}, {0x7CE4, 0xE3F4},{ 0x7C89, 0xE26D},
122     {0x7C2A, 0xE0E6}, {0x7BC6, 0xDF61}, {0x7B5D, 0xDDDC},{ 0x7AEF, 0xDC59},
123     {0x7A7D, 0xDAD8}, {0x7A06, 0xD958},{ 0x798A, 0xD7D9},{ 0x790A, 0xD65C},
124     {0x7885, 0xD4E1}, {0x77FB, 0xD367}, {0x776C, 0xD1EF},{ 0x76D9, 0xD079},
125     {0x7642, 0xCF04}, {0x75A6, 0xCD92}, {0x7505, 0xCC21},{ 0x7460, 0xCAB2},
126     {0x73B6, 0xC946}, {0x7308, 0xC7DB}, {0x7255, 0xC673},{ 0x719E, 0xC50D},
127     {0x70E3, 0xC3A9}, {0x7023, 0xC248}, {0x6F5F, 0xC0E9},{ 0x6E97, 0xBF8C},
128     {0x6DCA, 0xBE32}, {0x6CF9, 0xBCDA}, {0x6C24, 0xBB85},{ 0x6B4B, 0xBA33},
129     {0x6A6E, 0xB8E3}, {0x698C, 0xB796}, {0x68A7, 0xB64C},{ 0x67BD, 0xB505},
130     {0x66D0, 0xB3C0}, {0x65DE, 0xB27F}, {0x64E9, 0xB140},{ 0x63EF, 0xB005},
131     {0x62F2, 0xAECC}, {0x61F1, 0xAD97}, {0x60EC, 0xAC65},{ 0x5FE4, 0xAB36},
132     {0x5ED8, 0xAA0A}, {0x5DC8, 0xA8E2},{ 0x5CB4, 0xA7BD},{ 0x5B9D, 0xA69C},
133     {0x5A83, 0xA57E}, {0x5964, 0xA463}, {0x5843, 0xA34C},{ 0x571E, 0xA238},
134     {0x55F6, 0xA128}, {0x54CA, 0xA01C}, {0x539B, 0x9F14},{ 0x5269, 0x9E0F},
135     {0x5134, 0x9D0E}, {0x4FFB, 0x9C11}, {0x4EC0, 0x9B17},{ 0x4D81, 0x9A22},
136     {0x4C40, 0x9930}, {0x4AFB, 0x9843}, {0x49B4, 0x9759},{ 0x486A, 0x9674},
137     {0x471D, 0x9592}, {0x45CD, 0x94B5}, {0x447B, 0x93DC},{ 0x4326, 0x9307},
138     {0x41CE, 0x9236}, {0x4074, 0x9169}, {0x3F17, 0x90A1},{ 0x3DB8, 0x8FDD},
139     {0x3C57, 0x8F1D}, {0x3AF3, 0x8E62}, {0x398D, 0x8DAB},{ 0x3825, 0x8CF8},
140     {0x36BA, 0x8C4A}, {0x354E, 0x8BA0}, {0x33DF, 0x8AFB},{ 0x326E, 0x8A5A},
141     {0x30FC, 0x89BE}, {0x2F87, 0x8927}, {0x2E11, 0x8894},{ 0x2C99, 0x8805},
142     {0x2B1F, 0x877B}, {0x29A4, 0x86F6}, {0x2827, 0x8676},{ 0x26A8, 0x85FA},
143     {0x2528, 0x8583}, {0x23A7, 0x8511}, {0x2224, 0x84A3},{ 0x209F, 0x843A},
144     {0x1F1A, 0x83D6}, {0x1D93, 0x8377}, {0x1C0C, 0x831C},{ 0x1A83, 0x82C6},
145     {0x18F9, 0x8276}, {0x176E, 0x822A}, {0x15E2, 0x81E2},{ 0x1455, 0x81A0},
146     {0x12C8, 0x8163}, {0x113A, 0x812A}, {0x0FAB, 0x80F6},{ 0x0E1C, 0x80C8},
147     {0x0C8C, 0x809E}, {0x0AFB, 0x8079}, {0x096B, 0x8059},{ 0x07D9, 0x803E},
148     {0x0648, 0x8027}, {0x04B6, 0x8016}, {0x0324, 0x800A},{ 0x0192, 0x8002},
149     {0x0000, 0x8000}, {0xFE6E, 0x8002}, {0xFCDC, 0x800A},{ 0xFB4A, 0x8016},
150     {0xF9B8, 0x8027}, {0xF827, 0x803E}, {0xF695, 0x8059},{ 0xF505, 0x8079},
151     {0xF374, 0x809E}, {0xF1E4, 0x80C8}, {0xF055, 0x80F6},{ 0xEEC6, 0x812A},
152     {0xED38, 0x8163}, {0xEBAB, 0x81A0}, {0xEA1E, 0x81E2},{ 0xE892, 0x822A},
153     {0xE707, 0x8276}, {0xE57D, 0x82C6}, {0xE3F4, 0x831C},{ 0xE26D, 0x8377},
154     {0xE0E6, 0x83D6}, {0xDF61, 0x843A}, {0xDDDC, 0x84A3},{ 0xDC59, 0x8511},
155     {0xDAD8, 0x8583}, {0xD958, 0x85FA}, {0xD7D9, 0x8676},{ 0xD65C, 0x86F6},
156     {0xD4E1, 0x877B}, {0xD367, 0x8805}, {0xD1EF, 0x8894},{ 0xD079, 0x8927},
157     {0xCF04, 0x89BE}, {0xCD92, 0x8A5A}, {0xCC21, 0x8AFB},{ 0xCAB2, 0x8BA0},
158     {0xC946, 0x8C4A}, {0xC7DB, 0x8CF8},{0xC673, 0x8DAB},{ 0xC50D, 0x8E62},
```

Figure 45: Continuation of C file: twiddle_factors.c 4

Figure 46 shows a continuation of C file: twiddle_factors.c 5

```
159     {0xC3A9, 0x8F1D}, {0xC248, 0x8FDD},{ 0xC0E9, 0x90A1},{ 0xBF8C, 0x9169},
160     {0xBE32, 0x9236}, {0xBCDA, 0x9307},{ 0xBB85, 0x93DC},{ 0xBA33, 0x94B5},
161     {0xB8E3, 0x9592}, {0xB796, 0x9674},{ 0xB64C, 0x9759},{ 0xB505, 0x9843},
162     {0xB3C0, 0x9930}, {0xB27F, 0x9A22},{ 0xB140, 0x9B17},{ 0xB005, 0x9C11},
163     {0xAECC, 0x9D0E}, {0xAD97, 0x9E0F},{ 0xAC65, 0x9F14},{ 0xAB36, 0xA01C},
164     {0xAA0A, 0xA128}, {0xA8E2, 0xA238},{ 0xA7BD, 0xA34C},{ 0xA69C, 0xA463},
165     {0xA57D, 0xA57D}, {0xA463, 0xA69C},{ 0xA34C, 0xA7BD},{ 0xA238, 0xA8E2},
166     {0xA128, 0xAA0A}, {0xA01C, 0xAB36},{ 0x9F14, 0xAC65},{ 0x9E0F, 0xAD97},
167     {0x9D0E, 0xAECC}, {0x9C11, 0xB005},{ 0x9B17, 0xB140},{ 0x9A22, 0xB27F},
168     {0x9930, 0xB3C0}, {0x9843, 0xB504},{ 0x9759, 0xB64C},{ 0x9674, 0xB796},
169     {0x9592, 0xB8E3}, {0x94B5, 0xBA33},{ 0x93DC, 0xBB85},{ 0x9307, 0xBCDA},
170     {0x9236, 0xBE32}, {0x9169, 0xBF8C},{ 0x90A1, 0xC0E9},{ 0x8FDD, 0xC248},
171     {0x8F1D, 0xC3A9}, {0x8E62, 0xC50D},{ 0x8DAB, 0xC673},{ 0x8CF8, 0xC7DB},
172     {0x8C4A, 0xC946}, {0x8BA0, 0xCAB2},{ 0x8AFB, 0xCC21},{ 0x8A5A, 0xCD92},
173     {0x89BE, 0xCF04}, {0x8927, 0xD079},{ 0x8894, 0xD1EF},{ 0x8805, 0xD367},
174     {0x877B, 0xD4E1}, {0x86F6, 0xD65C},{ 0x8676, 0xD7D9},{ 0x85FA, 0xD958},
175     {0x8583, 0xDAD8}, {0x8510, 0xDC59},{ 0x84A3, 0xDDDC},{ 0x843A, 0xDF61},
176     {0x83D6, 0xE0E6}, {0x8377, 0xE26D},{ 0x831C, 0xE3F4},{ 0x82C6, 0xE57D},
177     {0x8275, 0xE707}, {0x8229, 0xE892},{ 0x81E2, 0xEA1E},{ 0x81A0, 0xEBAB},
178     {0x8163, 0xED38}, {0x812A, 0xEEC6},{ 0x80F6, 0xF055},{ 0x80C8, 0xF1E4},
179     {0x809E, 0xF374}, {0x8079, 0xF505},{ 0x8059, 0xF695},{ 0x803E, 0xF827},
```

Figure 46: Continuation of C file: twiddle_factors.c 5

Figure 47 shows a continuation of C file: twiddle_factors.c 6

```
180     {0x8027, 0xF9B8}, {0x8016, 0xFB4A},{ 0x800A, 0xFCDC},{ 0x8002, 0xFE6E}
181 };
182 #endif
183
184 #endif
185
```

Figure 47: Continuation of C file: twiddle_factors.c 6

Description of twiddle_factors.c:

This file contains all the hardcoded twiddle facts needed for the FFT calculations. This is stored in memory and included in the main file.

TCP index.js

Figure 48 shows TCP index.js

```
'use strict'

const Buffer = require('buffer').Buffer
const sizeof = require('object-sizeof')
const express = require('express')
const bodyParser = require('body-parser')
const app = express()
const port = 3000
app.use(bodyParser.json())
const net = require('net')
var clients = []
var server = net.createServer(function (socket){
    socket.name = socket.remoteAddress + ":" + socket.remotePort
    clients.push(socket);

    //socket.write("Welcome " + socket.name + "\n");
    //broadcast(socket.name + " joined the server\n", socket)
    console.log(socket.name + " joined the server\n")

    socket.on('data', function(data){
        var temparr = [intToHex(sizeof(data)), data]
        broadcast(Buffer.concat(temparr), socket);
    });

    socket.on('end', function () {
        console.log("client disconnected: " + socket.name)
        clients.splice(clients.indexOf(socket), 1)
       // broadcast(socket.name + " left the server. \n")
    });

    socket.on('error', function (error) {
        console.log("ignoring exception: " + error);
    });

```

Figure 48: TCP index.js

Figure 49 shows a continuation of TCP index.js

```
35      socket.on('close', function(error){
36          var bread = socket.bytesRead;
37          var bwrite = socket.bytesWritten;
38          console.log('Bytes read : ' + bread);
39          console.log('Bytes written : ' + bwrite);
40          console.log('Socket closed!');
41          if(error){
42              console.log('Socket was closed coz of transmission error');
43          }
44      });
45
46      function broadcast(message, sender){
47          clients.forEach(function (client){
48              if (client === sender) return;
49              client.write(message)
50          });
51          process.stdout.write(message + '\n')
52          console.log(Buffer.from(message, 'utf-8'))
53      }
54  }).listen(3001, ()=> console.log("tcp listening on port 3001"));
55  //https://stackoverflow.com/q/40740971/11389709
56
57  server.on('error',function(error){
58      console.log('ignoring Error: ' + error);
59  });
60
61
62  function numToString(num){
63      var data = "";
64      data = Math.floor(num /1000).toString()
65      data += Math.floor((num % 10) / 100).toString()
66      data += Math.floor((num % 100) / 10).toString()
67      data += Math.floor(num % 10).toString()
68      return data
69  };
70
71  function intToHex(num){
72      var buff = Buffer.alloc(4)
73      buff.writeUInt32LE(num,0)
74      console.log(buff)
75      console.log(buff.readUInt32LE(0))
76      return buff
77  };
```

Figure 49: Continuation of TCP index.js

Figure 50 shows a continuation of TCP index.js 2

```
78
79  app.get('/', (req, res) => {
80      var json = {
81          message: "Hello WOrld!"
82      }
83      res.send(json)
84  })
85  app.post('/', (req, res) =>{
86      console.log("body of request: " + JSON.stringify(req.body))
87
88      //console.log("params of request: " + JSON.stringify(req.params))
89      //console.log("params of request: " + req.params.note)
90      console.log(req.body.note[0])
91
92      //send response
93      var json = {
94          message: "Hello WOrld post request!"
95      }
96      res.send(json)
97  })
98
99  app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

Figure 50: Continuation of TCP index.js 2

Description of TCP index.js:

The code above acts as 2 nodeJS servers using the express and net libraries. Clients that connect to the server have the ability to send data, all data sent should be a midi object in json format followed by a newline character. Once data is sent to the server, it will calculate how large the data is and prepend the integer to the message and then sends the data to all other connected clients. If an error occurs from a client disconnecting at an unexpended time, the server will ignore the exception and simply stop sending the disconnected client packets.

Figure 51 shows Example Program output while server is running

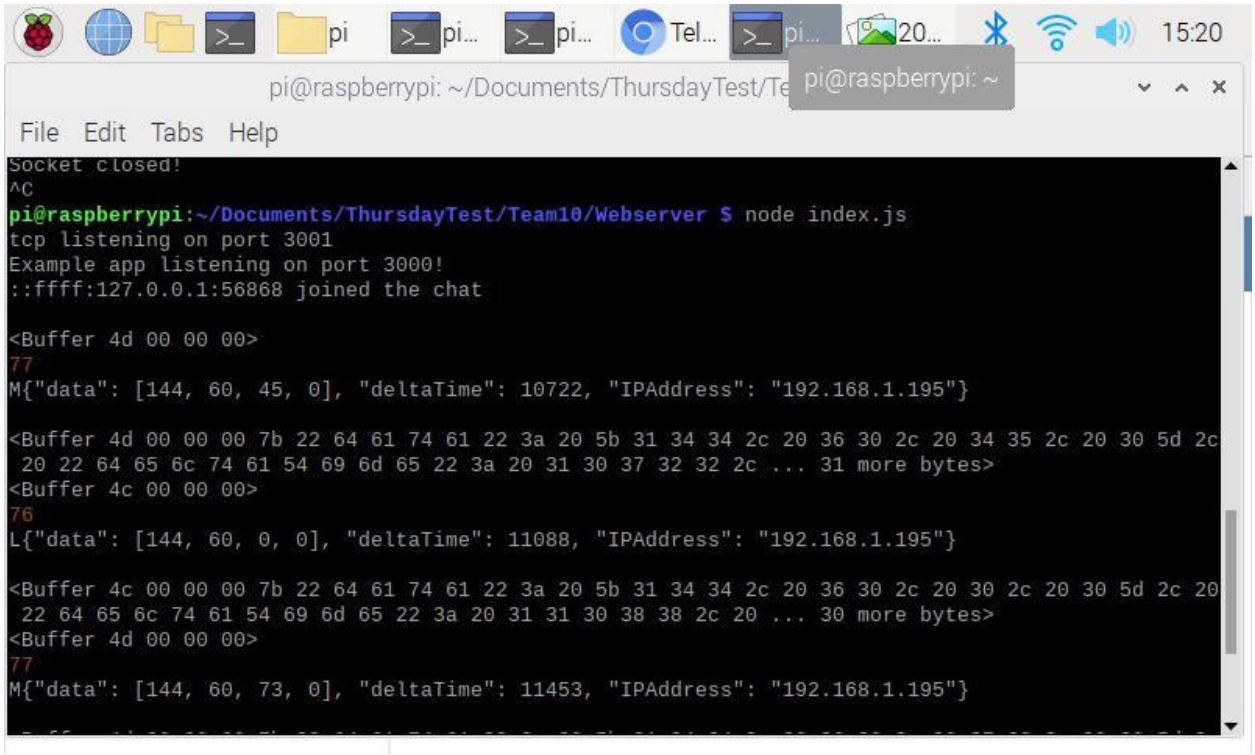

Figure 51 : Example Program output

Server printing the incoming byte arrays and decoding messages is seen in above in figure 48

VMA.py

Figure 52 shows VMA.py

```
import pygame;
import socket;
from pygame.midi import *;
import json;
import requests;
import sys

from tkinter import *
from tkinter.ttk import *

def tcp_format(data):
    json_string = json.dumps(data);
    b = bytearray();
    encoded_string = json_string.encode();
    b.extend(encoded_string);
    b.append(0x0a);
    return b;

def set_device_number():
    #initialization
    count = pygame.midi.get_count()
    devInfo = []


    for x in range(0, count):#fill array
        if pygame.midi.get_device_info(x)[2] == 1:
            devInfo.append([pygame.midi.get_device_info(x)[1],x])


    print(devInfo)

    window = Tk()
    window.title("Visual Music Assistant")
    window.geometry('800x480')

    combo = Combobox(window,values=devInfo,width=50)
    combo.current(0)
    combo.grid(column=1,row=1)

```

Figure 52: VMA.py

Figure 53 shows a continuation of VMA.py

```
    lbl = Label(window, text="Find MIDI Devices")
    lbl.grid(column=0,row=0)
    lbl2 = Label(window, text="Select MIDI Device:")
    lbl2.grid(column=0,row=1)

    def clicked():
        global DeviceNumber
        print("Device number: " , int(combo.get()[-1:]))
        DeviceNumber = int(combo.get()[-1:])
        window.destroy()#close gui

    btn = Button(window, text="Connect to Device", command=clicked)
    btn.grid(column=0,row=3)

    window.mainloop()

def get_ip(): # stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(('10.255.255.255', 1))
        IP = s.getsockname()[0]
    except:
        IP = '127.0.0.1'
        print('get_ip hit an exception')
    finally:
        s.close()
    return IP

#start main

try:
    pygame.midi.init();
    DeviceNumber = pygame.midi.get_default_input_id()
    set_device_number();#Set the device number
    print("The Device number is :" , DeviceNumber)
    input = pygame.midi.Input(DeviceNumber);
except MidiException:
    print("No midi device detected!")
    sys.exit()
```

Figure 53: Continuation of VMA.py

Figure 54 shows a continuation of VMA.py

```
TCP_IP = '127.0.0.1';
TCP_PORT = 3001;
BUFFER_SIZE = 1024;
MESSAGE="The MIDI BOX HAS CONNECTED!";

#with is unneccesary just set s to the socket.socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    #s.setsockopt(socket.SOL_SOCKET,socket.SO_SNDBUF,BUFFER_SIZE)
    my_ip_address = get_ip() #  Device IP Address
    s.connect((TCP_IP, TCP_PORT));
    #s.send(MESSAGE.encode());
    #data =  s.recv(BUFFER_SIZE);


    #print("received data: ", data);

    while 1:
        if input.poll():
            list = input.read(1);
            if (list[0][0][0] == 144) or (list[0][0][0] == 128):
                MIDIObject = {
                    "data": list[0][0],
                    "deltaTime": list[0][1],
                    "IPAddress": my_ip_address
                    }
                #print(json.dumps(list));
                print(json.dumps(MIDIObject));
                tcp_package = tcp_format(MIDIObject)
                print(sys.getsizeof(tcp_package))
                print('\n')
                s.sendall(tcp_package);

            #testdata = {'note': list}
            #res = requests.post(url=URL, json=testdata);
            #print(res.status_code);
```

Figure 54 : Continuation of VMA.py

Description of VMA.py:

This program launches a gui that will ask the user to select a midi device. After a midi device is selected, the gui will close and connect to the nodejs server. It then will listen to the selected midi device and send all midi packets to the server through TCP.

Figure 55 shows Example program output 2

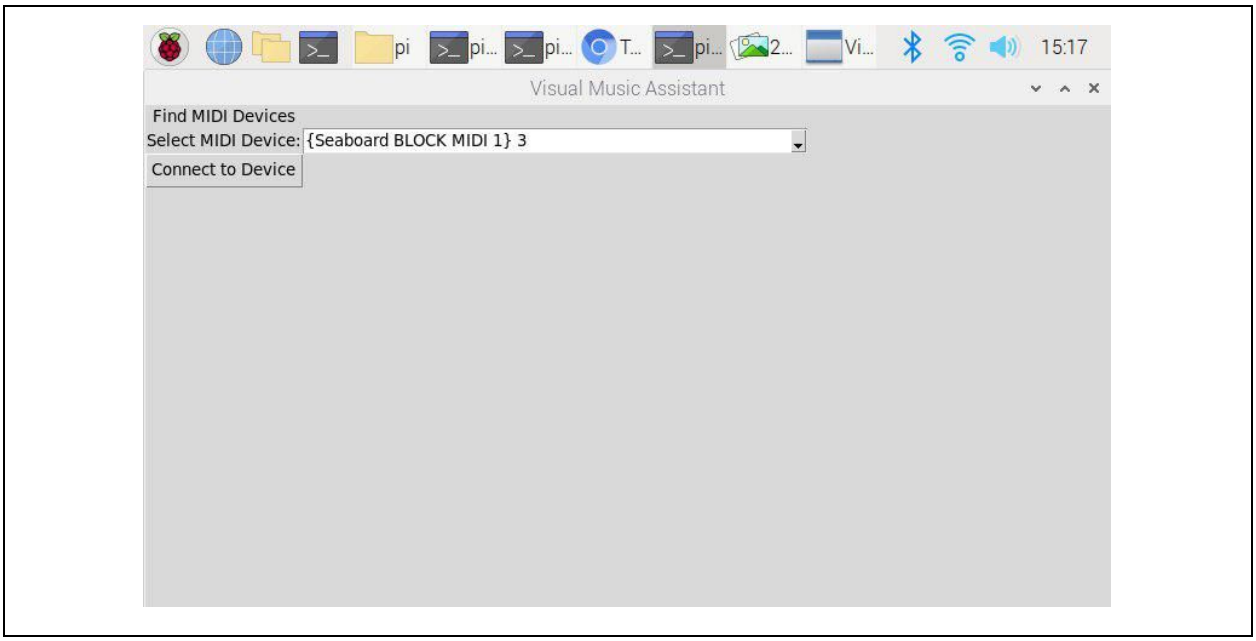

Figure 55: Example program output 2

The user is able to select a desired plugged in midi device.

Figure 56 shows Midi packets formulation

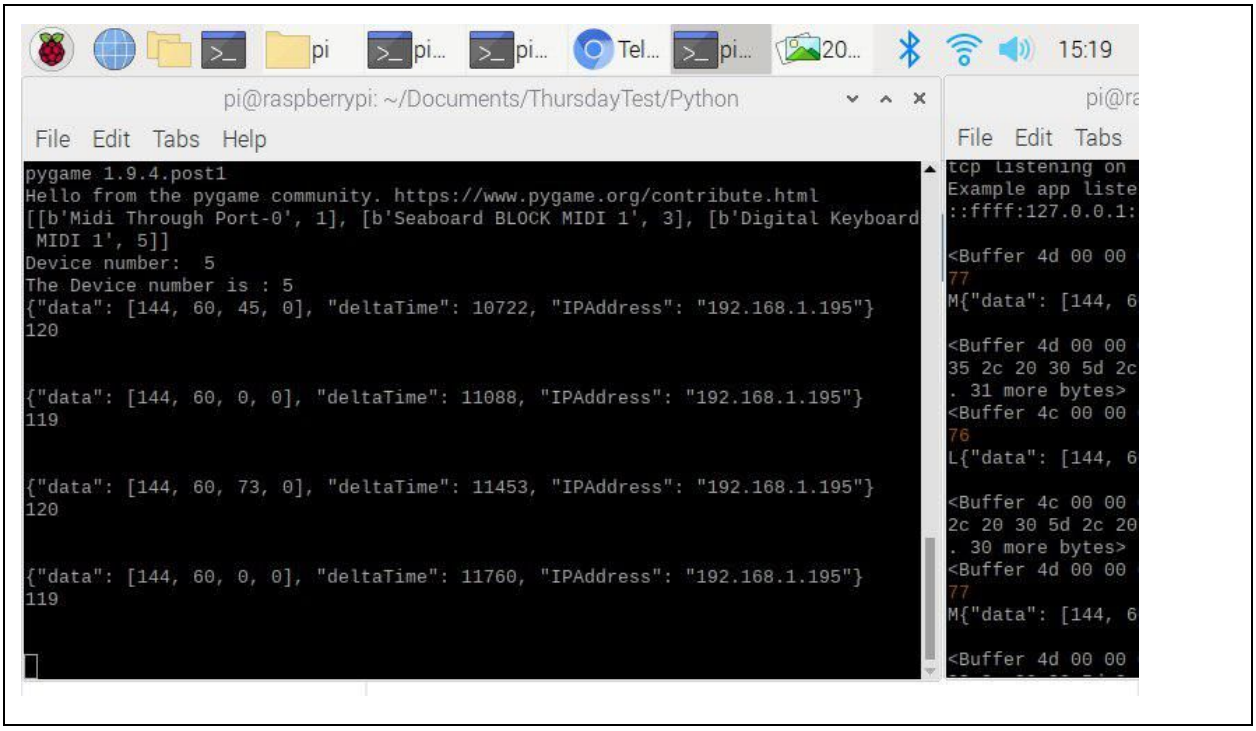

Figure 56: Midi packets formulation

Once a device is selected, the program will connect to the server and start formulating midi packets and send them over the network. The example output midi messages here.

Figure 57 shows client.cs, The unity runtime script for tcp connection handling


```
using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;
using System.Net.Sockets;
using System.IO;
using System.Linq;
using System;
using System.Text;
using Newtonsoft.Json;
using Diag = System.Diagnostics;
using Melanchall.DryWetMidi.Devices;

public class client : MonoBehaviour
{
    private bool socketReady;
    private TcpClient socket;
    private NetworkStream stream;
    private StreamWriter writer;
    private StreamReader reader;

    public Text servermessage;
    public InputField input;

    public AudioSource audioData;
    public GameObject pianoSounds;
    //public SinewaveExample piano;
    public OutputDevice midiOut;

    public PianoMIDI pianoMIDI;

    bool networkEnabled = true;

    //string host = "192.168.1.116";
    string host = "192.168.1.195";

    int port = 3001;

    // Use this for initialization
    void Start()
    {
        ConnectToServer();
        //piano = pianoSounds.GetComponent<SinewaveExample>();
        midiOut = OutputDevice.GetByName("Microsoft GS Wavetable Synth");
        midiOut.EventSent += OnEventSent;
    }

    // Update is called once per frame
    void Update()
    {
        if (socketReady)
        {
            if (stream.DataAvailable && networkEnabled)
            {
                try
                {
                    UnityEngine.Debug.Log("Got here");


                    var bytesRead = 0;
                    var offset = 0;

                    var beginBufferMessageSize = new byte[3];
```


Figure 57: client.cs

Figure 58 shows client.cs continued

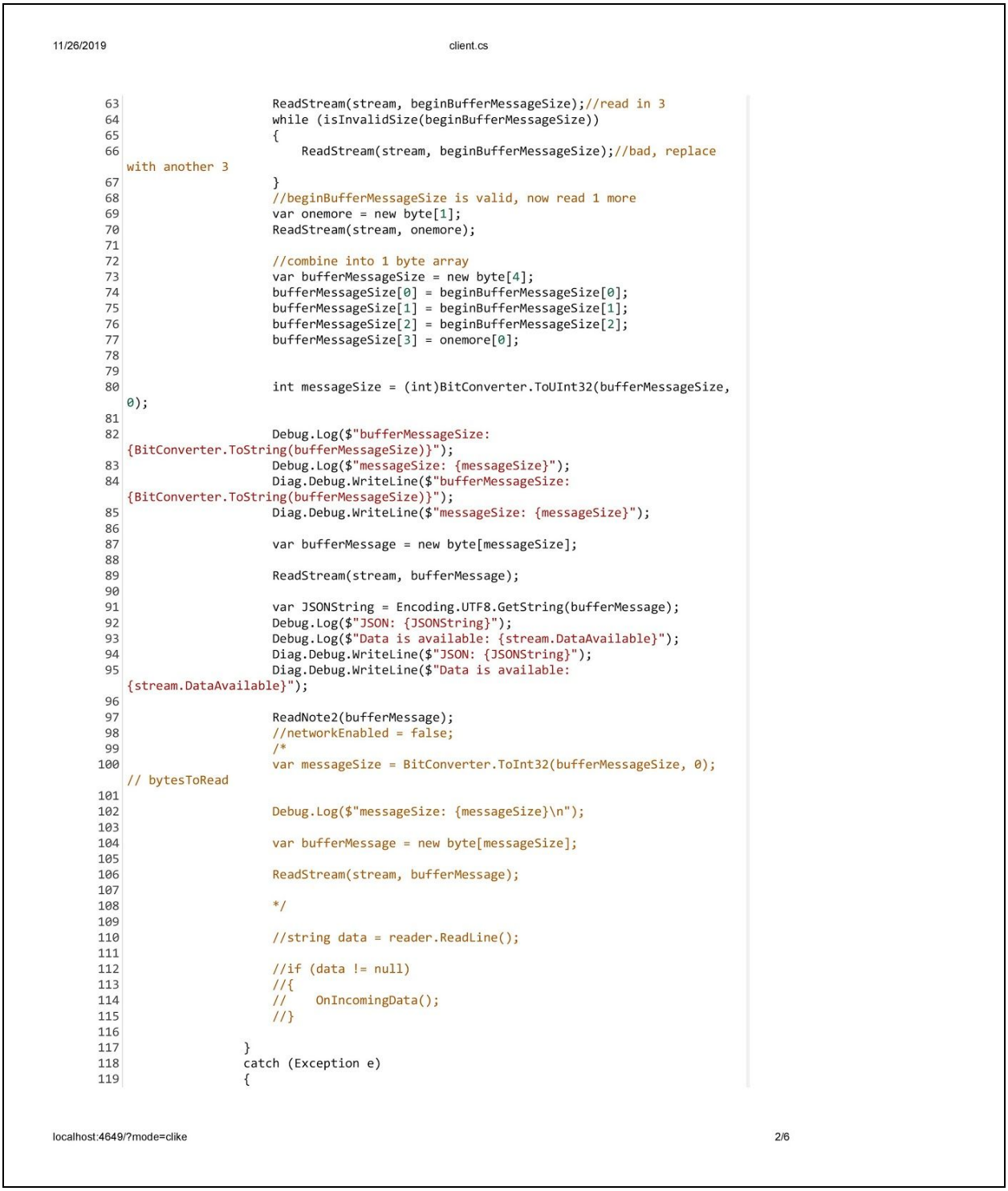

Figure 58: client.cs continued

Figure 59 shows client.cs continued


```
120                     Debug.Log("an error has occured getting data");
121                     Debug.Log(e.Message);
122                     Debug.Log(e.StackTrace);
123                     networkEnabled = false;
124                 }
125             }
126             if (Input.GetKeyDown(KeyCode.Return))
127             {
128                 OnOutgoingData("");
129                 Debug.Log("Return key was pressed.");
130                 audioData.Play(0);
131 
132             }
133         }
134     }
135 
136     public void ConnectToServer()
137     {
138         if (socketReady)
139             return;
140 
141         try
142         {
143             socket = new TcpClient(host, port);
144             stream = socket.GetStream();
145             writer = new StreamWriter(stream);
146             //reader = new StreamReader(stream);
147             socketReady = true;
148 
149             Debug.Log("Connected to server");
150         }
151         catch (System.Exception e)
152         {
153             Debug.Log("socket error :" + e);
154         }
155 
156     }
157 
158     private void ReadNote(byte[] bufferMessage)
159     {
160         //  if(data.Contains("Welcome"))
161         //  {
162         //      Debug.Log("Successfully connected to server!");
163         //      return;
164         //  }
165 
166         //  var dataStringList = data.Split(' ').ToList();
167         //  var midiStringList = dataStringList.GetRange(1, dataStringList.Count -
1);
168         //  var midiString = string.Join(" ", midiStringList);
169         List<byte> temp = new List<byte>();
170         List<Assets.Scripts.MIDIMessage> notes = new
List<Assets.Scripts.MIDIMessage>();
171 
172         for (int i = 0; i < bufferMessage.Length; i++)
173         {
174             if(bufferMessage[i] != 0x0A)
175             {
176                 temp.Add(bufferMessage[i]);
177             }
178             else
179             {
```


Figure 59: client.cs continued

Figure 60 shows client.cs continued


```
180                 notes.Add(JsonConvert.DeserializeObject<Assets.Scripts.MIDIMessage>
(Encoding.UTF8.GetString(temp.ToArray())));
181                 temp.Clear();
182             }
183         }
184 
185         //Assets.Scripts.MIDIMessage note =
JsonConvert.DeserializeObject<Assets.Scripts.MIDIMessage>(midiString);
186 
187         //Debug.Log("\n");
188         foreach(var note in notes)
189         {
190             var number = new
Melanchall.DryWetMidi.Common.SevenBitNumber((byte)note.data[1]);
191             var velocity = new
Melanchall.DryWetMidi.Common.SevenBitNumber((byte)note.data[2]);
192 
193             if (note.data[0] == 144)
194             {
195 
196                 if (note.data[2] == 0)
197                 {
198                     Debug.Log(note.data[1] + " unplayed");
199                     //piano.PauseNote(note.data[1]);
200                     midiOut.SendEvent(new
Melanchall.DryWetMidi.Core.NoteOnEvent(number, velocity));
201                 }
202                 else
203                 {
204                     Debug.Log(note.data[1] + " PLayed");
205                     //piano.PlayNote(note.data[1]);
206                     midiOut.SendEvent(new
Melanchall.DryWetMidi.Core.NoteOnEvent(number, velocity));
207                 }
208 
209             }
210             if (note.data[0] == 128)
211             {
212                 Debug.Log(note + " unplayed");
213                 //piano.PauseNote(note.data[1]);
214                 midiOut.SendEvent(new
Melanchall.DryWetMidi.Core.NoteOffEvent(number, velocity));
215             }
216 
217             //Debug.Log("server : " + data);
218             //servermessage.text = data;
219         }
220     }
221 
222     private void ReadNote2(byte[] bufferMessage)
223     {
224         List<byte> temp = new List<byte>();
225         List<Assets.Scripts.MIDIMessage> notes = new
List<Assets.Scripts.MIDIMessage>();
226 
227 
228         for (int i = 0; i < bufferMessage.Length; i++)
229         {
230             if (bufferMessage[i] != 0x0A)
231             {
232                 temp.Add(bufferMessage[i]);
233             }
234             else
```


Figure 60: client.cs continued

Figure 61 shows client.cs continued


```
235             {
236                 notes.Add(JsonConvert.DeserializeObject<Assets.Scripts.MIDIMessage>
(Encoding.UTF8.GetString(temp.ToArray())));
237                 temp.Clear();
238             }
239         }
240         foreach (var note in notes)
241         {
242             var number = new
Melanchall.DryWetMidi.Common.SevenBitNumber((byte)note.data[1]);
243             var velocity = new
Melanchall.DryWetMidi.Common.SevenBitNumber((byte)note.data[2]);
244 
245             if (note.data[0] == 144)
246             {
247 
248                 if (note.data[2] == 0)
249                 {
250                     Debug.Log(note.data[1] + " unplayed");
251                     //piano.PauseNote(note.data[1]);
252                     pianoMIDI.PauseNote(note.data[1]);
253                     //midiOut.SendEvent(new
Melanchall.DryWetMidi.Core.NoteOnEvent(number, velocity));
254                 }
255                 else
256                 {
257                     Debug.Log(note.data[1] + " PLayed");
258                     //piano.PlayNote(note.data[1]);
259                     pianoMIDI.PlayNote(note.data[1]);
260                     //midiOut.SendEvent(new
Melanchall.DryWetMidi.Core.NoteOnEvent(number, velocity));
261                 }
262 
263             }
264             if (note.data[0] == 128)
265             {
266                 Debug.Log(note + " unplayed");
267                 //piano.PauseNote(note.data[1]);
268                 pianoMIDI.PauseNote(note.data[1]);
269                 //midiOut.SendEvent(new
Melanchall.DryWetMidi.Core.NoteOffEvent(number, velocity));
270             }
271 
272             //Debug.Log("server : " + data);
273             //servermessage.text = data;
274         }
275     }
276 
277     public void OnOutgoingData(string data)
278     {
279         if (!socketReady)
280             return;
281         if (input.text == "")
282         {
283             writer.WriteLine(data);
284         }
285         else
286         {
287             writer.WriteLine(input.text);
288         }
289         writer.Flush();
290         input.text = "";
291     }
```


Figure 61: client.cs continued

Figure 62 shows client.cs continued

11/26/2019 client.cs

```
292
293     public static void ReadStream(NetworkStream reader, byte[] data)
294     {
295         var offset = 0;
296         var remaining = data.Length;
297         while (remaining > 0)
298         {
299             var read = reader.Read(data, offset, remaining);
300             if (read <= 0)
301                 throw new EndOfStreamException
302                     (String.Format("End of stream reached with {0} bytes left to
read", remaining));
303             remaining -= read;
304             offset += read;
305         }
306     }
307
308     //precondition: 3 byte array
309     //postcondition: whether or not it is valid
310     public static bool isInvalidSize(byte[] compare) {
311         if(compare[0] == 0xEF && compare[1] == 0xBF && compare[2] == 0xBD)
312         {
313             return true;
314         }
315         return false;
316     }
317
318     private void OnEventSent(object sender, MidiEventSentEventArgs e)
319     {
320         var midiDevice = (MidiDevice)sender;
321         Diag.Debug.WriteLine($"Event sent to '{midiDevice.Name}' at {DateTime.Now}:
{e.Event}");
322     }
323 }
```

Figure 62: client.cs continued

The file above, titled **client.cs,** is the routine that represents the network module for the Microsoft Hololens. It is written in the C# programming language and is intended to be used in conjunction with Unity to allow the HoloLens to connect to a Node.JS server and subscribe to notes played by a user. The **Start()** function initializes the TCP socket connection to the server using a provided IP Address. The **Update()** function checks to see if data is available on the TCP socket and if data is available, it determines the size of the message available and reads that many bytes from the stream. The **ReadNote()** function takes the bytes read in and converts them to a UTF8-encoded string that is the JSON string representing the MIDI message forwarded by the Node.JS server. It then deserializes the JSON into a C# object type. The deserialized JSON string is then used to play a sound representing the note received.

Figure 63 shows DisplayMIDI.cs


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;

public class DisplayMIDI : MonoBehaviour
{
    public Text text;
    public GameObject prefabCube, parentGameobject;

    private MIDIPlayer.MIDIFile midiFile;

    private const float START_HEIGHT = 0;
    private const int Z_COORD = 0;
    private const float NOTE_WIDTH = 22f / 128f;

    private float curY = START_HEIGHT;

    private Dictionary<int, float> noteDict = new Dictionary<int, float>(); // Key: Note Number, Value: Deltatime

    // Start is called before the first frame update
    void Start()
    {
        using (var stream = new FileStream("C:/Users/lfrit/source/repos/VMAClient/Assets/StreamingAssets/coldplay-a_sky_full_of_stars.mid", FileMode.Open, FileAccess.Read))
        {
            midiFile = new MIDIPlayer.MIDIFile(stream);
        }

        //GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
        //prefabCube.transform.position = new Vector3(0, START_HEIGHT, Z_COORD);
        //prefabCube.transform.localScale = new Vector3(NOTE_WIDTH, 1, 0.1f);
        //Instantiate(prefabCube);

        foreach (var note in midiFile.tracks[0].notes)
        {
            Debug.Log($"Delta Time: {note.deltaTime}, Note Type: {note.type} Note Number: {note.noteNumber}");
            curY += (note.deltaTime / 1000f);

            if(note.type == 1 && note.velocity > 0) //  If note on, store reference to current Y in dictionary
            {
                noteDict[note.noteNumber] = curY;
            }

            else if((note.type == 1 && note.velocity <= 0) || note.type == 2) // If note off, find difference in height and then inst. rectangle of that height
            {
                //notenumber is between 0 and 127, vison is between -11 and 11
                prefabCube.transform.position = new Vector3(((float)note.noteNumber - 64)*(11f/64f), noteDict[note.noteNumber] + ((curY - (noteDict[note.noteNumber]))/2f), Z_COORD);
                prefabCube.transform.localScale = new Vector3(NOTE_WIDTH, curY - (noteDict[note.noteNumber]), 0.1f);
                Instantiate(prefabCube, parentGameobject.transform);
            }


```


Figure 63: DisplayMIDI.cs

Figure 64 shows DisplayMIDI.cs continued

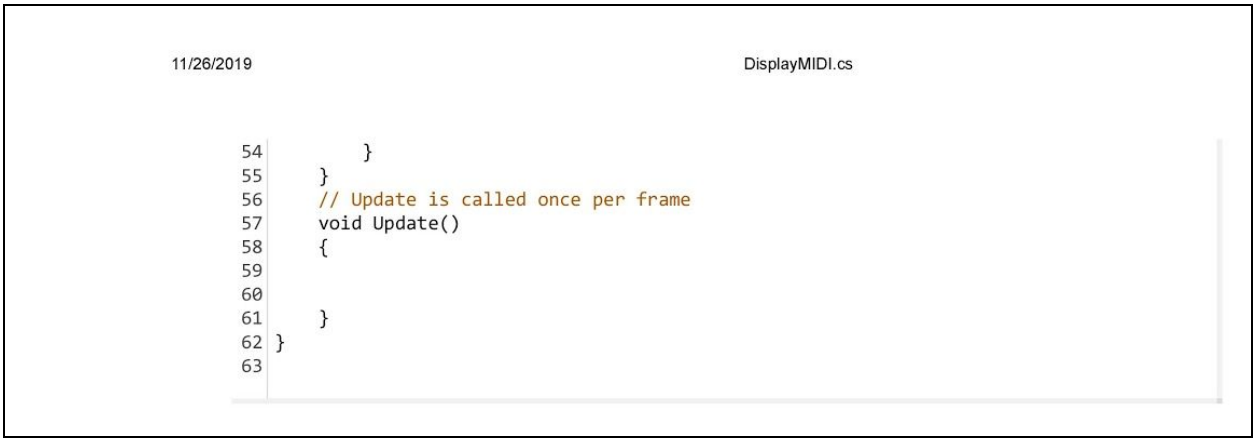

Figure 64: DisplayMIDI.cs continued

Description of DisplayMIDI.cs:

The file above, titled **DisplayMIDI.cs**, contains the code for the note visualization module to be used on the Microsoft Hololens. It is written in the C# programming language. The module starts by opening up a filestream to the desired MIDI file to be visualized and then passes that filestream to the project's MIDI parsing module in order to retrieve a C# object representing the notes contained in the desired file. Once the MIDI file object is retrieved, the module iterates over every note of the file, spawning Unity game objects representing each note. The module determines the horizontal position of the game objects based on their note number, and it determines the vertical position of the game objects based on the differences between the notes in time. A dictionary is used to store the time at which each note has an "on" press so that when an "off" press happens the difference in time can be calculated.

Figure 65 shows MidiFile.cs


```
using UnityEngine;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MIDIPlayer
{
    class MIDIFile
    {
        public int format { get; }
        public int trackCount { get; }
        public int unitsPerNote { get; }
        public List<MIDITrack> tracks = new List<MIDITrack>();

        public MIDIFile(Stream filePathIn)
        {
                using (var reader = new BinaryReader(filePathIn))
                {
                    //  First four bytes of file are ASCII string representing
chunk type
                    byte[] chunkTypeBytes = reader.ReadBytes(4);

                    string chunkType = Encoding.ASCII.GetString(chunkTypeBytes);
                    Debug.Log($"(Header) Chunk Type: {chunkType}");

                    //  Next four bytes are 32-bit integer representing length of
header (always 6 for MIDI 1.0)
                    byte[] headerDataLengthBytes = reader.ReadBytes(4);
                    int headerDataLength = Utils.BytesToInt(headerDataLengthBytes);
                    Debug.Log($"(Header) Data Length: {headerDataLength}");

                    //  For MIDI 1.0, next two bytes are the MIDI file format.
                    //  Info on formats here:
csie.ntu.edu.tw/~r92092/ref/midi/#mff0
                    byte[] formatBytes = reader.ReadBytes(2);

                    format = Utils.BytesToInt(formatBytes);
                    Debug.Log($"(Header) Format: {format}");

                    byte[] numTracksBytes = reader.ReadBytes(2);
                    trackCount = Utils.BytesToInt(numTracksBytes);
                    Debug.Log($"(Header) Number of tracks: {trackCount}");

                    byte[] divisionBytes = reader.ReadBytes(2);

                    char[] divisionBinary = (Utils.ToBinaryString(divisionBytes[0],
8) +

 Utils.ToBinaryString(divisionBytes[1], 8)
                                                ).ToCharArray();
                    char divMSB = divisionBinary[0];

                    if (divMSB == '0')
                    {
                        //  Bits 0-14 represent the number of delta-time units in
each a quarter-note.
                        char[] unitsPerNoteBinary =
divisionBinary.Skip(1).Take(15).ToArray();
                        string accumulator = "";
```


Figure 65: MidiFile.cs

Figure 66 shows a continuation of MidiFile.cs

11/26/2019 MIDIFile.cs

```
56                          for (int i = 0; i < unitsPerNoteBinary.Length; i++)
57                          {
58                              accumulator += unitsPerNoteBinary[i];
59                          }
60
61                          unitsPerNote = Convert.ToInt32(accumulator, 2);
62                          Debug.Log($"(Header) Delta-time units (ticks) per quarter-
    note: {unitsPerNote}");
63                      }
64
65                      else if (divMSB == '1')
66                      {
67                          Debug.Log("SMTPE Frames");
68                      }
69
70                      else
71                      {
72                          Debug.Log("Bits: " + string.Concat(divisionBinary));
73                          Debug.Log("MSB: " + divMSB);
74                      }
75
76                      //var track = new MIDITrack(reader, format, 1);
77
78                      for(int i = 0; i < trackCount; i++)
79                      {
80                          if (reader.BaseStream.Position != reader.BaseStream.Length)
81                          {
82                              var track = new MIDITrack(reader, format, i + 1);
83                              tracks.Add(track);
84                          }
85                      }
86
87                      while (reader.BaseStream.Position != reader.BaseStream.Length)
88                      {
89                          byte test = reader.ReadByte();
90                          Debug.Log(test.ToString("X2") + " ");
91                      }
92
93                      //while (true)
94                      //{
95
96                    //  }
97                  }
98          }
99      }
100 }
101
```

Figure 66: Continuation of MidiFile.cs

Description of MIDIFile.cs

The file above, titled **MIDIFile.cs**, is the code that implements the MIDI file parsing module to be used by the Microsoft Hololens. The module initially opens up a Binary Reader to read byte-by-byte the MIDI file passed into it. The module then reads in the various characteristics of the file contained in its header section, including the format of the file, the number of tracks contained in it and the time division of the tracks contained in it. Once the header has been completely read in, the module reads in the file track-by-track and adds each track to a track array.

Figure 67 shows MidiTrack.cs


```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

namespace MIDIPlayer
{
    class MIDITrack
    {
        const int SEQUENCE_NUMBER = 0x00;
        const int TEXT_EVENT = 0x01;
        const int COPYRIGHT_NOTICE = 0x02;
        const int SEQTRACK_NAME = 0x03;
        const int INSTRUMENT_NAME = 0x04;
        const int LYRIC = 0x05;
        const int MARKER = 0x06;
        const int CUE_POINT = 0x07;
        const int MIDI_CHANNEL_PREFIX = 0x20;
        const int END_OF_TRACK = 0x2F;
        const int SET_TEMPO = 0x51;
        const int SMTPE_OFFSET = 0x54;
        const int TIME_SIGNATURE = 0x58;
        const int KEY_SIGNATURE = 0x59;

        public int length { get; }
        public string name;
        private bool reachedEnd = false;
        private bool lastEventWasNote = false;
        public List<MIDINote> notes = new List<MIDINote>();

        public MIDITrack(BinaryReader reader, int format, int trackNumber)
        {
            name = "Track " + trackNumber.ToString();

            byte[] trackChunkBytes = reader.ReadBytes(4);
            string trackChunk = Encoding.ASCII.GetString(trackChunkBytes);

            byte[] trackLengthBytes = reader.ReadBytes(4);


            int trackLength = Utils.BytesToInt(trackLengthBytes);
            Debug.Log($"(Track {trackNumber}) Length: {trackLength}");
            length = trackLength;

            if (format == 1 && trackNumber == 1)
            {

                //  1. Time signature
                //  2. Sequence/Track Name
                //  3. Sequence Number
                //  4. Marker
                //  5. SMTPE Offset

            }
            //for(int i = 0; i < 11; i++)
            while(!reachedEnd && reader.BaseStream.Position !=
reader.BaseStream.Length)
            {
                ReadEvent(reader);
```


Figure 67: MidiTrack.cs

Figure 68 shows continuation of MidiTrack.cs


```
            }
        }

        public void ReadEvent(BinaryReader reader)
        {
            long deltaTime = Utils.ReadVariableLength(reader);
            Debug.Log($"Ticks: {deltaTime} ");
            byte identifier = reader.ReadByte();

            switch(identifier)
            {
                case 0xF0:
                    Debug.Log("Sysex Event");
                    lastEventWasNote = false;
                    break;
                case 0xF7:
                    Debug.Log("Sysex Escape");
                    lastEventWasNote = false;
                    break;
                case 0xFF:
                    ReadMetaEvent(reader);
                    lastEventWasNote = false;
                    break;
                case 0xB0:
                    //  Of the form 0xB0NN, where NN is the channel #
                    //  Followed by 0ccccccc 0vvvvvvv where c's are the controller
#
                    //  and v's are the controller value
                    Debug.Log("Control Change");
                    reader.ReadBytes(2);
                    lastEventWasNote = false;
                    break;
                case 0xC0:
                    //  Of the form 0xC0NN, where NN is the channel #
                    //  Followed by 0ppppppp where p's are the new program number.
                    Debug.Log("Program Change");
                    reader.ReadBytes(1);
                    lastEventWasNote = false;
                    break;
                case 0x0A:
                    //  Can't find much about pan values; they seem
                    //  to be one byte long.
                    Debug.Log("Pan Value");
                    lastEventWasNote = false;
                    reader.ReadByte();
                    break;
                case byte n when (n >= 0x5A && n <= 0x5F):
                    Debug.Log("Effects Depth");
                    lastEventWasNote = false;
                    reader.ReadByte();
                    break;
                default:
                    ReadMIDIEvent(identifier, reader, deltaTime);
                    break;
            }
        }

        public void ReadMIDIEvent(byte identifier, BinaryReader reader, long
deltaTime)
        {
            switch (identifier)
            {
                case byte n when (n >= 0x90 && n <= 0x9F):
```


Figure 68: Continuation of MidiTrack.cs

Figure 69 shows continuation of MidiTrack.cs 2


```
123                     int notePlayed = Utils.BytesToInt(reader.ReadBytes(1));
124                     int velocity = Utils.BytesToInt(reader.ReadBytes(1));
125                     Debug.Log($"(Note On) Note:{notePlayed}, Vel: {velocity}");
126                     notes.Add(new MIDINote(Clamp(notePlayed,0,127), Clamp(velocity,
    0, 127), 1, deltaTime));
127                     lastEventWasNote = true;
128                     break;
129                 case byte n when (n >= 0x80 && n <= 0x8F):
130                     int noteOff = Utils.BytesToInt(reader.ReadBytes(1));
131                     int offVelocity = Utils.BytesToInt(reader.ReadBytes(1));
132                     Debug.Log($"(Note Off) Note:{noteOff}, Vel: {offVelocity}");
133                     notes.Add(new MIDINote(Clamp(noteOff,0,127), Clamp(offVelocity,
    0, 127), 2, deltaTime));
134                     lastEventWasNote = true;
135                     break;
136                 default:
137                     //Debug.Log("MIDI Event");
138                     //reader.ReadBytes(2);
139                     if (lastEventWasNote)
140                     {
141                         int tmpNoteOn = (int)identifier;
142                         int tmpoffVelocity = Utils.BytesToInt(reader.ReadBytes(1));
143                         Debug.Log($"(Note Off) Note:{tmpNoteOn}, Vel:
    {tmpoffVelocity}");
144                         notes.Add(new MIDINote(Clamp(tmpNoteOn, 0, 127),
    Clamp(tmpoffVelocity, 0, 127), 1, deltaTime));
145                         lastEventWasNote = true;
146                         break;
147                     }
148                     else
149                     {
150                         Utils.ReadVariableLength(reader);
151                         break;
152                     }
153             }
154         }
155 
156         public void ReadMetaEvent(BinaryReader reader)
157         {
158             int metaIdentifier = reader.ReadByte();
159 
160             switch (metaIdentifier)
161             {
162                 case SEQUENCE_NUMBER:
163                     Debug.Log("Sequence Number");
164                     byte[] seqNumberBytes = reader.ReadBytes(3);
165                     break;
166                 case TEXT_EVENT:
167                     Debug.Log("Text Event: ");
168                     string text = GetTextData(reader);
169                     Debug.Log(text);
170                     break;
171                 case COPYRIGHT_NOTICE:
172                     Debug.Log("Copyright Notice: ");
173                     string cprtText = GetTextData(reader);
174                     Debug.Log(cprtText);
175                     break;
176                 case SEQTRACK_NAME:
177                     Debug.Log("Sequence/Track Name: ");
178                     name = GetTextData(reader);
179                     Debug.Log(name);
180                     break;
181                 case INSTRUMENT_NAME:
```


Figure 69: Continuation of MidiTrack.cs 2

Figure 70 shows continuation of MidiTrack.cs 3


```
                    Debug.Log("Instrument Name: ");
                    string instrName = GetTextData(reader);
                    Debug.Log(instrName);
                    break;
                case LYRIC:
                    Debug.Log("Lyric(s): ");
                    string lyrics = GetTextData(reader);
                    Debug.Log(lyrics);
                    break;
                case MARKER:
                    Debug.Log("Marker: ");
                    string marker = GetTextData(reader);
                    Debug.Log(marker);
                    break;
                case CUE_POINT:
                    Debug.Log("Event Cue: ");
                    string cue = GetTextData(reader);
                    Debug.Log(cue);
                    break;
                case MIDI_CHANNEL_PREFIX:
                    Debug.Log("MIDI Channel Prefix");
                    byte[] channelPrefixBytes = reader.ReadBytes(2);
                    break;
                case END_OF_TRACK:
                    reader.ReadByte();
                    Debug.Log("End of Track");
                    reachedEnd = true;
                    break;
                case SET_TEMPO:
                    byte[] setTempoBytes = reader.ReadBytes(4);
                    Debug.Log("Set Tempo");
                    break;
                case SMTPE_OFFSET:
                    byte[] SMTPEBytes = reader.ReadBytes(6);
                    Debug.Log("SMTPE Time");
                    break;
                case TIME_SIGNATURE:
                    Debug.Log("Time Signature");
                    byte[] timeSignatureBytes = reader.ReadBytes(5);
                    break;
                case KEY_SIGNATURE:
                    byte[] keySignatureBytes = reader.ReadBytes(3);
                    Debug.Log("Key Signature");
                    break;
                default:
                    Debug.Log("Unknown Meta Event");
                    int length = (int)Utils.ReadVariableLength(reader);
                    reader.ReadBytes(length);
                    break;
            }
        }

        public string GetTextData(BinaryReader reader)
        {
            int textLength = (int)Utils.ReadVariableLength(reader);
            byte[] textBytes = reader.ReadBytes(textLength);
            return Encoding.ASCII.GetString(textBytes);
        }

        public int Clamp(int n, int min, int max)
        {
            if(n < min)
            {
```


Figure 70: Continuation of MidiTrack.cs 3

Figure 71 shows continuation of MidiTrack.cs 4

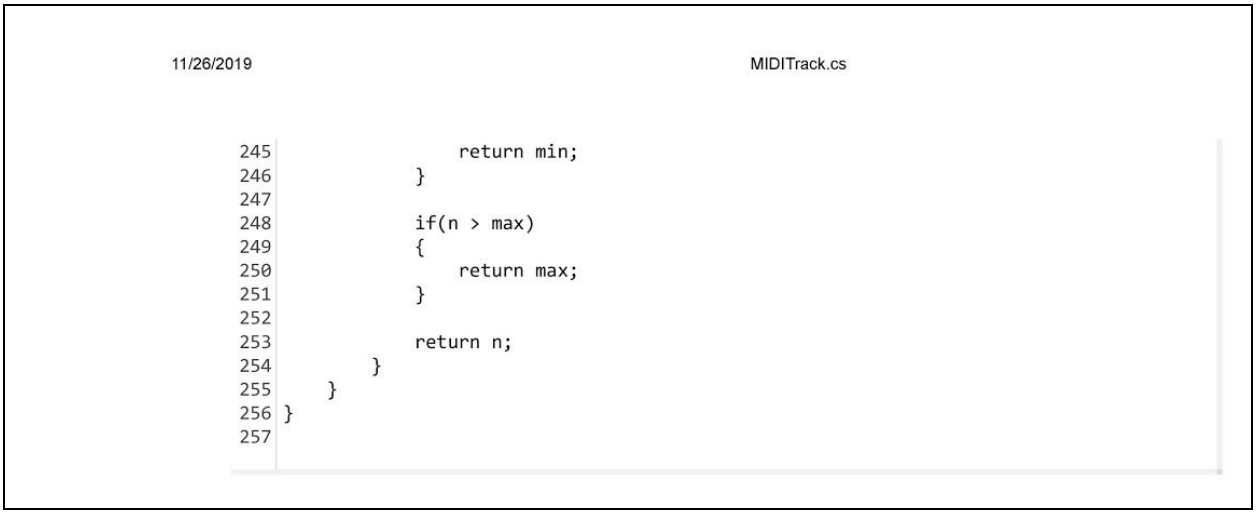

Figure 71: Continuation of MidiTrack.cs 4

Description of MidiTrack.cs

The file above, titled **MIDITrack.cs**, contains the code for the MIDI track parsing module to be used by the Microsoft Hololens. It continuously reads in a delta time byte array followed by an event identifier byte. The module uses the identifier byte to determine whether the event following it is a Meta, Sysex, or MIDI event and then reads in the event. In any case where the event read is not specifically a Meta "End of Track" event or a MIDI "Note On" or "Note Off" event, the event is simply logged to the program output for debugging purposes. In the case where the event read is a Meta "End of Track" event, the parsing of the current track is ended and the current track object is added to the MIDI File object associated with it. When a MIDI "Note On" or "Note Off" event is read, a MIDI Note object is added to the MIDI Note object array attached to the current track object.

Figure 72 shows MidiNote.cs

11/26/2019 MIDINote.cs

```
1  using UnityEngine;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace MIDIPlayer
9  {
10     public class MIDINote
11     {
12         public int noteNumber;
13         public int velocity;
14         public int type;
15         public long deltaTime;
16
17         public MIDINote(int noteNumber, int velocity, int type, long deltaTime)
18         {
19             this.noteNumber = noteNumber;
20             this.velocity = velocity;
21             this.type = type;
22             this.deltaTime = deltaTime;
23
24         }
25     }
26 }
27
```

Figure 72: MidiNote.cs

Description of MidiNote.cs

The file above, titled **MIDINote.cs**, is the C# class code representing the MIDI notes read in by the MIDI track class. It contains a note number that corresponds to the key to be pressed on the piano, a velocity number that represents how hard the piano key should be pressed, a type number that identifies whether the piano key should be pressed or released, and a delta time number representing how long to wait before the piano key is pressed.

Level 0 block diagram w/ functional requirements table LF BG DK KV

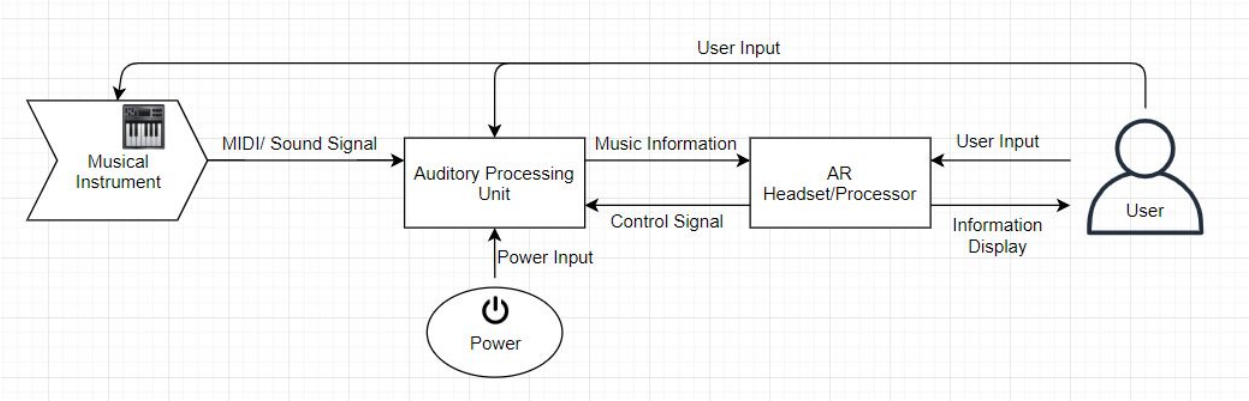

Figure 73: Level 0 block diagram (entire system)

Table 7: Functional Requirements Table 1

Module	Music Recording Module
Designers	Larry Fritz, Bridger L. Garman, David Klett, Kyle Vasulka
Inputs	Power: 120 volts AC rms, 60Hz. Audio input signal: ?V peak. Control digital input signal: ?V peak Programming port: USB; User inputs
Outputs	WiFi 802.11ac OR Bluetooth 4.2: Web Server Audio Jack: ?V peak value
Description	Processes the signal from the instrument. Once processed, an algorithm will use music theory to suggest notes, chords, and the general key that would sound good(as determined by music theory). This information will be sent to and displayed on the AR headset.

Table 8: Functional Requirements Table 2

Module	Alternate Reality Software
Designers	Larry Fritz, David Klett, Kyle Vasulka
Inputs	WiFi 802.11ac OR Bluetooth 4.2: Tracking data User inputs
Outputs	WiFi 802.11ac OR Bluetooth 4.2: Control signal AR Display
Description	Analyze data received from music module and display information to the user using software analytics.

Figure 74: Mechanical sketch of system KV

6. Team Information DK

Table 9: Team Information

Name	Major	Embedded?
Larry Fritz	CpE	Yes
Bridger Garman	EE	No
David Klett	CpE	Yes
Kyle Vasulka	EE/CpE	Yes

7. Parts List

Figure 75 shows the Build of materials for the Visual Music Assistant

DT #_____10 Project Leader:____Kyle Vasulka_____ Email: kjv13@zips.uakron.edu

Qty.	Refdes	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Hyperlink	Cost	Total Cost
1	N/A	83-16872	Element 14 7" Pi Touchscreen LCD Display	MicroCenter	640522710386	https://www.microcenter.com/prc	60.00	60.00
1	N/A	CMC-6036-40T	MIC COND ANALOG OMNI -40DB 6MM	Digi-Key	102-4100-ND	https://www.digikey.com/short/p	1.49	1.49
1	U	TLV2252AIP	IC OPAMP GP 2 CIRCUIT 8DIP	Digi-Key	296-7431-5-ND	https://www.digikey.com/short/p	4.68	4.68
1	U	MCP1700-3302E/	IC REG LINEAR 3.3V 250MA TO92-3	Digi-Key	MCP1700-3302E/TO-N	https://www.digikey.com/short/p	0.38	0.38
1	D	SA6.0A	TVS DIODE 6V 10.3V DO204AC	Digi-Key	SA6.0ALFCT-ND	https://www.digikey.com/short/p	0.36	0.36
3	C	C330C105K5R5T	CAP CER 1UF 50V X7R RADIAL	Digi-Key	399-4389-ND	https://www.digikey.com/short/p	0.47	1.41
20	C	FG28X5R1E106M	CAP CER 10UF 25V X5R RADIAL	Digi-Key	445-181284-1-ND	https://www.digikey.com/short/p	0.32	6.40
2	C	FK26X5R0J476M	CAP CER 47UF 6.3V X5R RADIAL	Digi-Key	445-8540-ND	https://www.digikey.com/short/p	0.46	0.92
10	R	CF14JT4K70CT-N	RES 4.7K OHM 1/4W 5% AXIAL	Digi-Key	CF14JT4K70	https://www.digikey.com/short/p	0.04	0.40
20	R	CF14JT47K0	RES 47K OHM 1/4W 5% AXIAL	Digi-Key	CF14JT47K0CT-ND	https://www.digikey.com/short/p	0.04	0.80
1	N/A	RASPBERRY PI 4	RASPBERRY PI 4 MODEL B 4GB SDRAM	Digi-Key	1690-RASPBERRYPI4	https://www.digikey.com/short/p	68.75	68.75
1	U	DSPIC33FJ256GI	IC MCU 16BIT 256KB FLASH 100TQFP	Digi-Key	DSPIC33FJ256GP710A	https://www.digikey.com/short/p	7.32	7.32
1	U	SG-210STF 7.500	XTAL OSC XO 7.5000MHZ CMOS SMD	Digi-Key	SER3881-ND	https://www.digikey.com/short/p	1.44	1.44
								0.00
							Total	$154.35

Figure 75: Build of material list

Figure 76 shows the cost of the Build of materials for the Visual Music Assistant

Qty.	Part Num.	Description	Unit Cost	Total Cost
1	CMC-6036-40T	MIC COND ANALOG OMNI -40DB 6MM	1.49	1.49
1	TLV2252AIP	IC OPAMP GP 2 CIRCUIT 8DIP	4.68	4.68
1	83-16872	Element 14 7" Pi Touchscreen LCD Display	60.00	60.00
1	MCP1700-3302E/	IC REG LINEAR 3.3V 250MA TO92-3	0.38	0.38
1	SA6.0A	TVS DIODE 6V 10.3V DO204AC	0.36	0.36
3	C330C105K5R5T	CAP CER 1UF 50V X7R RADIAL	0.47	1.41
20	FG28X5R1E106M	CAP CER 10UF 25V X5R RADIAL	0.32	6.40
2	FK26X5R0J476M	CAP CER 47UF 6.3V X5R RADIAL	0.46	0.92
10	CF14JT4K70CT-N	RES 4.7K OHM 1/4W 5% AXIAL	0.04	0.40
20	CF14JT47K0	RES 47K OHM 1/4W 5% AXIAL	0.04	0.80
1	RASPBERRY PI 4	RASPBERRY PI 4 MODEL B 4GB SDRAM	68.75	68.75
1	DSPIC33FJ256GI	IC MCU 16BIT 256KB FLASH 100TQFP	7.32	7.32
1	SG-210STF 7.500	XTAL OSC XO 7.5000MHZ CMOS SMD	1.44	1.44
				0.00
			Total	$154.35

Figure 76: Cost of Build of material list

8. Project Schedules LF BG DK KV

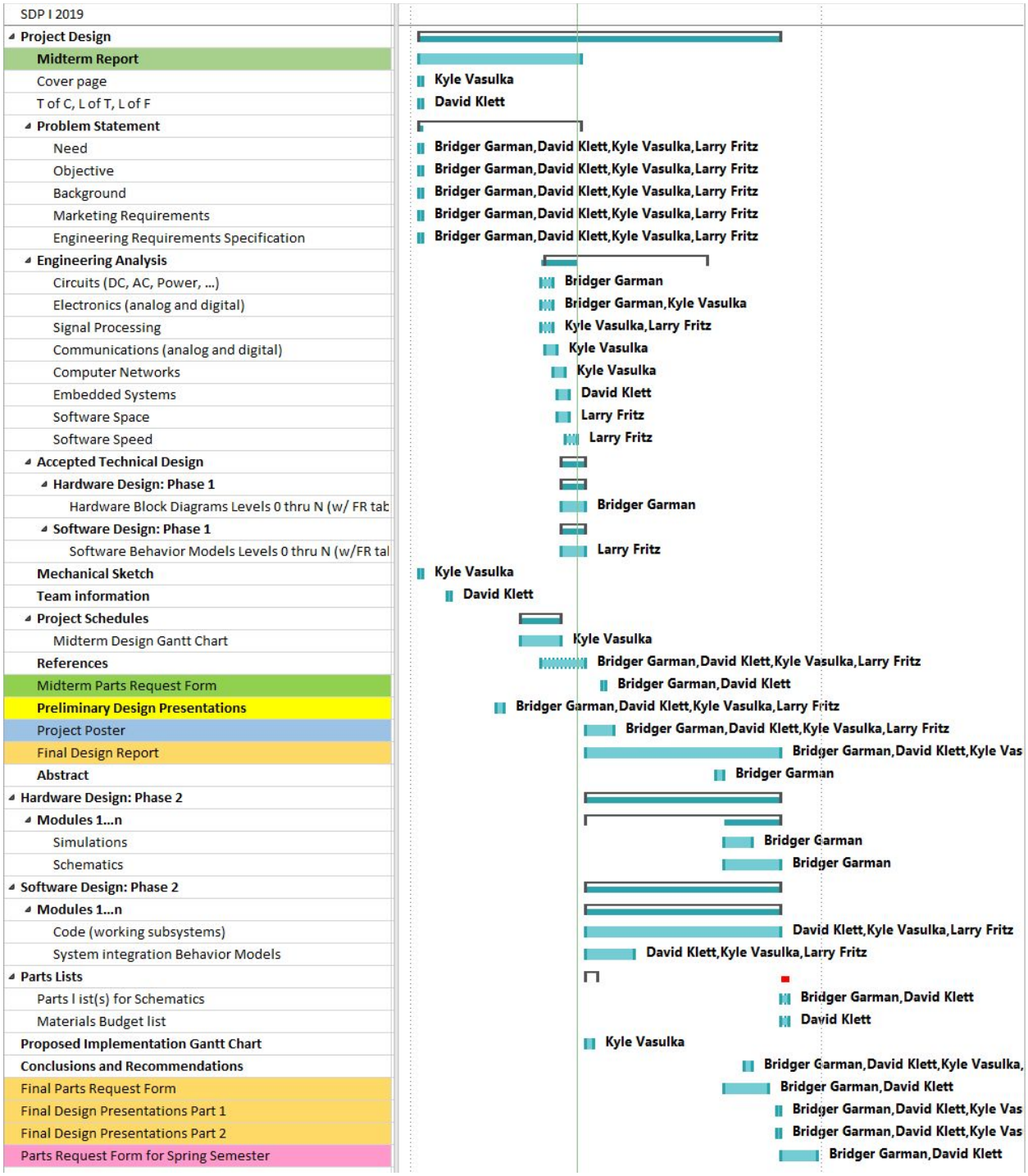

Figure 77: Gantt Chart

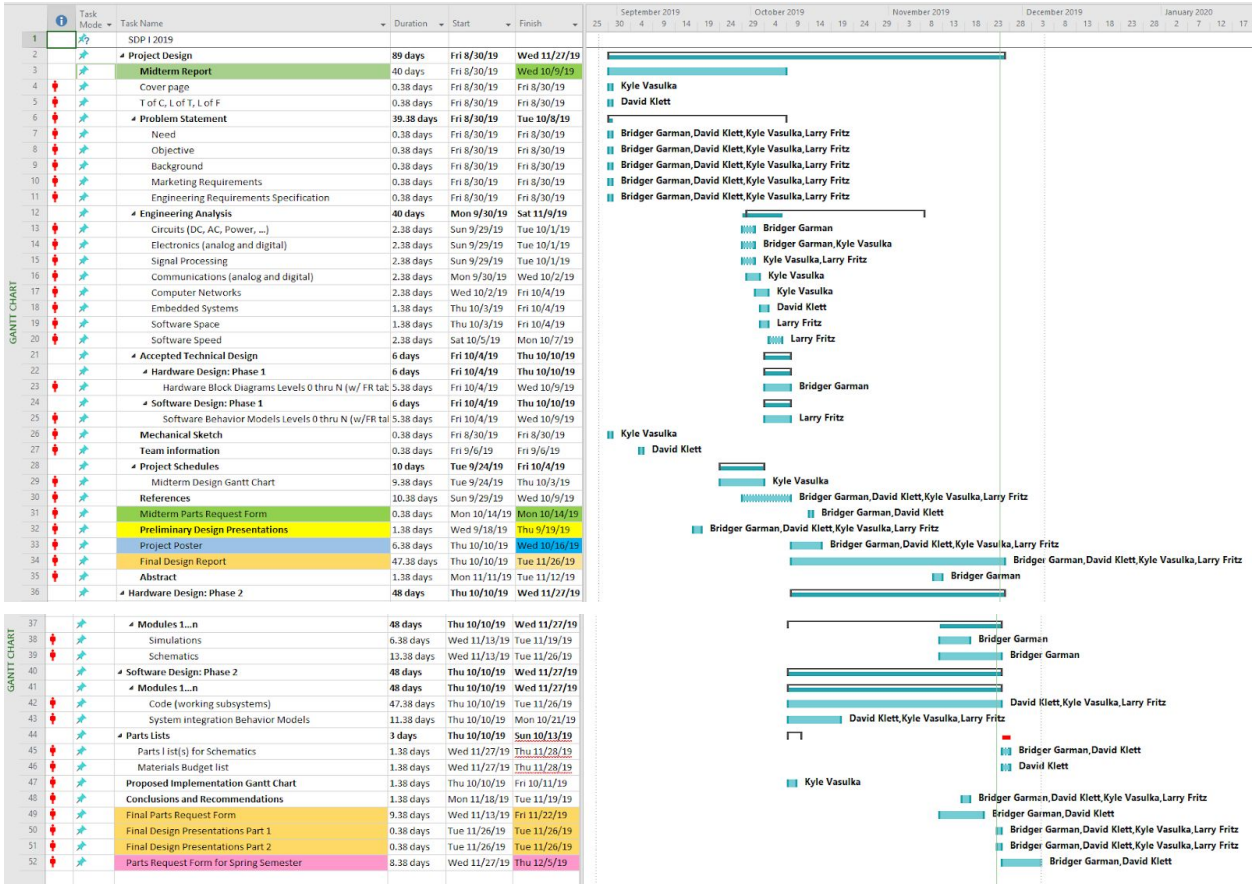

Figure 78: Updated Gantt chart (11/26/29, put in by David)

Conclusion KV

This product will be used by someone who wants to learn how to play a keyboard. The user will wear an AR headset and sit in front of a keyboard. They will go through a calibration process to tell the AR headset where the keyboard is located. The user can then take lessons from the product with lit up keys that indicate what the user should play. Alternatively, the user can play freely and have the product suggest notes and chords that would sound good to play as determined by the music theory algorithm. The final demonstration will include a keyboard setup and an AR headset.

References

Bryant, Sharon. (2014, June 9). How Children Benefit from Music Education in Schools. Retrieved March 18, 2019, from https://www.nammfoundation.org/articles/2014-06-09/how-children-benefit-music-education-schools

Chen, S. (2018). A Study on Integrating Augmented Reality Technology and Game-based Learning Model to Improve Motivation and Effectiveness of Learning English Vocabulary. Retrieved March 11, 2019, from https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/stamp/stamp.jsp?tp=&arnumber=8567161

ChildTrends DataBank (November 2015). Participation in School Music or Other Performing Arts. Retrieved from https://www.childtrends.org/wp-content/uploads/2015/11/36_Participation_in_Performing_Arts1.pdf

Colpani, R. (2015, July 8). An innovative augmented reality educational framework with gamification to assist the learning process of children with intellectual

disabilities. Retrieved March 11, 2019, from https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/document/7387964?arnumber=7387964&SID=EBSCO:edseee

Eady, F. (2005, August 18). Implementing 802.11 with Microcontrollers. Retrieved March 11, 2019, from https://ebookcentral.proquest.com/lib/uakron/detail.action?docID=270061

Hamalainen, P. (2018). *U.S. Patent No. US20180336871A1*. Washington, DC: U.S. Patent and Trademark Office.

Hart. (2016, January 12). Hart. "How Much Should I Be Charging For Lessons?". Retrieved March 18, 2019 from http://www.rgt.org/blog/tuition-fees-how-much-should-i-be-charging-for-lessons

Huang, F. (2011, October 10). Piano AR: A Markerless Augmented Reality Based Piano Teaching System. Retrieved March 11, 2019, from https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/document/6038212

Kaipainen, M. (2014). *U.S. Patent No. US20140041511A1*. Washington, DC: U.S. Patent and Trademark Office.

Klapuri, A. (2017). *U.S. Patent No. US9767705B1*. Washington, DC: U.S. Patent and Trademark Office.

Klapuri, A. (2019). *U.S. Patent No. US10182093B1*. Washington, DC: U.S. Patent and Trademark Office.

Meet the New Raspberry Pi 4, Model B. (n.d.). Retrieved from https://www.hackster.io/news/meet-the-new-raspberry-pi-4-model-b-9b4698c24 .

(n.d.). Retrieved from https://www.csie.ntu.edu.tw/~r92092/ref/midi/midi_channel_voice.html.

(n.d.). Retrieved from https://www.csie.ntu.edu.tw/~r92092/ref/midi/midi_channel_voice.html

Narayan, S. (2015, January 8). Performance test of IEEE 802.11ac wireless devices. Retrieved March 11, 2019, from https://ieeexplore.ieee.org/document/7218076

Synthesia LLC. (2019, January 29). Synthesia, Piano for Everyone. Retrieved March 18, 2019, from https://www.synthesiagame.com/

Yousician. (2019, March 18). Yousician | Learn to Play | Your Personal Music Teacher. Retrieved from https://yousician.com/#

Texas Instruments Inc (May 2002). Fixed Point Math Library. Retrieved from: http://www.ece.uidaho.edu/hydrofly/OLD/Code/math/doc/math_mdl.pdf

Microchip Technology Inc (2008). dsPIC30F/33F Programmer's Reference Manual. Retrieved from: http://ww1.microchip.com/downloads/en/devicedoc/70157c.pdf

Wikipedia (2019). Discrete Fourier transform. Retrieved from: https://en.wikipedia.org/wiki/Discrete_Fourier_transform