ECE/COE 1896

Senior Design

Music Notation to Audio Sheet Music Trainer Conceptual Design

Team 7

Prepared By: Jack Carnovale
Chloe Hale
Cameron Henning

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The Music Notation to Audio Sheet Music Trainer is a standalone device designed to help new musicians learn to read and play sheet music. A user can place one sheet of music in front of the device's camera, and through computer vision, the device will translate the piece into sound.

In the world of music, sheet music is the international standard. It is the global language of musicians. Some pieces can be played by ear, but this becomes difficult quickly if the piece is not simple. Therefore, learning how to read and play from sheet music, or sight read, is key to anyone practicing an instrument.

However, this process can be challenging, especially if someone is learning by themselves. There are plenty of videos and articles online to help learn the notation, but when using this method to learn a piece of the musician's choice, it can add inconveniences. It requires a musician to keep these references up with their piece and learn by switch tasking. This method doesn't allow a user to follow along audibly and know the timing and sound of when notes are played. There are also apps that can scan sheet music and allow a user to read and hear the music digitally. These solutions resolve many of the problems associated with having reference articles and videos up when a musician is learning a piece of music. However, these methods only work best when a user has a large screen available.

The Music Notation to Audio Sheet Music Trainer aims to address and correct these shortcomings. It will allow a user to follow along with the sheet music visually and audibly. It will allow the user to learn on their printed copy instead of having to digitize it. It will have blinking LED lights to follow and highlight which note is being played on the sheet. At the same time as a note is played, not only will an LED light, but the letter note being played will appear on a small LCD screen as well at the top of the device. This allows the user to have all information needed to learn how to play the piece in one place.

The device will work as follows. A user will place the sheet music within the appropriate area on the device in front of the camera. They will use a dial to adjust playback rate to allow for learning at a slower pace, and when ready, they will press the play button. At this point, an image of the sheet music will be sent to a Raspberry Pi microprocessor which will use computer vision to transcribe it into code. Once the processing is complete, a metronome sound will click 3 times signaling the start of the playback. From there, the Pi will light LEDs on which line and measure the note is being played, put notes on the screen, and send MIDI commands to an Arduino microcontroller to do audio playback. This will be a portable assembly large enough to place a sheet on, and it will have a rechargeable battery. This Music Notation to Audio Sheet Music Trainer will be designed, constructed, and tested in the manner outlined by this document.

2. Background

The Music Notation to Audio Sheet Music Trainer presents a solution to the problem that learning to sight read sheet music is difficult. Learning to read sheet music, or score, is like learning another language. There are unfamiliar symbols, letters, and other parts of music theory

embedded in the score that a learning musician may not understand [1]. Other difficulties arise with keeping with the timing of the piece or memorizing how to play the same few pieces [2]. However, score is the universal language of music. It is the means for clear communication of musical information. It allows a musician to play any piece of music that they want, and it allows them to learn it quickly [3]. It allows for easy transcription and coordination of complex pieces like symphonies and band performances, as well as transcription of an individual's own compositions. Last, it is universal and highly transferable for all instruments [4].

Not being able to read sheet music is a detriment to someone's study of music, and the study of music is common in the United States. According to a study from the NAMM (National Association of Music Merchants) based on a report from the National Center for Education Statistics, 47% percent of all students participate in music education in middle school and 18% in high school during the school day [5]. In another study from NAMM, 52% of US households have at least one person who is currently playing a musical instrument, and 40% of US households report having two or more persons actively involved in playing music [6]. Playing music and thus learning to play music is something that is deeply embedded in American culture and society. Therefore, a sheet music trainer would be a device that could benefit people all over America.

There are already some existing solutions to turning sheet music to audio to learn to sight read. Web based programs like ScanScore [7] and SmartScore64 [8] can turn sheet music into audio. They allow users to take a pdf scan or an image and import it into an editing software. These software options allow users to playback and edit the score they import. SmartScore64 even has MIDI playback. There are also some apps that allow for a similar type of interaction. Apps like SheetMusicScanner [9], PlayScore 2 [10], and ScanScore [11]. These apps fundamentally do the same thing as the web based solutions. They allow for scanning, playback at different speeds, and editing of sheet music. While these are all effective and portable solutions, the drawback is that they are entirely digital solutions. All playback and reading is done digitally and through a screen whereas most bands, orchestras, and private players use paper sheet music. Furthermore, according to an article from Scientific American, the brain psychologically prefers paper learning. The article cites the results of an experiment done in 2011 at the Technion - Israel Institute of Technology. The experiment showed that studying and learning using paper documents leads to more effective learning and a deeper memory in the subject area as compared to using computer documents [12]. In this way, a solution is needed where the learning is done on paper. This is what the solution proposed in this document will do by playing audio with the paper sheet still in front of the user.

There are some other examples online of other students and hobbyists algorithmically implementing music notation to audio. All examples refer to the process of reading in and processing the music notation as Optical Music Recognition (OMR). In one example from the Stanford Vision Lab, three different machine learning algorithms were used: Gaussian SVM, linear SVM, and K-nearest neighbors. This software was only limited to whole, half, and quarter notes, and only used grayscale images. While K-nearest neighbors produced some error and linear SVM produced high error, the Gaussian SVM produced a low error of only 2.2% [13]. Another example uses OpenCV and processes images using noise reduction and binarization before classifying notes. This classifier is also limited to whole to eighth notes. This example finds success through template matching which is far less intensive in terms of computation, but it had roughly a 20% error on a simple song ("Mary Had a Little Lamb") [14]. Another

implementation done by McGill University uses a sliding frame and a convolutional neural network to classify discrete notes in score. This algorithm was trained with the large, open source PrIMus dataset for score and got a symbolic error rate of less than 1% and a sequence error rate of around 20% which considers accuracies of whole musical sequences instead of individual symbols [15].

All of these examples provide insight into the possibilities for classification, but they all have a few things in common that will be corrected by The Music Notation to Audio Sheet Music Trainer. First, they all have to be run on a computer whereas this project will operate on a Raspberry Pi, which will limit algorithm complexity but allow for it to function as a standalone device. There is an example of someone using a Raspberry Pi online for a project called MusicalPi, but this project requires sheets to be pdf scans imported in [16]. Also, any MIDI playback for these projects is done through libraries which takes up processing power whereas this senior design project will use a microcontroller with stripped functionality to make fast and simple MIDI output. Last, these are not built for training someone to practice sheet music. They just read and playback the music.

To understand the solution outlined in this paper, there is some important information that must be understood first. Primarily, to understand The Music Notation to Audio Sheet Music Trainer, the basics of sheet music must be understood. To begin, all notes reside on lines called a staff. At the beginning of the staff is the clef and the key signature. The key signature used in this project is the treble clef, or the G clef. After the clef is the key signature or the time signature. The key signature is denoted through either sharps (pound symbol) or flats (b shaped symbol). The sharps or flats before the time signature will display which note is either sharp or flat in the key signature. For example, if a flat is located at a B, third line from the bottom, then all B's within the piece are flat no matter the octave. If there is no notation before the key signature, the key of C is used in the piece and no notes are sharp or flat. The following key signatures will be used in the Sheet Music Trainer: C major (all natural), G major (F sharp), D major (F and C sharp), F major (B flat), and B flat major (B and E flat). Additionally, an accidental is a sharp of a flat that is indicated within a measure but not within a key signature. An accidental, if marked at the beginning of a measure, will be carried out throughout the measure unless a natural symbol is indicated. The time signature indicates the notes per measure. For example, 2/4 time signature indicates there are four beats in a measure with one quarter note per two beats. The following time signatures used in the Sheet Music Trainer: 4/4, 3/4, 2/4, cut time, and common time. Figure 1 displays the following concepts mentioned above.

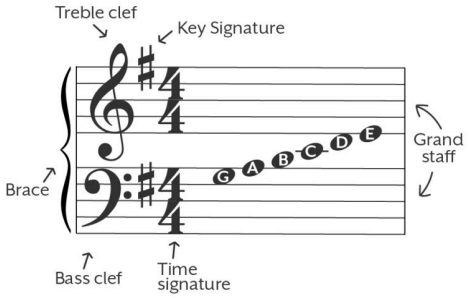


Figure 1: Beginning of the Staff Piano Reference [17]

Another aspect to sheet music that will be implemented within the device is rhythms and dynamics. Rhythms are denoted through the combination of different notes and rests. Figure 2 displays the different notes and rests that will be accounted for in the prototype. Additionally, if there is a dot next the note indicates that an additional half of the note's value is added to its beat. For example, a quarter note dot is equal to one and a half beats. Lastly, dynamics play a key role in understanding the flow and emotion of a piece. Dynamics can be broken down into two sections loud (forte) and soft (piano). Forte is depicted by a cursive f and can have different levels of volume. Piano is depicted by a cursive p and can have different levels of volume as well. This can be visually seen in Figure 3, where the figure labels and describes the dynamics as a volume meter. The following dynamics will be accounted for in the prototype: forte, fortissimo, piano, and pianissimo. Additionally, sheet music can display an increase in volume through crescendos (depicted as two lines parting) and a decrease in volume through decrescendos (depicted as two lines converging). Crescendos and decrescendos will be simulated in the Music Notation to Audio Sheet Music Trainer as well.

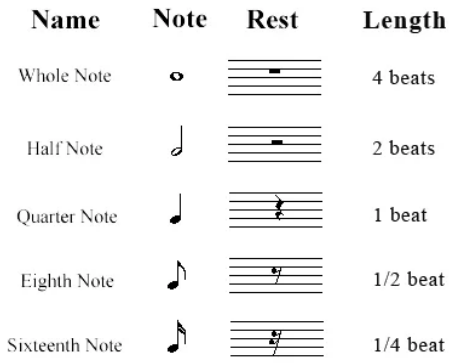


Figure 2: Note Reference [18]

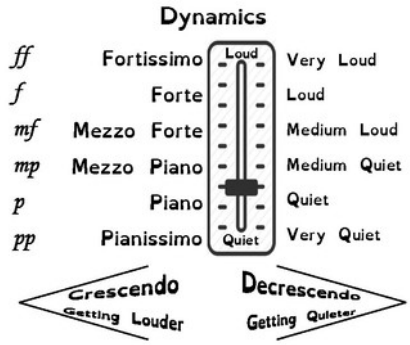


Figure 3: Dynamics Reference [19]

Regarding music, modified MIDI communication will be used in this project. At the basic level it will be used for this project, MIDI works according to the following

communication protocol. Commands are sent in strings of 3 bytes. The first byte says to turn a note on or off as well as the number of the instrument channel being used. Then there are two data bytes. The first data byte 0xPP being the pitch of the note from 0 to 127. The second data byte 0xVV is the velocity or volume of the note [20]. This communication will be the commands that the audio microcontroller will see.

There are some other important technical aspects to The Music Notation to Audio Sheet Music Trainer. To begin, this project is fundamentally based in the theory of Computer Vision. Computer Vision is the part of computer science that is based around using software to process images and the pixels within them to recognize objects [21]. For this project, Computer Vision will be used to identify and extract the different notes and symbols within the sheet music in the order that they appear.

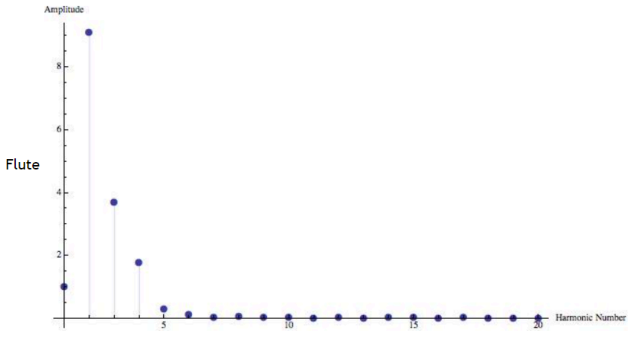


Figure 4: Flute Harmonic Spectrum [22]

For audio, spectral analysis using the Fast Fourier Transform will be conducted. This concept works on the basis that any analog signal can be constructed by a summation of scaled sinusoids that have frequencies at multiples of a fundamental frequency [23]. For a given instrument, each note has the same Fourier representation but with a different fundamental frequency. This is based on the physical construction of the instrument [22]. This concept will be the basis for signal reconstruction in this project.

The Music Notation to Audio Sheet Music Trainer aims to use these concepts and improve on these examples to make a useful device for practicing sight reading.

3. System Requirements

3.1 Hardware

3.1.1 Power Supply Requirements

3.1.1.1 Operation on Battery Supply

The system will have the ability to operate on battery supply when the initial power supply (e.g 120VAC) is removed. This requirement is considered a desired feature in the eyes of a customer since having a backup battery allows the customer to make the device more mobile, however this is a requirement for the design module. The system is required to operate on solely battery power for at least one hour of operation. This is due to the average practice time being an hour along with the estimated battery life span being around two hours.

$$Battery\ Life\ =\ \frac{Battery\ Capacity\ (mAh)}{Load\ Current\ (mA)}$$

Equation 1: Calculating Battery Life [24]

3.1.1.2 Rechargeability

The system must be able to recharge the battery when powered by the initial power supply while the system is operating and when plugged into an outlet and the power switch is open. Additionally, the system must recognize when the battery has reached maximum capacity and stop charging. At the moment, there are no requirements for the time necessary to charge the battery as well. This requirement shall be able to fulfill the requirements in the previous section (3.1.1.1).

3.1.2 Audio Requirements

It is required that the microcontroller MIDI synthesizer takes MIDI commands and plays a tone at the frequency associated with that note within some error. The bounds for this frequency range are as follows:

$$Lower\ Bound\ =\ f_0\frac{1+\frac{1}{2^{1/12}}}{2} \qquad\qquad Upper\ Bound\ =\ f_0\frac{1+2^{1/12}}{2}$$

Equation 2: Lower Bound for Audio Requirements

Equations 3: Upper Bound for Audio Requirements

It is required that the MIDI synthesizer can play tones from C4 to C6 (261-1046 Hz). These are the notes that comprise the treble clef. It is also required that the audio filter has a 3dB bandwidth of at most 22000 Hz.

It is desired that the MIDI synthesizer can play tones for the whole standard audio range of a piano (27.5 - 4186 Hz).

3.1.3 Microprocessor I/O Requirements

It is required that the Raspberry Pi microcontroller can read sheet music off of any user inserted USB drive. It is required that there are “Play” and “Process” buttons connected to a digital input. It is required that the Raspberry Pi has a serial communication path to the MIDI synthesizer, and it is required that the Raspberry Pi can send a signal to trigger audio output on the MIDI synthesizer.

It is also desired that the user can input analog selections for instrument and playback speed by turning potentiometers. These potentiometers will produce analog signals that will be converted into digital signals for the Pi to read. It is also desired that when audio is played that an LED will blink and an LCD screen will show the corresponding note played. It is desired that the device will have an LED turn on where the note is on the page.

3.1.4 Communication Requirements

The standard for MIDI communication is a baud rate of 31,250 bits per second [25]. This will thus define the standard for all controls and communication within the device. It is required that SPI communication between the Raspberry Pi and the MIDI synthesizer is at least at 31,250 bits per second.

However, it is desired that output lag for audio and visuals is minimized. Therefore, it is desired that the audio output through the DAC and MIDI commands operate at as fast of speeds as possible.

It is also required that the largest time difference between two coordinated outputs, such as a sound signal and an LED flash, is no larger than 37 milliseconds. This value is one half of the duration of a 16th note at a tempo of 200 BPM [26], which is the fastest note that could be played by the device. This latency should be small enough to barely be perceived by the human eye if at all. It is desired, however, that the maximum latency is under 13 milliseconds. The 37 millisecond latency would be mostly imperceivable, but the threshold of latency detection for humans is roughly 13 milliseconds [27] so it is a desirable latency.

3.1.5 Enclosure Requirements

It is required that the enclosure for this project should have a flat area larger than 8.5x11" so that it may fit a full page of sheet music on it. It must be large enough to also house an LCD screen, LED light strips, two potentiometers, a button, and a speaker. It must also be portable. The Department of Defense definition for man-portable sets an upper limit of 31 pounds to be considered portable [27]. The enclosure had a final weight of 5.7 pounds and met this requirement.

It is desired that the camera is not in the way of a user's direct eyesight so that a user may easily read along with the sheet music without adjusting their field of view. The device should also be lightweight and is desired to be less than 10 pounds to be comparable to other music stands.

3.2 Software

3.2.1 Image Processing

It is required that the software program has some sort of image processing to allow manipulation of an image to be used as a parameter for various computer vision and feature extracting libraries. An image must initially be screenshotted from the website, Noteflight.com, and saved as a .png file. If the sheet is longer than three lines, the page will be screenshotted in sections, and then concatenated together to represent a full sheet of music, the equivalent of a 8.5" by 11" piece of paper. The .png is then read in through the OpenCV library and converted to grayscale in order to be processed by the computer vision algorithm.

3.2.2 Computer Vision

It is required that the software program uses some type of computer vision in order to classify the different musical variables. Features must be extracted from a labeled set of templates. These templates consist of various notes, rests, dynamics, time signatures, tempos, and key signatures, screenshotted from the Noteflight website. The algorithm is able to classify the different types of notes, rests, dynamics, time signatures, key signatures, tempos, and a treble clef. Once this has been completed, the resulting matches are put through a clustering algorithm in order to sort the matches to resemble how sheet music is played. The output of this clustering algorithm is then encoded into integers to output to the MIDI controller.

4. Design Constraints: Standards and Impacts

The following section outlines the design constraints that were taken into consideration during the start of the project along with the constraints discovered while creating the preliminary design concepts for each module of the Sheet Music Trainer.

4.1 Design Constraints

4.1.1 Time

Time is the largest constraint in terms of design for the project. An ideal Music Notation to Audio Sheet Music trainer would be able to accommodate more aspects of sheet music. However, due to the amount of time to implement the training needed, it is not possible to account for advance markers or different clefs. Additionally, if implemented in one semester, the accuracy of those readings would be low and below the requirements for the system. Lastly, with more time, more work could be put into fine tuning the outputs of the components of the enclosure to look more like a sellable device.

4.1.2 Budget

The budget set from the department for the project is $200. This constraint will affect the individual modules as well as the system as a whole. Overall the estimated budget for the project ranges around $170, thus allowing more room for additional last minute cost if designs need altered. However, the budget proves as a constraint when considering the Music Notation to Audio Sheet Music Trainer as a sellable device. The recommended budget for a product is a low production cost to yield a higher profit. This creates design constraints for a reliable design due to needing to restrict the budget while using the best components available.

4.1.3 Manpower

Manpower is a significant design constraint when it comes to the main focus of the project, which is image processing and computer vision. For this project, we were only allowed three members: two electrical engineers and one computer engineer. Manpower for the software module would be difficult because there is a lot of work that needs to be done on the main computer vision algorithm, but there is not another computer engineer to split up the software requirements and testing. If there was another computer engineer, the work can be split up into image processing/training and image classification/output.

4.1.4 Musical Symbols

Sheet music and music notation in general has a large amount of variability from piece to piece. Given that this project is using a microcontroller to process and understand sheet music, it will not have the processing power to be able to recognize all edge cases and more rare symbols within music. This device will be able to recognize whole, half, quarter, eighth, and sixteenth notes as well as associated rests of those durations. It will also be able to recognize four dynamics (how loud or soft the music is played) Pianissimo, Piano, Forte, Fortissimo.

4.1.5 Notes Range

For the note range, we will be limiting the range of notes to just the treble clef, starting from middle C and working our way up to high C, which is an octave higher. In total, there should be 15 non-accidental notes (no sharps or flats) that we will be working with. More notes will ensue depending on the key signature.

4.1.6 Tempo Range

For the tempo range, we will be using a low of 40 BPM (beats per minute) and a high of 200 BPM. Most songs will have a tempo of about 120 BPM.

4.1.7 Time Signature and Accidentals

For key signatures, we will be limiting the algorithm to recognize the most common key signatures which include 2/4 (two quarter-note beats per measure), ¾ (three quarter-note beats per measure), and the most common, 4/4 (four quarter-note beats per measure). We will also limit the algorithm to recognize sharps and flats based off of the key signature. Double sharps and double flats are not too common in most popular songs, so this should not be a large restriction on the device's usability.

4.1.8 Communication Speed

The communication speed between the Raspberry Pi and the MIDI controller will be constrained by the slower clock of the microcontroller in the MIDI controller, which will be 84 MHz for the Arduino Due.

4.1.9 Image Consistency

All images and templates created will be screenshotted using the Gyazo tool in order to keep consistency within the computer vision algorithm.

4.1.10 Page Design

For page design, we will limit what pieces of music get scanned to ones that we create ourselves. That way, we can keep the font of the tempo, the spacing between the sections of staff lines, and the margins relatively the same across all sheets.

4.1.11 RoHS Compliant

RoHS (Restriction of Hazardous Substances) is a standard that is responsible for regulating what products are made of, in order to prevent hazardous materials from being used in products that are bought by consumers. We will be limited to purchasing only RoHS compliant products, in order to meet RoHS's standard and to protect the health and safety of our consumers.

4.2 Impacts in Non-Technical Contexts

The Music Notation to Audio Sheet Music Trainer may not have large impacts on the environment or the economy; however it can be argued that it improves the general health, welfare, and culture of the user. Additionally, the device helps equal the playing field for any musician with any level of experience.

There are thousands of studies stating the positive impacts of learning music and playing an instrument has on the human brain. Learning music can help increase pattern recognition, increase emotional development, and increase spatial intelligence [28]. These benefits can only occur if the musician is willing to put the time and energy into practicing. Most musicians get discouraged when trying to learn how to read sheet music in the beginning, figure out harder rhythms later on, or when understanding a difficult key signature. With the aid of the Sheet Music Trainer, musicians can decrease the time needed to first analyze a piece along with increasing their motivation to continue practicing. With this new help to elevate frustrations, the user can subconsciously benefit as they practice. Additionally, it is proven that learning music, at any age, can help increase happiness, decrease anxiety levels, and boost confidence [29]. Lastly, music is a way people can express their culture and backgrounds. Learning pieces from different countries is one of the best ways to understand their culture. Through making the learning process easier, this device can assist the learning and understanding process.

All in all, through assisting with teaching sheet music and new pieces, the goal of the Music Notation Sheet Music Trainer is to assist in expanding the impacts of learning and music to all who are willing.

5. Summary of Design

This section outlines the summary of the designs for the hardware, software, and enclosure for the final prototype implementation of Music Notation to Audio Sheet Music Trainer.

5.1 Power and Hardware Design Concepts

5.1.1 Power Supply Design

In this section will be the summary of the power design for the device. The figure below displays an overview of the power system. The green boxes indicate the voltages the power supply provides to the components within the system.

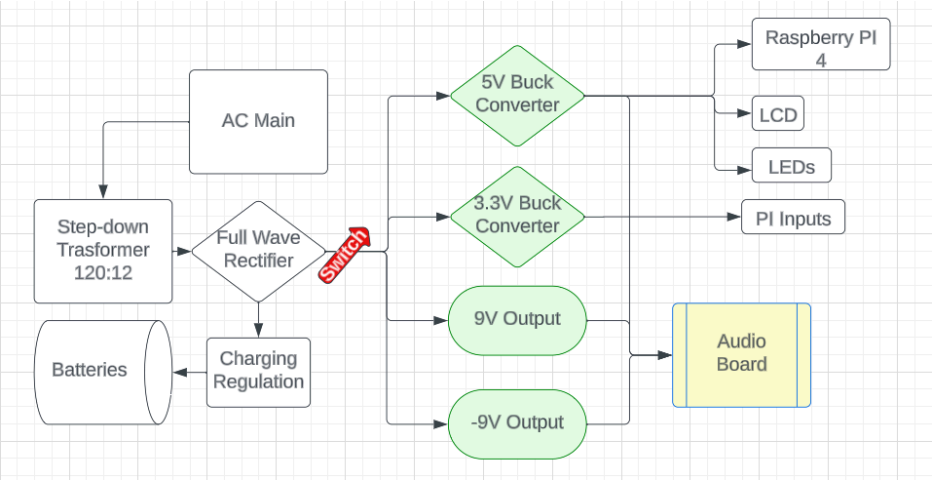


Figure 5: Overall Power Supply Design

5.1.1.1 Full Wave Rectifier

The power supply for the whole system is powered off of a plug that inserts into any standard outlet and a rechargeable battery that supplies the system when there is no power being supplied from the 120 VAC input. To walk through the system, the first step in the design is to utilize a center tapped transformer to step down the 120 V to 20 V. Then the signal will go through a full wave rectifier to transform from AC to DC. Equation 4 was utilized to select the proper step down transformer to ensure the DC output would be equal to 9 volts and -9 volts after the rectifier. Figure 5 displays the schematic for the design mentioned previously. Additionally, a fuse will be added to the design for protection and a smoothing capacitor for the DC output.

$$V_{DC} = 0.9V_{RMS}$$

Equation 4: Calculating V_{DC} Output from a Full Wave Rectifier [30]

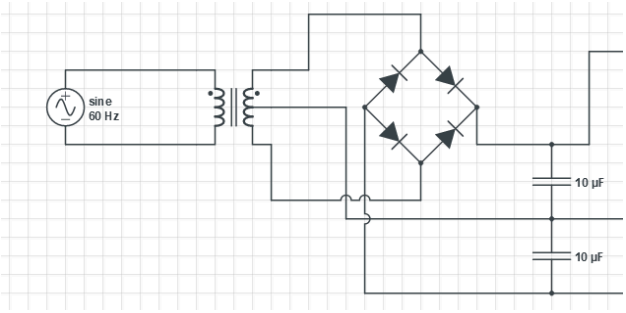


Figure 6: Full Wave Rectifier

5.1.1.2 Battery Charging Circuit

The battery charging circuit schematic is displayed in the figure below. To project the batteries while in charging mode and while trickle charging, there is a 1 kilo-Ohm resistor. The diodes within the circuit act as protection from back voltage from the device itself and allows the batteries to discharge when needed. Additionally, two DPST switches will be implemented to allow the batteries to disconnect from the center ground and negative rail while in battery charging mode. The reason behind this design is to increase the charging current. Section 6.2.4 displays the charging circuit results before and after adding the switches.

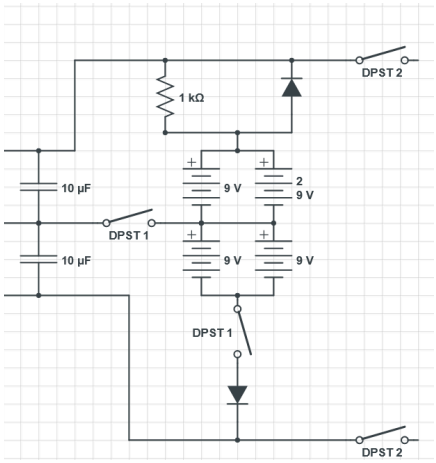


Figure 7: Battery Charging Schematic

5.1.1.3 Voltage Output Designs

To generate the needed 9 volts and -9 volts rail a centerate tapped transformer was utilized as described in section 5.1.1.1 and created as well with a centered tapped battery system as described in section 5.1.1.2. The 5 volts and 3.30 volts outputs were generated through the use of buck converters to decrease power loss.

5.1.2 Outputs

5.1.2.1 LED Lights

Figure 8: Individually Addressable LED Strip Lights [34]

A strip of light emitting diodes (LED) will be used as an output for the Raspberry Pi microprocessor. The strips are individually addressable which will allow the device to produce a light that can follow the desired or set tempo of the piece. The LED above the measure the device is on will be lit during the whole duration of the measure and the next will be lit as soon as the device reaches the following measure. The mathematical code to calculate the speed of the traveling of the lights across the page and down the page will be done on the Pi after the sheet music has been processed. The strips will be powered through the supply with 5VDC and connected to ground. Additionally, there will be a connector that allows the three signal wires plug to be inserted into the PCB and connect to the Pi.

5.1.2.2 LCD Screen

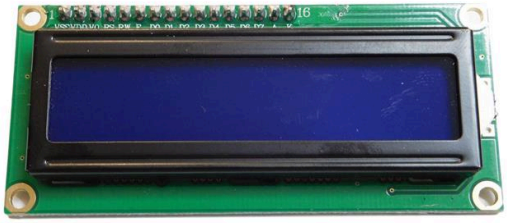


Figure 9: LCD 1602 2x16 Blue-White [35]

A liquid crystal display screen will be utilized to display messages to the user along with the note being processed. The display code will be written in the Raspberry Pi microprocessor. For the LCD screen the LiquidCrystal library is available and will be used to write to the screen more effectively and efficiently [36]. The LCD will be powered through the supply with 5VDC and connected to ground. Additionally, the signal wire will be connected directly to the pins of the Pi.

5.1.3 Inputs

5.1.3.1 Playback Speed and Instrument Selectors with Analog to Digital Converter

Figure 10: 10K Ω Potentiometer with Cap [37]

A 10 kilo ohm potentiometer will be used to allow the user to pick the tempo reduction of the piece along with the instrument. The Raspberry Pi can determine the selection based off of the current reading back into the controller. The Raspberry Pi does not have any analog input pins, so the ADS1115 4 channel ADC [38] will be used to read these analog input values.

5.1.4 Overall Enclosure

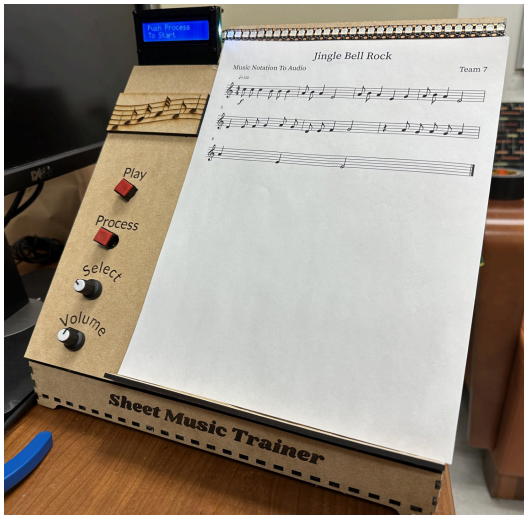


Figure 11: Photo of the Final Designed Enclosure

The enclosure is to represent a music stand that a beginner can place anywhere. Hence the triangular shape of the design. The left hand side of the design is the selection potentiometers and scan button. To the right of the user selections is the stand that will hold the paper along with the LED strips. On the top of the enclosure is the LCD screen. On the side of the enclosure will be the speaker. Finally, the charging cord for the device, along with the opening to access the batteries, will be on the back of the device.

The enclosure will consist of laser cut and 3D printed components. The base of the enclosure will be laser cut from medium thick wood for stability and ability to produce clean engraved designs. The base will have holes cut for the two potentiometers, scan button, speaker, power switch, power cable, and for any necessary mounting. Additionally, the base will be engraved above buttons, switches, and potentiometers to indicate their purpose for the user. Lastly, the LCD for the system will be held by a 3D printed stand that will be attached to the base of the enclosure by screws.

5.2 Control and Audio Design Concepts

For the control and audio of The Music Notation to Audio Sheet Music Trainer, the design will operate according to the following flow structure. See the subsections below to see all control and audio design concepts.

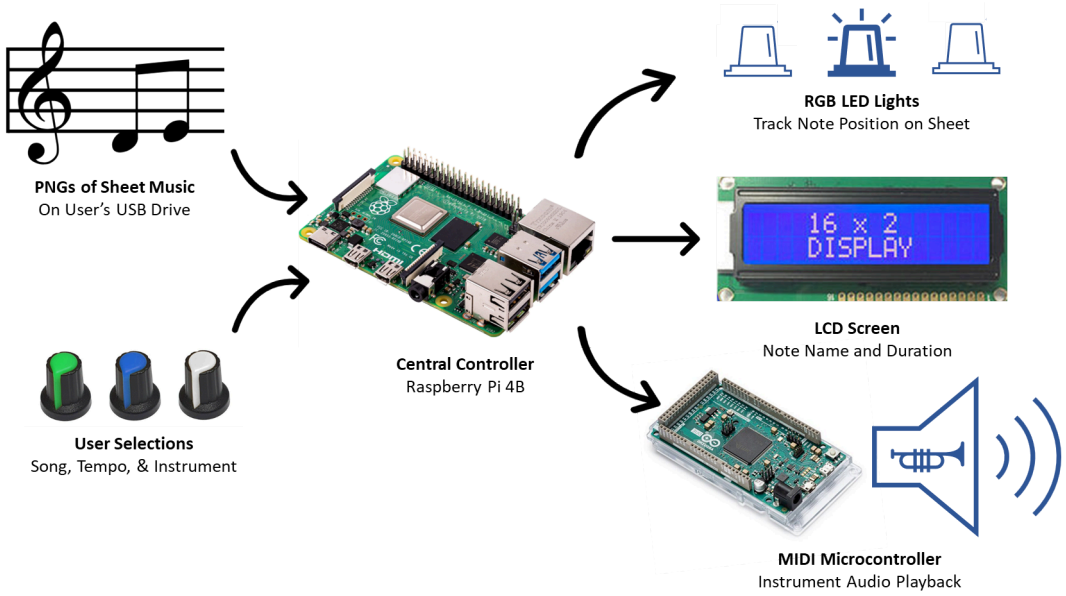


Figure 12: System Flow Diagram

5.2.1 Central Controller

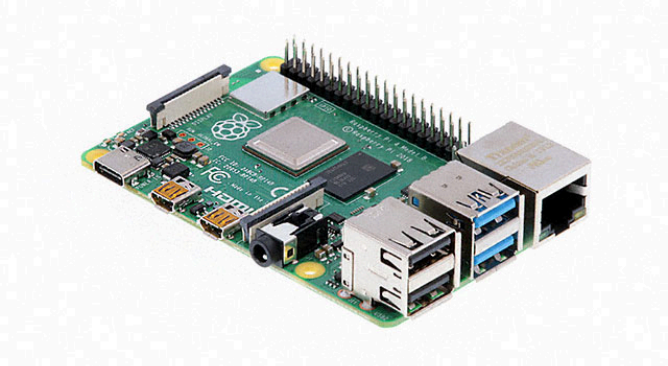


Figure 13: Raspberry Pi 4B [39]

The central controller used in this project is the Raspberry Pi 4B. It has a 1.8GHz quad-core ARM Cortex-A72 CPU and 2 GB of SDRAM. This processing power was sufficient for the computer vision and control portions of this project. Originally a Raspberry Pi Zero 2 W was the first controller tested for this project, but due to sporadic connectivity issues on the Pi Zero inhibiting the prototyping process, a Pi 4B owned by one of the group members was used instead. This was not only an improvement in processing power, but it also allowed for savings in the team's budget. A new Pi did not have to be purchased. The Pi 4B also has onboard USB-A ports which allows for ease of integration of the USB drive input.

The central controller handles all processing, inputs, and outputs. All computer vision is done onboard on the Pi's processor. The Pi accepts GPIO inputs from the ADC about user selections and button pushes, and it times and executes all audio and visual playback outputs such that the device keeps the appropriate musical timing. The ADC input and LCD output will be communicated with through I2C, and the MIDI microcontroller will be sent commands using serial communication. This serial will operate at a speed of 38,400 bits per second to exceed the

speed minimum of 31,250 as described in the System Requirements section. See the table below for the pins used on the Pi and the device connected to each pin.

Raspberry Pi Pin Number	Pin Name	Connection
2	5V	5V Buck Converter Output
3	SDA	SDA ADC, SDA LCD
5	SCL	SCL ADC, SCL LCD
8	TX	MIDI Microcontroller RX
9	Ground	Ground
10	RX	MIDI Microcontroller TX
18	GPIO 24	Process Button
22	GPIO 25	Play Button
32	GPIO 12	LED Light Data Line

Table 1: Raspberry Pi Pinout

5.2.2 Central Controller General Code Structure

Below is a flow chart of the Raspberry Pi Python code run by the device.

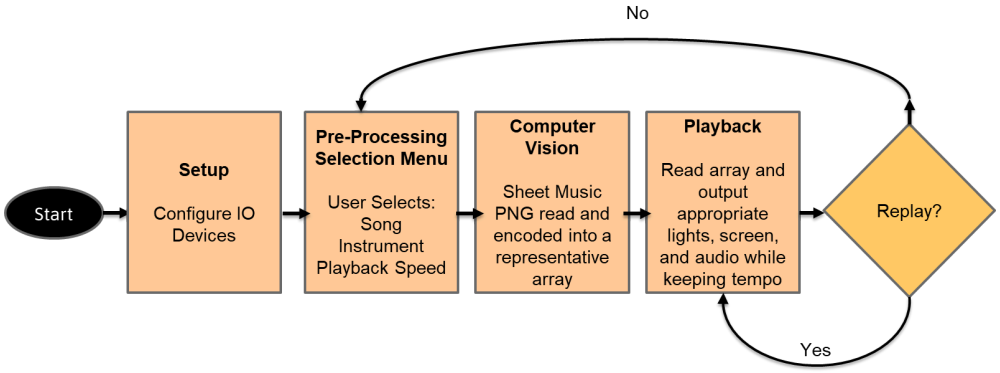


Figure 14: Raspberry Pi Central Control Code

First, the user is prompted to a menu that appears on the LCD screen. Using the selection knob and process button, they select the image file for the song that they are playing from the USB drive. Then, they make their selections for instrument and playback speed.

Figure 15: Example of Instrument Selection on LCD Menu

There are 8 instrument options and 4 playback speed options for a user. Each menu selection screen also allows the user to go back to the start of the menu. These options allow a user to cater the sound output to what they are practicing and to slow down the output when they are learning a new piece. The options are listed below. Note that the instrument sounds offered on this device are common instruments seen in band, chorus, and orchestra groups. These groups comprise a large portion of the people that actively use sheet music.

Instrument Options	Return to Menu Sinusoidal Tones Alto Saxophone Clarinet Piano Tenor Saxophone Trumpet Violin Voice
Playback Speed Options	Return to Menu Original Tempo ¾ Tempo ½ Tempo ¼ Tempo

Table 2: Instrument and Playback Speed Options

Following all selections, the user is shown a screen that says “Processing…” while the Computer Vision algorithm runs, dissecting the sheet music and creating an encoded array representing all information about key signature, tempo, dynamics, what notes are present, and the order in which they appear. When the processing is complete, the user is prompted to press play.

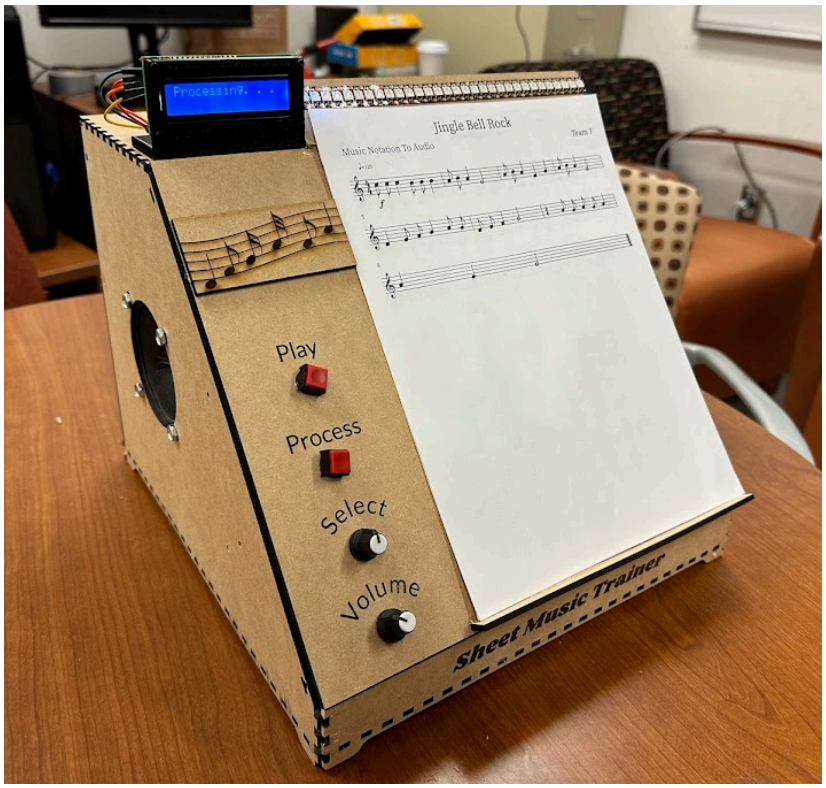


Figure 16: Processing Screen

Figure 17: Ready to Play User Prompt

Once the user pushes the play button, a three second countdown is displayed on the screen, and playback begins. During playback, the encoded array produced by the computer vision algorithm is looped through. Output timing is determined by tempo and playback speed, and for each output cycle, an LED above the note is blinked, the screen displays the note name and duration, and a pseudo MIDI command is sent to the MIDI microcontroller to be turned into an instrument sound at a specified frequency and volume. Note that the computer vision encoded array is designed to allow for fast, mathematical navigation of Lookup Tables for note frequency, name, and duration. Lookup Tables are adjusted for each line of sheet music based on key signature.

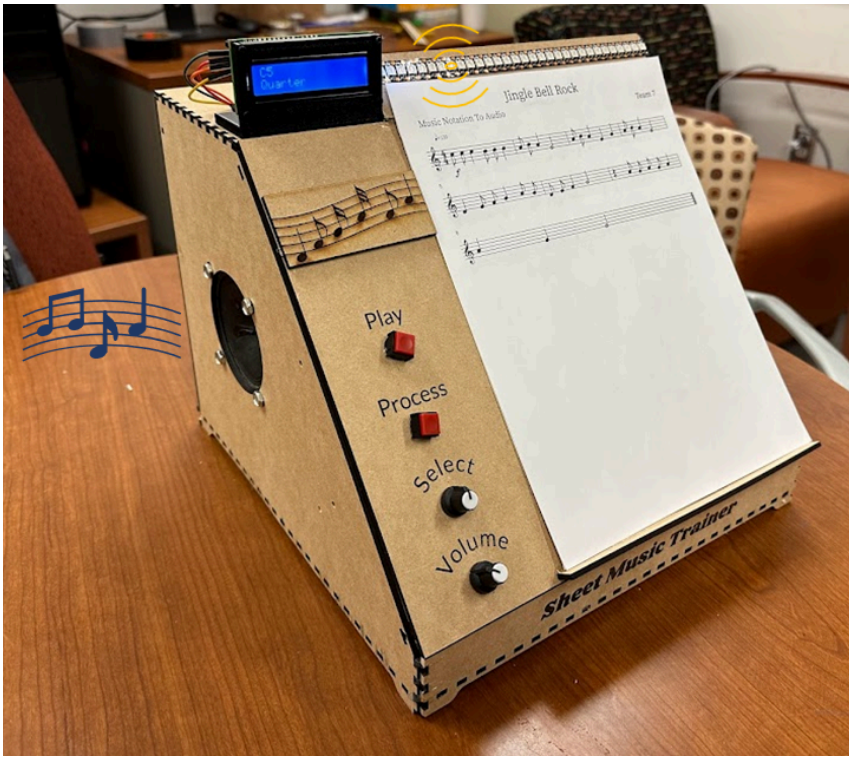


Figure 18: Note Name and Duration Output to LCD Screen

After playback is finished, the user has the option to press the play button to replay or to press the process button to return to the menu and start over. It is a fairly simple process and interface for the user to interact with.

5.2.3 Pseudo MIDI Code

Electronic Music produced by traditional MIDI is communicated using a three byte structure. The first byte holds information about what instrument is being played and whether a note is being turned on or off. The second byte holds information about the frequency of the note being played. The third byte holds information about the volume at which the sound is being played. This device communicates a modified MIDI code serially, following that same structure. It is built for this musical device that handles 8 instruments and 4 dynamics that indicate musical volume.

Byte	Information	Range of Values
Status Byte	Note Off/ Note On with Specific Instrument Sound	0x38 - Note Off 0x39 through 0x40 - Note On With 0x39 as Instrument 1 and 0x40 as Instrument 8
Frequency Byte	Musical Note which corresponds to a specific frequency	0x15 (21) through 0x6C (108) Represents all notes that can be played on a piano. See figure below.
Volume Byte	Volume Level based on Dynamic	0x30 - Fortissimo (Very Loud) 0x31 - Forte (Loud) 0x32 - Piano (Soft) 0x33 - Pianissimo (Very Soft)

Table 3: Pseudo MIDI Code Structure

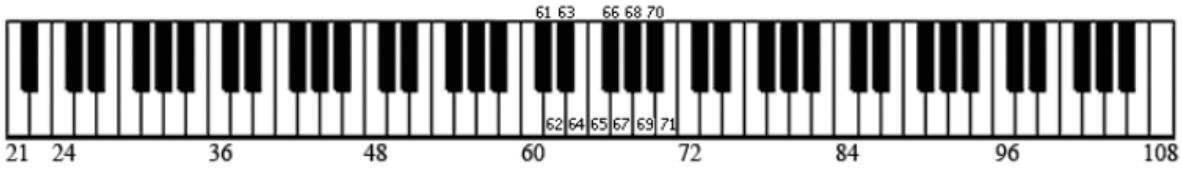


Figure 19: MIDI Frequency Byte Numbers in Decimal Shown on a Piano [41]

5.2.4 MIDI Synthesizer Microcontroller

This project requires a MIDI synthesizer to play the audio commands. This synthesizer is an Arduino Due microcontroller.

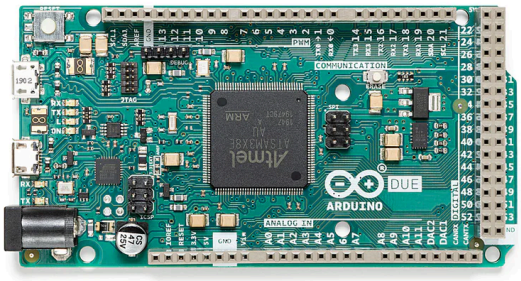


Figure 20: Arduino Due Development Board [42]

This microcontroller has already been used by the team in a previous project and one was already owned by the team. This development board runs on 3.3V communication like the Raspberry Pi, so there would be no need for logic conversion and extra hardware in the serial communication wiring lines. The Arduino Due also has a built-in 12-bit DAC. This fact made this microcontroller more appealing than other boards like the Arduino Uno. The Uno would have required an external DAC module, and the extra communication steps would significantly slow down the output speed. This DAC is limited to 0.55V-2.75V output only, but because the analog output goes through filtering and amplification before reaching the speaker, this is

acceptable. The output signal can be shifted and amplified as needed for optimal audio output. The Arduino Due also has an 84 MHz clock which is much faster than the 16 MHz clock on the Uno and other microcontrollers.

The Due accepts serial communication from the central controller that transmits a modified version of MIDI commands. Serial communication was used because it enabled testing of the module through the Serial Monitor Interface on the Arduino IDE for PC. The module could therefore be tested and developed disconnected from the rest of the circuit by sending commands from a computer. The MIDI microcontroller hardware is configured according to the following flow chart.

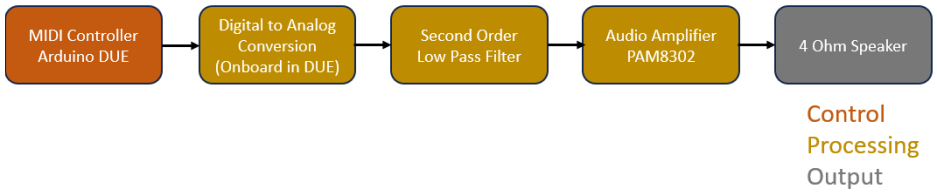


Figure 21: MIDI Microcontroller Structure

5.2.5 Instrument Sound Construction and Playback

A key component of this project is creating sound digitally. This device has 8 different instrument options: sinusoidal tones, alto saxophone, clarinet, piano, tenor saxophone, trumpet, violin and voice. This is done using Fourier Reconstruction. The sound is then played back using Direct Digital Synthesis. This presents a way to recreate instrument sounds in a unique way and without the use of an externally downloaded library.

5.2.5.1 Fourier Reconstruction and Audio Envelopes

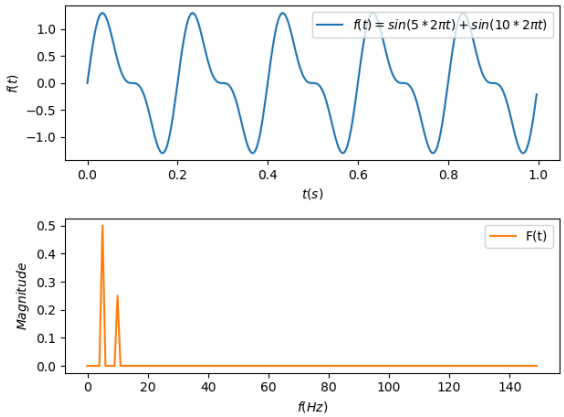


Figure 22: Simple Fast Fourier Transform Example [43]

Every instrument has its own associated fourier spectrum based on its physical composition [22]. No matter what note is played, the fourier spectrum will be the same. All that changes is the fundamental frequency.

Thus, to build the instrument sounds, live samples for one full octave from A3 at 220 Hz to A4 at 440 Hz were analyzed for all instruments. These instrument recordings were provided by Dr Steve Jacobs. Using MATLAB, the FFT was taken of all samples, and an average

spectrum was found for each instrument. From this spectrum, the first 20 harmonics were extracted in magnitude and phase.

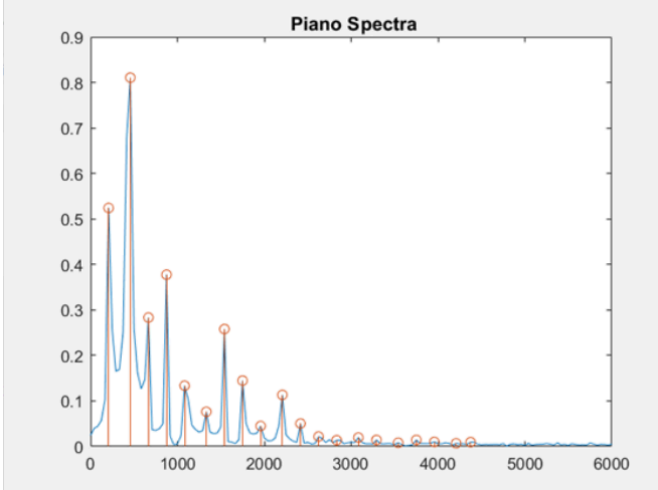


Figure 23: Sampled Piano Spectrum

These harmonic coefficients were then normalized to 1 and input into the Arduino. At setup, one period of each instrument's waveform volume levels was defined as a 1000 element array for each of the 4 discrete volume levels. The 1000 elements is to ensure that the waveform has a high enough resolution to capture all harmonics included in the output waveform. This array was built as the sum of all 20 harmonic sine waves for the instruments over the one period. This overall waveform was then rounded, scaled, and shifted to fit within the 0 to 4095 integer range of the Arduino Due DAC. Note that each nth harmonic could be defined by the following equation for H_n where A_n is the spectral component at the nth harmonic and p_n is the spectral phase offset on the sine wave for harmonic n,.

$$H_n = A_n sin(nt + p_n)$$

Equation 5: Harmonic Component of a Signal

However, a musical instrument sound is not just defined by its frequencies, but also by the volume envelope that surrounds the audio that is based on the way they are played. Pianos and guitars for example have a relatively exponentially decaying audio envelope because their notes are played with a sharp pluck. Horn and woodwind instruments have a gradually increasing and steadying envelope because of the way that musicians gradually expel air into these instruments. To determine these audio envelopes for the instruments studied for this project, their MATLAB samples were analyzed. All relative maxima were extracted to form a rough envelope, and an estimated general envelope was fit to these functions.

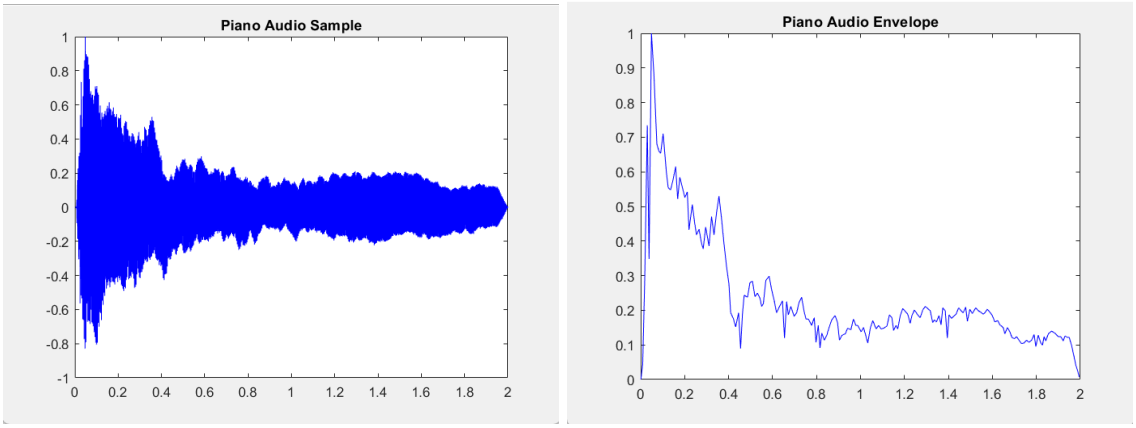


Figure 24: Piano Audio Sample and Audio Envelope

Below are the estimated audio envelopes for each instrument. These envelopes were defined for each function for 5 seconds at 200 samples per second.

Instrument(s)	Envelope Function	Description
Sinusoidal Tones	N/A	N/A
Alto Sax, Clarinet, Tenor Sax,Trumpet	$1-e^{-1.25t}$	Gradual Increase of Volume
Piano	$0.7e^{-2t} + 0.3e^{-0.1t}$	Sharp Strike followed by Soft, Slight Decay
Violin	$0.85e^{-1.25t} + 0.1\sin(2\pi t)$	Gradual Increase of Sound with Slow Pseudo Vibrato
Voice	$0.85e^{-1.25t} + 0.1\sin(5\pi t)$	Gradual Increase of Sound with Somewhat Slow Pseudo Vibrato

Table 4: Audio Envelope Functions for All Instruments

5.2.5.2 Direct Digital Synthesis on Arduino

The audio in this MIDI microcontroller is produced using Direct Digital Synthesis. This is a common approach to generating sine wave outputs from digital systems described in [JC3]. For this project, it is applied to each sum of sinusoids for each instrument sound. This method works by taking one period of a signal and traversing it with a constant playback sampling rate but a varying step size. Increment step size is directly proportional to frequency, so higher frequency sounds are played back faster with a larger step size. When the increment causes the index to become larger than the size of the array, the index wraps to the front of the array as if the output was sampling an infinite number of periods of the waveform. All increments corresponding to all MIDI frequency values are calculated and stored in a Lookup Table at the Due's initialization. All instrument arrays have the same length so that the increment value is the same for one frequency across all instruments. Note that instead of using fractional increment

values, all increments and indices are bit shifted left by 18 bits and fixed point math is done. When a value is needed from the sample arrays, this index value is bit shifted right by 18 to get the actual index of the output value in the array. This fixed point math instead of floating point math, combined with using Fourier Reconstructed waveforms populated at startup, keeps the interrupt and sound output operating fast. With audio, timing is everything, so all microcontroller operations must be fast.

The sound output is played at a constant playback sampling frequency of 48 kHz. This 48 kHz is established by using Timer Interrupts on the Due. The sampling rate of 48 kHz was selected because it is a very common value for modern audio recording [74].

5.2.5.3 Arduino Due Code

Below is a flow chart of the Arduino Due code.

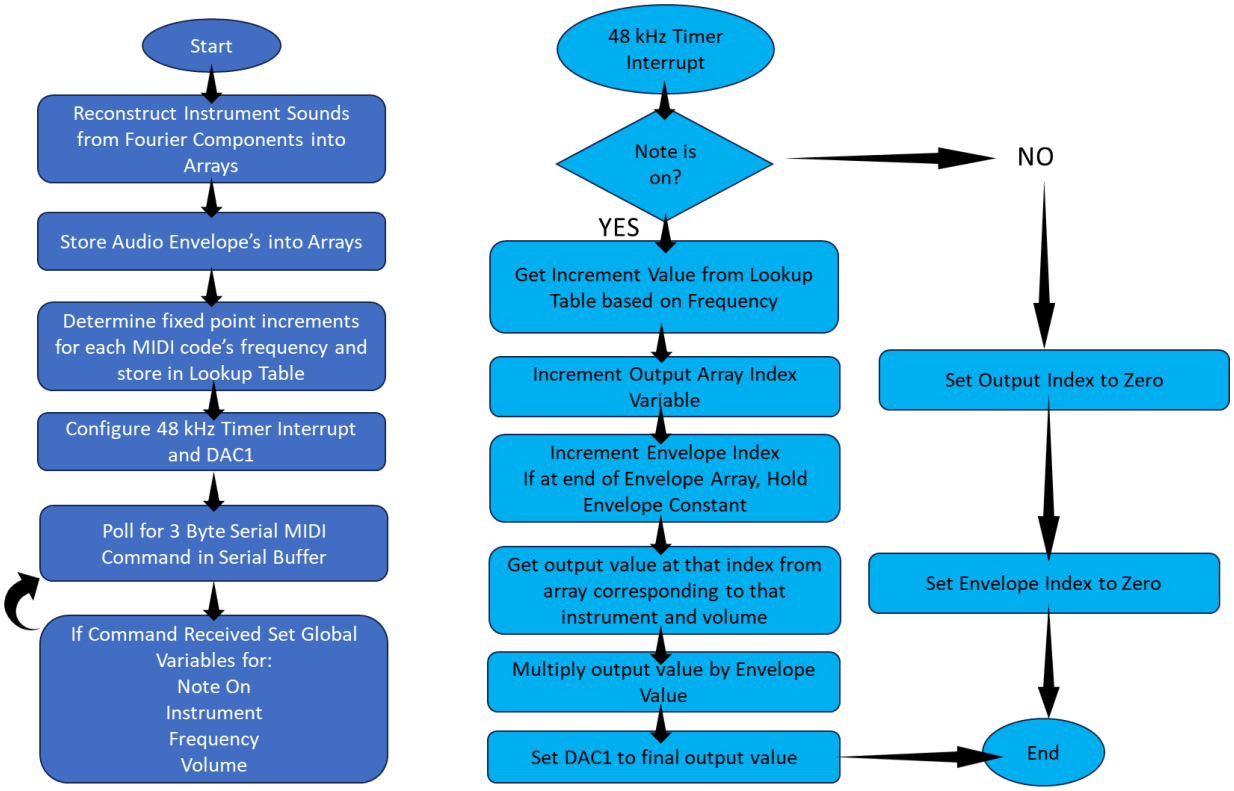


Figure 25: Arduino Code Flow Chart

5.2.6 Audio Filter and Amplification

The sections below detail all audio amplification and filtering circuitry designed and implemented in this project.

5.2.6.1 Audio Filter Design

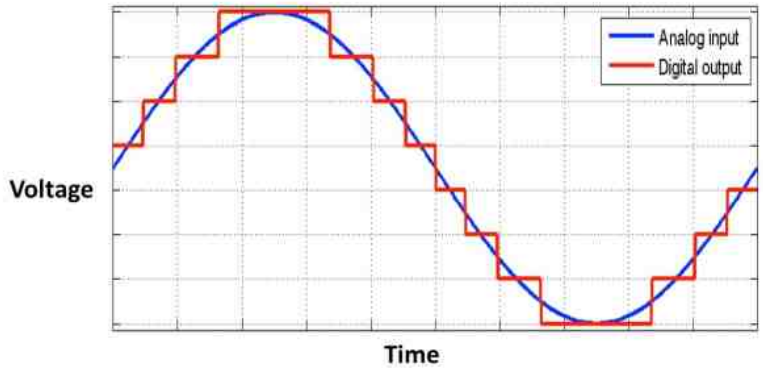


Figure 26: DAC Discrete Output [44]

When doing digital to analog conversion, the output is discrete steps like shown above. These high frequency changes as well as any frequency noise should be filtered out of the output signal to create a smoother, cleaner waveform. Because the audio output occurs at a constant frequency of 48 kHz, the Nyquist Criterion [71] requires an antialiasing filter with a cutoff frequency of at maximum 24 kHz for testing. To ensure that frequencies at and above 24 kHz were sufficiently suppressed and to suppress some other unwanted high frequency noise, a low pass filter was designed for a cutoff frequency of around 18 kHz. The filter is a second order Butterworth low pass filter as pictured below. Note that the op amp is a TL074, 4 channel JFET amplifier.

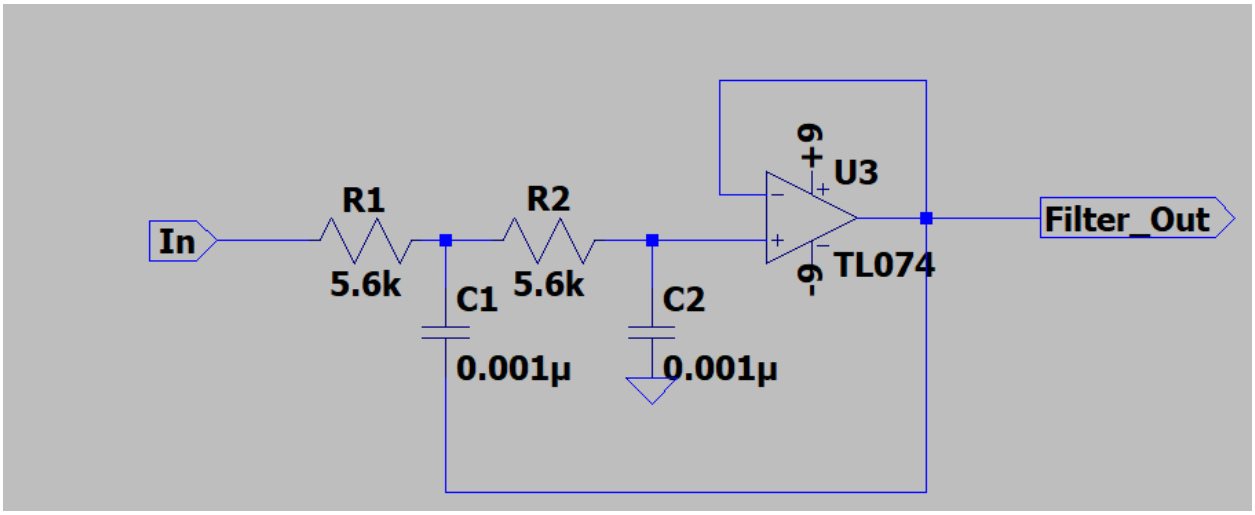


Figure 27: Second Order Butterworth Low Pass Filter

5.2.6.2 Audio Amplification

In audio circuitry, impedance matching, matching the impedance of the amplifier to the impedance of the speaker, is vital. If the output impedance of the amplifier is much higher than that of the speaker, the audio signal will become highly distorted and sound drastically worse. This is why the TL074 JFET op amp [72] used for the other audio circuitry could not be used. It has a 120 ohm output impedance, which is too high and leads to significant output waveform distortion.

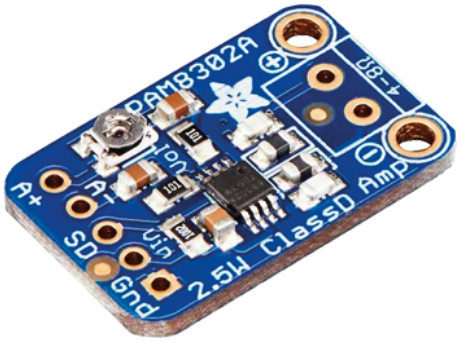


Figure 28: PAM8302A 2.5W Amplifier Module [47]

To avoid this issue, the PAM8302, 5V, 2.5W low-voltage mono amplifier [47] was used to drive the 4 ohm speaker on the device. It is a low output impedance amplifier meant for this application. This amplifier module has an input range of 0V to 5V so the following amplifier stage was designed to get the 0V to 3.3V Arduino Due analog output waveform to a 0V to 5V waveform. This would allow for maximum output signal amplitude. See the preamplifier circuit below. Note that the 10k Ohm potentiometer allows for volume control of the output. This potentiometer is placed on the surface of the device and is wired to the final PCB.

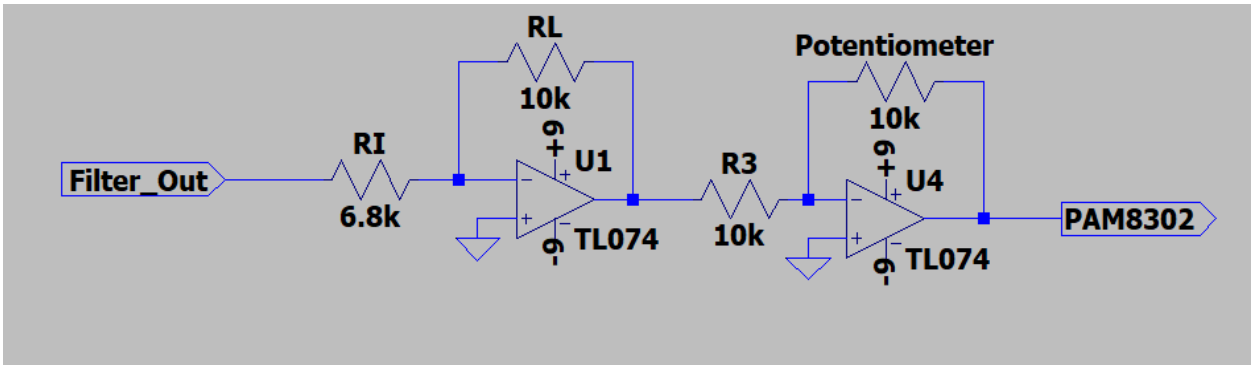


Figure 29: Preamplifier Circuit

5.2.6.3 Filter and Audio PCB Implementation

Below is the final PCB for the filter and preamplifier circuits. Note that there are female headers for all power and audio inputs and outputs, as well as for the potentiometer wiring (which in this picture is a static 10K ohm resistor), placed at the edges of the board for ease of access and landing wires. All signal inputs are on one side. The speaker output and potentiometer are on the other side. Note that silkscreen values don't match with the actual soldered component values. This is because the circuit has always kept the same structure throughout the design process but its component values have been adjusted multiple times. Also note the standoffs on the corners to prevent shorting and allow the board to be mounted on the final device.

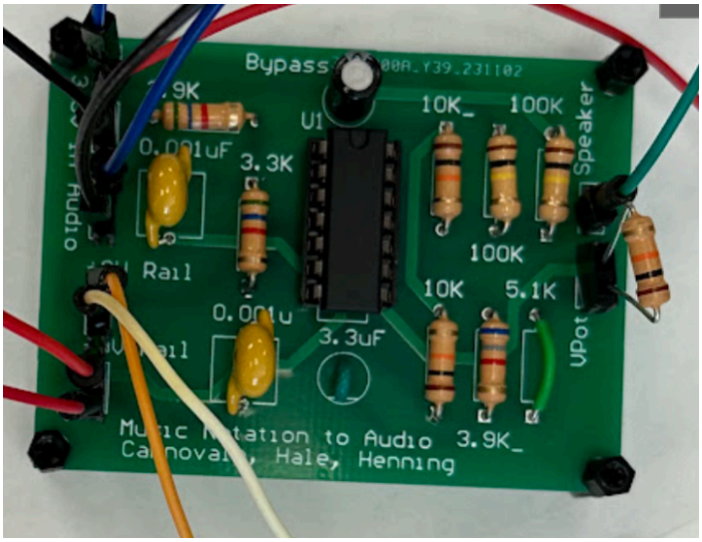


Figure 30: Filter and Audio PCB

5.3 Computer Vision Design Concepts

For the computer vision aspect of this project, template matching and clustering were the main driving force in classifying the notes and reordering the matches to resemble a sheet of music. Below is the table used to encode the different musical templates. Each template is represented by a hex value, later to be converted into an integer for communication purposes.

Symbol	**Value (hex)**	**Representing**
Treble Clef	0xFF	Beginning of a new line
Key Signature	0x00…0x0E	15 different key signatures featuring sharps and flats
Time Signature	0x24, 0x34, 0x34	2/4, 3/4, and 4/4 time
Tempo	0x28…0xC8	40 to 200 tempo in increments of 10
Notes & Rests	0x(0-E)(0-9)	1st 4 bits represent letter of the note, 2nd 4 bits represent type of note
Dynamics	0x100, 0x200, 0x300, 0x400	4 different dynamics (ff, f, p, pp)

Table 5: Software Output Encoding

5.3.1 Image Processing - Formatting the Templates and Music Sheets

The first step for the software side of the project is the preprocessing step. A list of templates were created in order to perform template matching on the music sheets. These templates consist of screenshots of different musical notes, rests, key signatures, time signatures, tempos, a treble clef, and dynamics, provided by Noteflight.com. Below is an example of what these templates look like:

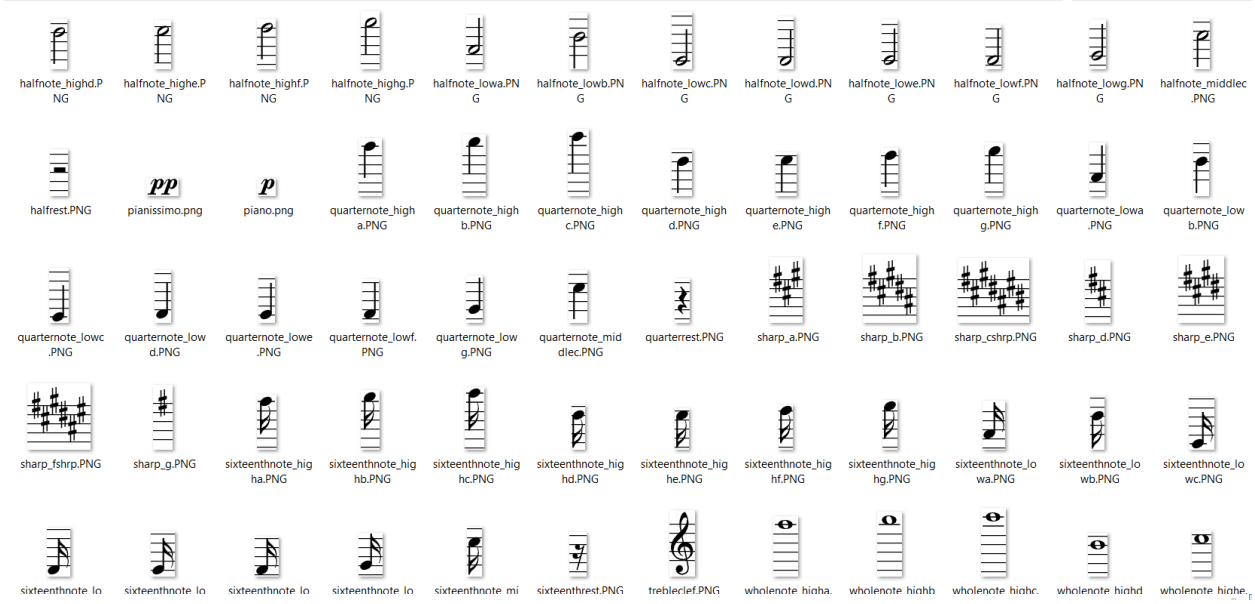


Figure 31: Example Templates for Template Matching

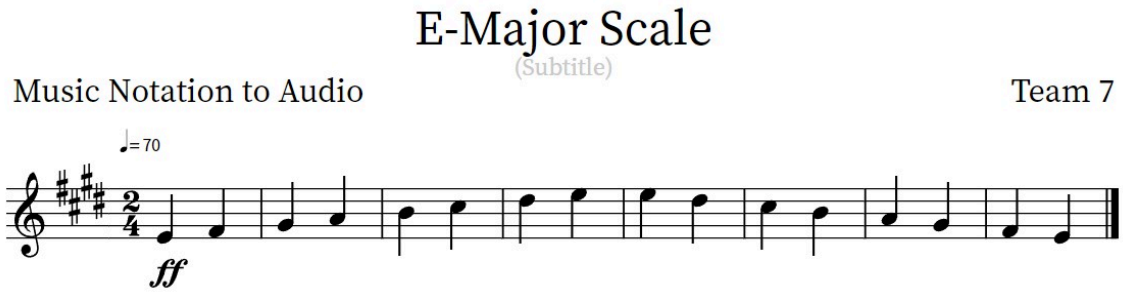


Figure 32: Example Sheet Music

Along with these images, sheet music had to be created and screenshotted the same way as these templates in order to maintain consistency. These sheets were created and screenshotted using the Noteflight website. Screenshots were either processed “as-is” or concatenated together to represent a full sheet of music. The music sheets were then converted to grayscale as a last-step prep for the computer vision algorithm.

5.3.2 Template Matching

For the computer vision aspect of the project, I used a multi-template matching library with various parameters in order to perfect the matching between my templates and the sheet music [66]. This library was used to be able to identify multiple instances of the same template appearing in the sheet, while also being able to distinguish between the unique types of templates. There are six-different parameters that went into the multi-template-matcher:

```
(listTemplate,
 sheet_img,
 score_threshold=0.925,
 searchBox=(0, 0, sheet_img.shape[1], sheet_img.shape[0]),
 method=cv.TM_CCOEFF_NORMED,
 maxOverlap=0.3)
```

Figure 33: Template Matcher Parameters

The first parameter represents all the templates that I created, stored in a folder called "templates". The second parameter represents the sheet of music being processed. The third parameter represents the "score threshold" that the objects on the sheet music have to meet in order to be considered a match. The fourth parameter represents the search box of where the program should be looking to find matches. It begins with the x and y coordinates (0,0) and uses the dimensions of the sheet being processed to define the search area. The fifth parameter represents the type of method the template matcher uses. In this case, it uses TM_CCOEFF_NORMED, or Normalized Cross-Correlation Coefficient. For this type of template matching, the different templates will "slide" over the sheet music to find matches. Light spots represent a positive value and dark spots represent a negative value. The dot product is constantly being computed as the template traverses the sheet image. Light spots over light spots will return a positive value for the dot product. This also occurs for dark spots over dark spots, because the dot product of two negatives returns a positive value. After the template has been compared, it returns a value between 0 and 1, 0 being no correlation and 1 being a perfect match. Most matches fall between 0.96 and 1 due to the last parameter. The sixth and final parameter is the max overlap parameter. This means that no more than 0.3 or 30% overlap between matches on the music sheet. This is to prevent the same object from being identified multiple times. After all of this is done, the template matcher returns a list of "hits" that consist of all of the template images it found from the music sheet. Below is an example:

	TemplateName	BBox	Score
32	trebleclef	(70, 245, 34, 88)	1.000000
33	trebleclef	(70, 404, 34, 88)	1.000000
7	eighthnote_middlec	(424, 261, 19, 67)	0.999978
14	halfnote_lowa	(716, 406, 22, 72)	0.999588
20	quarternote_lowa	(147, 247, 21, 71)	0.999509
1	44_timesignature	(115, 258, 20, 60)	0.999334
21	quarternote_lowa	(494, 247, 21, 71)	0.998892
17	quarternote_highe	(964, 259, 28, 58)	0.998144
11	eighthnote_lowf	(365, 420, 31, 53)	0.997261
8	eighthnote_middlec	(622, 261, 19, 67)	0.995033
22	quarternote_lowa	(296, 247, 21, 71)	0.994293
9	eighthnote_middlec	(540, 420, 19, 67)	0.993431
23	quarternote_lowa	(117, 406, 21, 71)	0.992286
18	quarternote_highe	(815, 259, 28, 58)	0.992165
15	halfnote_lowa	(675, 247, 22, 72)	0.988366
12	eighthnote_lowf	(372, 261, 31, 53)	0.983114
10	eighthnote_middlec	(1097, 261, 19, 67)	0.981144
19	quarternote_highe	(889, 259, 28, 58)	0.980624
24	quarternote_lowa	(222, 247, 21, 71)	0.970947
13	eighthnote_lowf	(570, 261, 31, 53)	0.970938
0	120_tempo	(112, 203, 52, 27)	0.953994
2	eighthnote_highf	(1043, 255, 20, 60)	0.945880

Figure 34: Example Output of Template Matcher

The “TemplateName” column represents all of the templates that were identified from the sheet music. The “BBox” column represents the x and y location of where the match was found, as well as the width and height of the box it found it in. The “Score” column represents the similarity between the template and the object it was compared to on the music sheet. Once these matches are found, the data needs to be sorted to represent an actual piece of music.

5.3.3 Clustering Algorithm

After the program gets all of the matches, the matches need to be reordered. For traditional sheet music, it gets read top to bottom & left to right, like a book. A K-Means clustering algorithm is used to achieve this [67]. K-Means figures out patterns in data based on a provided “K” input, which represents the amount of “clusters” or groups you want present in the data. The number of clusters is based on the number of treble clefs detected. Since there is a treble clef on every line of music, that is how to determine the number of lines in the piece. Once the program groups all the lines together, the groups are then sorted by their mean y-coordinate. Y-coordinates closer to 0 are going to be closer to the top. Once all the clusters are sorted, the individual elements within the clusters need to be sorted. The items within the clusters are then sorted by the x-coordinate, with x-coordinates near 0 being placed closer to the left. Once all the matches are sorted, the program goes through the last step which is post-processing.

5.3.4 Post-Processing

The post-processing included in this design is to handle edge cases and to take care of constraints that we defined. It also is supposed to work on encoding the matches into integers for the MIDI controller to process

5.3.4.1 Removal of Time Signatures & Key Signatures

There are certain instances in music where time signatures and key signatures change. To help the musician, these changes will appear at the end of the previous line. Here is an example:

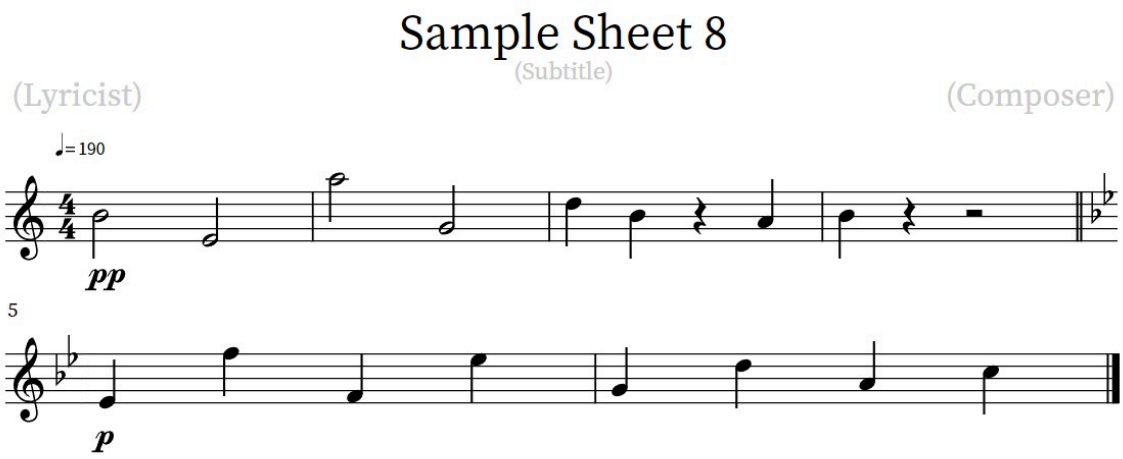


Figure 35: Key Signature Change Example

For the code however, this can mess up the program. The absence of a key signature represents the C-Major scale. The first line of the sample sheet is set in C-Major, but since the program detects a key signature within that cluster, the key signature would be set to B-Flat instead of C-Major. This can also occur with time signatures. Based on our constraints, time-signature and key signature can only change at the beginning of a line. To fix this, logic is in place to check the last two spots of the cluster for either a time signature or a key signature. If they are present, the program will remove them from the dataframe that contains all of the matches.

5.3.4.2 Insertion & "Carry-over" Logic

For the beginning of every line, there should be 4 values that represent the following in order: treble clef, key signature, time signature, and tempo. However, in music, these values do not always appear on every line, so they need to be added in order to keep the information consistent. Also, the program does not know that time signature and tempo carry over throughout the rest of the piece unless changed.

For missing values, or the absence of a value, logic is in place to insert these absent values into the dataframe of all the matches. If no tempo is identified, the default is set to 120 BPM. If no time signature is identified, the default is set to 4/4 time. If no key signature is identified, C-Major scale is set as the key signature. All of these values are placed into the dataframe based on the location of the treble clef within that specific cluster (line). Key signature is 1 to the right of the treble clef, time signature is 2 to the right, and then finally, tempo is 3 to the right. Here's an example of the insertion logic:

Number of initial matches before post-processing: 20

	TemplateName	BBox	Score	Cluster	HexValue
0	trebleclef	(89, 261, 34, 88)	0.999999	0	255
1	cmajor	Default Key Signature	1.000000	0	0
2	44_timesignature	(127, 260, 20, 60)	0.999335	0	68
3	120_tempo	(140, 190, 57, 33)	0.955851	0	120
4	quarternote_lowc	(160, 266, 26, 73)	0.999542	0	2
5	pianissimo	(163, 322, 43, 24)	0.997023	0	1024

Inserted logic

Figure 36: Inserted Value of C-Major

The next part of the logic is the "carry-over" logic. Since tempo and time-signature do not have to appear on every line, the program cannot remember these values on its own. For tempo, if it is identified in the first cluster (first line), it gets set to a carry-over variable that gets inserted into the dataframe at its respective spot at the beginning of every new line. For time-signature, when it is identified by template matcher, it gets set to a carry-over variable. Until the time-signature changes, that carry-over value is inserted into the dataframe at its respective spot at the beginning of every new line. Here's an example of my dataframe outputted to a .txt file with the sheet that was tested:

27	trebleclef	(86, 421, 34, 88)	1.0	1	255
28	cmajor	Default Key Signature	1.000000	1	0
29	44_timesignature	Carried-Over Time Signature	1.000000	1	68
30	80_tempo	Carried-over Tempo	1.000000	1	80
31	quarternote_lowa	(126, 414, 21, 71)	0.992289	1	82
32	quarternote_lowa	(199, 414, 21, 71)	0.999455	1	82
33	halfnote_highe	(271, 419, 22, 65)	0.956724	1	147

Figure 37: Carry-Over Logic

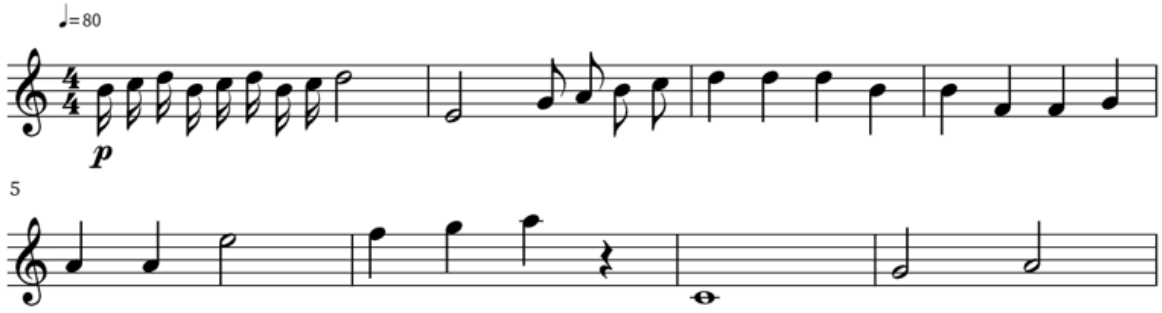


Figure 38: Sample Sheet Tested with Carry-Over Logic

From the above two images, the 80 BPM tempo and 4/4 time signature do not appear on the next line, so they are inserted into the dataframe.

5.3.4.3 Encoding & Calculation of Measures

Once all of the templates have been matched, sorted, and the proper post-processing logic has been performed, the last step is to calculate the number of measures and encode the matches. The program goes through all of the matches and based on the time signature and notes/rests, will count the number of beats per measure. Once the number of beats equals or is greater than the top number of the time signature, the number of measures increases by 1 and resets the

number of beats. Once the program hits a treble clef, it appends the total number of measures for that line to an array. This array would later be sent to the next module to help keep track of where the program was in the music sheet.

For the encoding, each template had an associated hex value (refer to the table in Section 5.3) This was to give each template a unique representation for each section (notes & rests, dynamics, time signatures, and key signatures were all separate dictionaries containing the template name-hex value pairs). These values were then converted to integers in order for the MIDI controller to be able to read them. Here's a layout of what this output looked like:

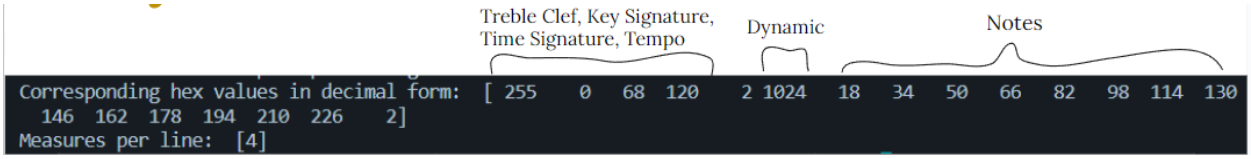


Figure 39: Encoded Output With Number of Measures

The first 4 integers values of every line represent the treble clef, key signature, time signature, and tempo. The treble clef had a unique value of 255 (or 0xFF in hex) and let the code know when the beginning of a new line was. Dynamics were also unique in the fact that they were the only objects that had a hex value greater than 0xFF. Everything else represented a note or rest that would be passed on to the next module. The sheet tested for this output was a simple one-line sheet consisting of the C-Major scale.

6. System Test and Verification

To test and verify the system, the process will be broken down into two general categories. The categories include the software tests and verifications test and hardware tests and verifications. Below contains the information about both test and verification methods.

6.1 Software Systems

6.2 Computer Vision Software

The sections below outline all testing and verification done for the computer vision algorithm used in the final prototype.

6.2.1 Templates Correctly Identified

This metric is defined as the number of templates that are visible on the sheet of music that were properly identified within the dataframe. This only applies to physical objects that appeared on the music sheet and not the inserted/carry-over logic. The goal is to have an average accuracy of 90% or greater of templates correctly identified over all of the music sheets tested.

6.2.2 Templates In Correct Order

This metric is defined as the number of templates that appear in the correct order, reflected on the music sheet. This only applies to physical objects that appear on the music sheet and not the inserted/carry-over logic. An example of templates in the correct order would be having treble clefs start at the beginning of a line, dynamics appearing before the note directly above them, and in general, notes on the left appearing before notes on the right, as well as notes towards the top appearing before notes toward the bottom. The goal is to have an average

accuracy of 90% or greater of the templates being in the correct order over all of the music sheets tested.

6.2.3 Correct Number of Measures

This metric is defined as the number of measures that appear on every line for a music sheet matching up with the correct output calculated by the algorithm. The goal is to have an average accuracy of 95% or greater for the correct number of measures over all of the music sheets tested.

6.2.4 Software Results

This section outlines all of my results regarding templates being correctly identified, templates being in the correct order, and the correct number of measures being outputted. This section also includes some of the various testing sheets I used to calculate my results.

Sheets Tested	Lines Tested	Notes/Rests Tested	Non-Notes/Rests Tested
31	60	736	159

Table 6: Software Testing Statistics

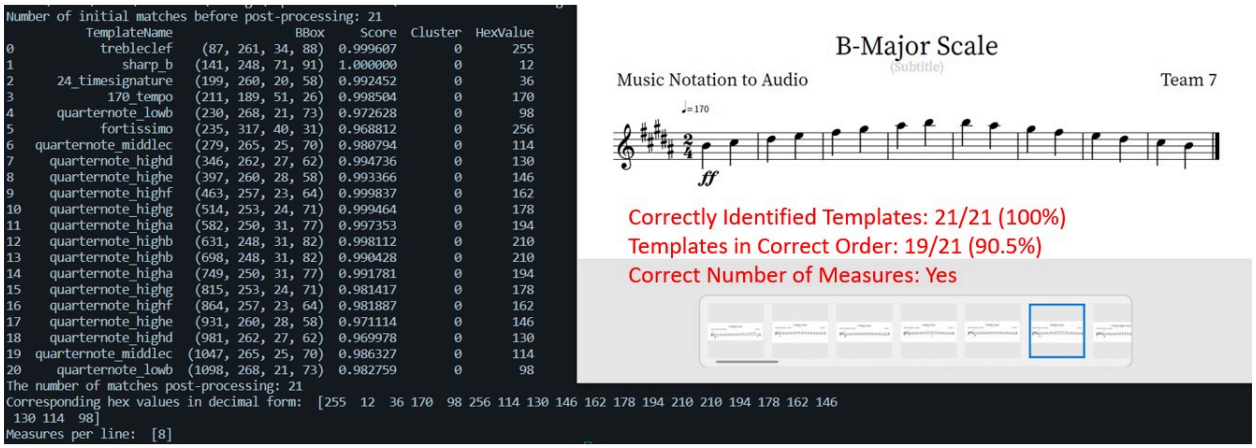
The table above includes the statistics of my testing. The sheets ranged from single-line to full page music sheets (8 lines). Lines consist of a single treble clef, key signature, time signature, dynamics, and the notes/rests that follow until the next treble clef. The notes and rests I tested consisted of sixteenths to whole. The non-notes/rests I tested consisted of the treble clefs, time signatures, key signatures, tempos, and dynamics. Below are my results:

	Average Accuracy
Templates Correctly Identified	100%
Templates In Correct Order	95.8%
Correct Number of Measures	100%

Table 7: Software Results

For all 31 sheets that I tested, I had a 100% average accuracy. This means that for every single sheet, everything that appeared on the music sheet appeared within my data frame. There

was not anything missing and there were not any false-positives. This was achieved by testing different template-matching threshold and overlap parameters until the best values were selected. For all 31 sheets, I had an average accuracy of 95.8% for templates showing up in the correct order. This was due to the double-letter dynamics. During the clustering phase, templates were organized based on the center of their template. The double-letter dynamic templates tended to be wider than the note that appeared above them, so they would always come after the note, when the dynamic should have come before the note. Lastly, for all 31 sheets that I tested, I was able to calculate the correct number of measures that appeared in the music sheet every time, giving me a 100% accuracy. Here are some examples of how I calculated my results:



```
Number of initial matches before post-processing: 21
          TemplateName                BBox     Score  Cluster  HexValue
0           trebleclef    (87, 261, 34, 88)  0.999607        0       255
1              sharp_b   (141, 248, 71, 91)  1.000000        0        12
2      24_timesignature   (199, 260, 20, 58)  0.992452        0        36
3           170_tempo   (211, 189, 51, 26)  0.998504        0       170
4      quarternote_lowb   (230, 268, 21, 73)  0.972628        0        98
5           fortissimo   (235, 317, 40, 31)  0.968812        0       256
6   quarternote_middlec   (279, 265, 25, 70)  0.980794        0       114
7     quarternote_highd   (346, 262, 27, 62)  0.994736        0       130
8     quarternote_highe   (397, 260, 28, 58)  0.993366        0       146
9     quarternote_highf   (463, 257, 23, 64)  0.999837        0       162
10    quarternote_highg   (514, 253, 24, 71)  0.999464        0       178
11    quarternote_higha   (582, 250, 31, 77)  0.997353        0       194
12    quarternote_highb   (631, 248, 31, 82)  0.998112        0       210
13    quarternote_highb   (698, 248, 31, 82)  0.990428        0       210
14    quarternote_higha   (749, 250, 31, 77)  0.991781        0       194
15    quarternote_highg   (815, 253, 24, 71)  0.981417        0       178
16    quarternote_highf   (864, 257, 23, 64)  0.981887        0       162
17    quarternote_highe   (931, 260, 28, 58)  0.971114        0       146
18    quarternote_highd   (981, 262, 27, 62)  0.969978        0       130
19  quarternote_middlec  (1047, 265, 25, 70)  0.986327        0       114
20     quarternote_lowb  (1098, 268, 21, 73)  0.982759        0        98
The number of matches post-processing: 21
Corresponding hex values in decimal form:  [255  12  36 170  98 256 114 130 146 162 178 194 210 210 194 178 162 146
 130 114  98]
Measures per line:  [8]
```


Figure 40: Testing the B-Major Scale

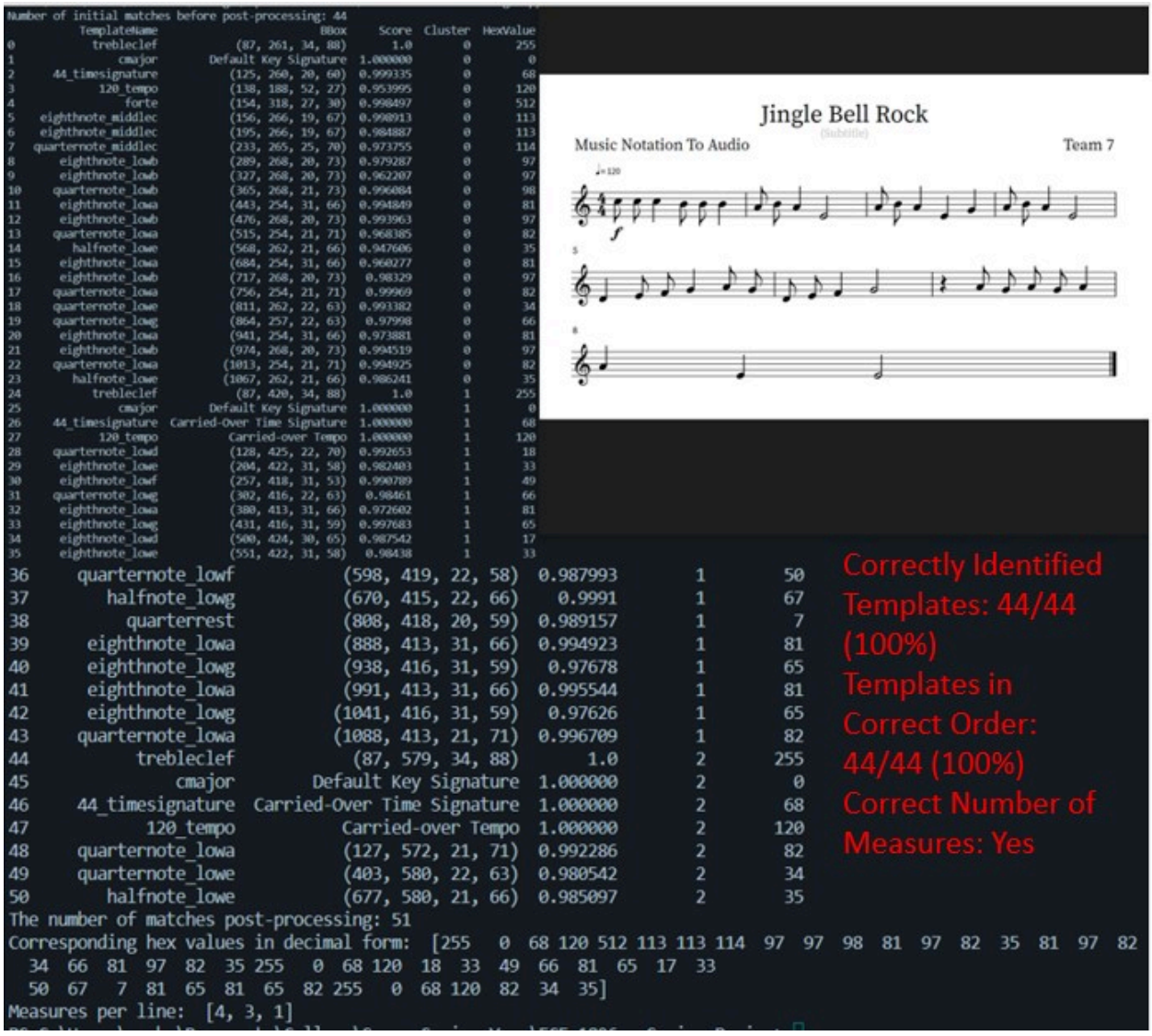


Figure 41: Testing Jingle Bell Rock

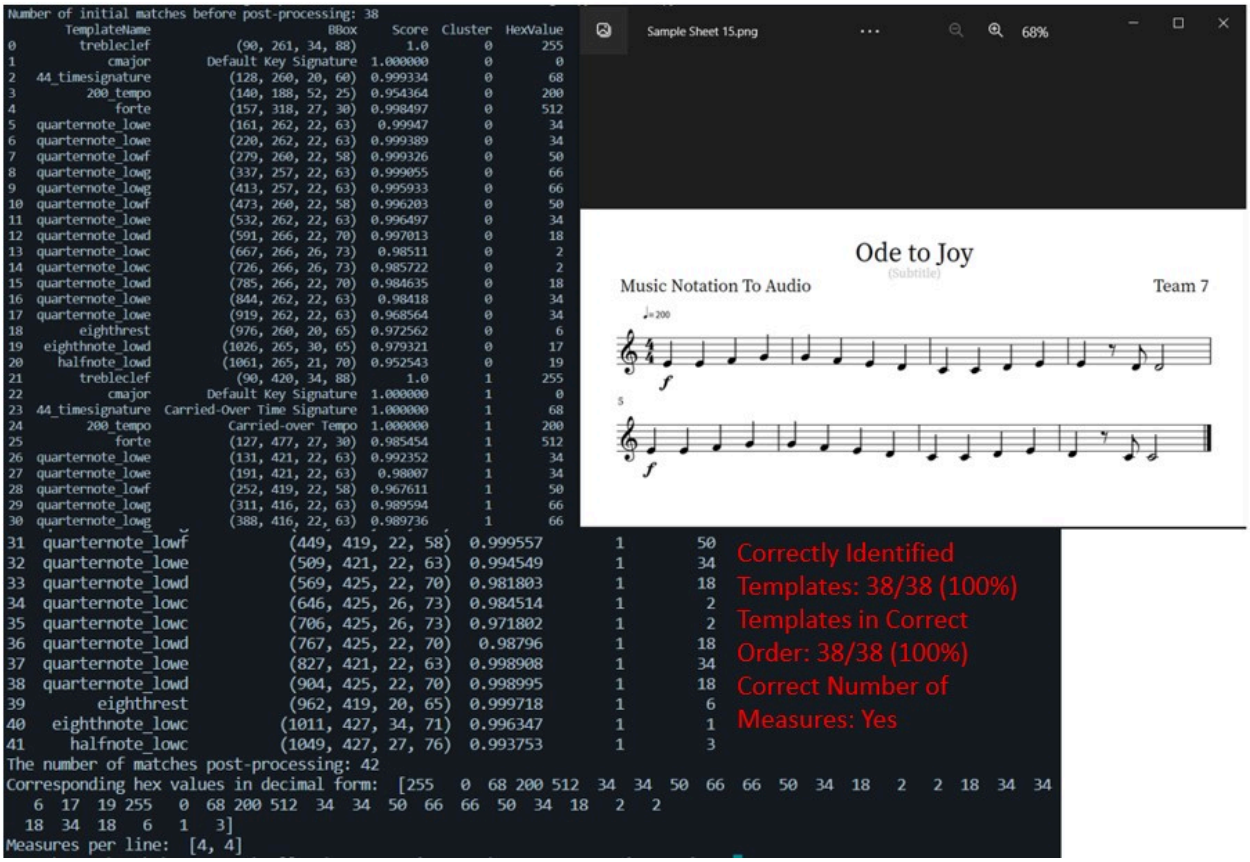


Figure 42: Testing Ode To Joy

6.3 Hardware Systems

6.3.1 Audio Filter Testing

For this project an audio filter was constructed with a cutoff frequency of roughly 18 kHz. The frequency response of this audio filter would be the primary focus of this testing. Given that the resistors used have a 5% tolerance, it was desired that the filter have a percent error in cutoff frequency of less than 5%.

6.3.1.1 Procedure

To test the low pass audio filter designed in this project, the circuit was built on LTSpice. Then, a sweep simulation was performed to get the analytical frequency response of the low pass filter. Then, the actual designed and constructed circuit was tested in the lab to get its experimental response. A sweep was run on Sweep VI, as used in ECE 1212 Circuit Design Lab, using a 3.3 V_{pp} signal with a 1.65V offset produced by function generators and measured by digital multimeters. This input signal is representative of the maximum amplitude signal that could be produced by the Arduino Due's analog output. The sweep was run from 100 Hz to 100kHz in 500 Hz increments. Sweep VI produced a CSV file of the results, which was imported into MATLAB where it was analyzed. The expected and experimental bode plots were compared and the percent error of cutoff frequency was measured as the following.

$$Error = \frac{f - f_{expected}}{f_{expected}} * 100\%$$

6.3.1.2 Results

Below are the Bode plots from LTSpice and from Sweep VI as measured in the lab. As can be seen, the two plots line up closely.

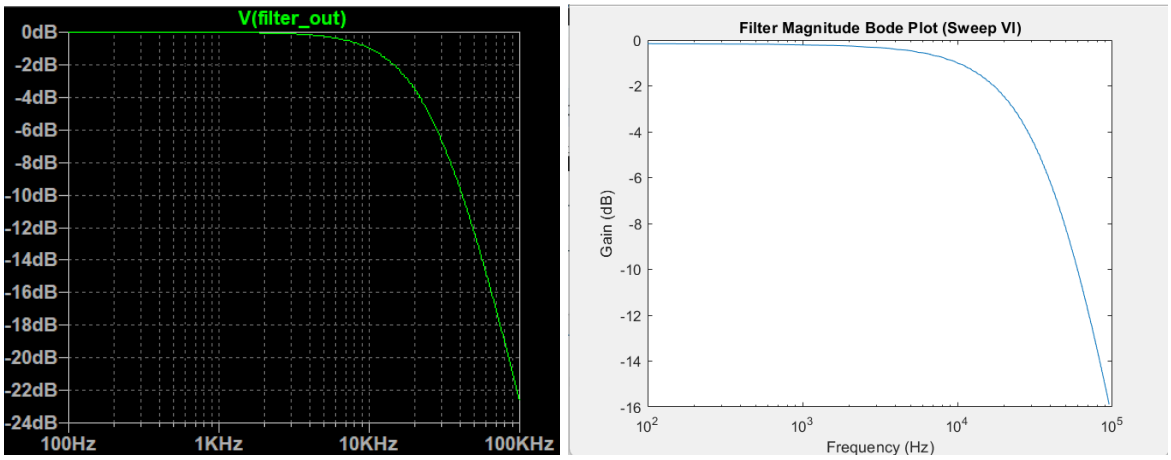


Figure 43: Bode Plot Comparison

Below is a table showing the expected cutoff, measured cutoff, and percent error. Note that the percent error is less than 5% which is acceptable given the tolerances of the components used. It is a small margin of error and supports the idea that the filter behaves as designed.

Metric	Measured Value
Expected Cutoff Frequency	18.27 kHz
Experimental Cutoff Frequency	19.1 kHz
Percent Error	4.03%

Table 8: Bode Plot Results

6.3.2

6.3.3 Audio Signal Spectral Comparison

The MIDI player uses Fourier Synthesis to recreate instrument sounds from the instruments recorded harmonic spectrum. It outputs these instrument sounds at various fundamental frequencies using Direct Digital Synthesis. In music and sound, all notes are based on frequency, so having a small error in this domain is important. The Direct Digital Synthesis was tested by looking at the fundamental frequencies of the output, and the Fourier Synthesis was tested by looking at the harmonic spectra.

6.3.3.1 Fundamental Frequency Error with Direct Digital Synthesis

6.3.3.1.1 Procedure

The fundamental frequency of the note played will be compared to the desired fundamental frequency of that note. The error will be defined as the difference between the frequency and the desired frequency, normalized to the difference between the note and the adjacent note. Notes in music are separated by a factor of $2^{1/12}$. The goal is to have an error less than 5% for each note measured.

Error if $f<f_0$: Error if $f>f_0$:

$$Error = \frac{f_0 - f}{f_0 - \frac{f_0}{2^{1/12}}} * 100\%$$

Equation 6: Fundamental Frequency Error for a Result Less Than Expected Value

$$Error = \frac{f - f_0}{f_0 - 2^{1/12} f_0} * 100\%$$

Equation 7: Fundamental Frequency Error for a Result Greater Than Expected Value

To measure the error, the MIDI controller was commanded to output waveforms at a variety of frequencies across the full range of frequencies that can be played on standard musical instruments (27 Hz to 4.1 kHz) [45]. Since this was evaluating the effectiveness of Direct Digital Synthesis, the instrument used in testing was kept as the simple tones throughout all testing. The maximum deviation for each frequency was recorded and used to find the fundamental frequency error.

6.3.3.1.2 Results

Note	Expected Frequency (Hz)	Measured Frequency (Hz)	Fundamental Frequency Error
A1	55	55.0	0%
C2	65.41	65.4	0.3%
C4	261.63	261.7	0.5%
A4	440	440.0	0%
C6	1046	1047	1.6%
C8	4186	4188	0.8%

Table 9: Fundamental Frequency Error

All errors recorded were less than 2% which supports that all notes played with Direct Digital Synthesis are almost exactly the desired frequency associated with that note. This means that these digitally produced sounds almost exactly align with their desired real sound in terms of base note frequency. Thus, this digital instrument is "in tune."

6.3.3.2 Fourier Power Spectrum Error for Instrument Sounds

The error within a predicted spectrum will be evaluated by finding the ratio of the power of the error compared to the power of the first 20 harmonics of the desired spectrum, as these are the harmonics that are used to recreate the instrument sounds. With c_n and d_n representing the complex coefficient of the nth harmonic of the desired spectrum and recreated audio spectrum respectively, the total error will be defined as shown below. Note that error is desired to be less than 10%.

$$P_c = \sum_1^{20} |c_n|^2$$

Equation 8: Power in the First 5 Harmonics of the Original Spectrum

$$P_{error} = \sum_1^{20} |c_n - d_n|^2 \qquad Error = \frac{P_{error}}{P_c} * 100\%$$

Equation 9: Power of the Error in the First 5 Harmonics

Equation 10: Error in the Power Spectrum

6.3.3.2.1 Procedure

To measure power spectrum error, samples of each instrument were taken across all notes of the full musical operating range of the device in C Major Scale, C4 (261 Hz) to C6 (1046 Hz). Using the Arduino Serial Monitor, the device was commanded to output these waveforms, and an oscilloscope was attached to where the speaker is connected to the circuit. Once the MIDI controller was outputting the waveform, the oscilloscope was zoomed in until it was capturing 5 to 10 periods of the waveform, and a csv file of that data was saved. This data was then imported into MATLAB where a Fast Fourier Transform was taken and the measured coefficients were compared to the desired spectrum using the equations above to find the error for each instrument over the frequency range of the instrument.

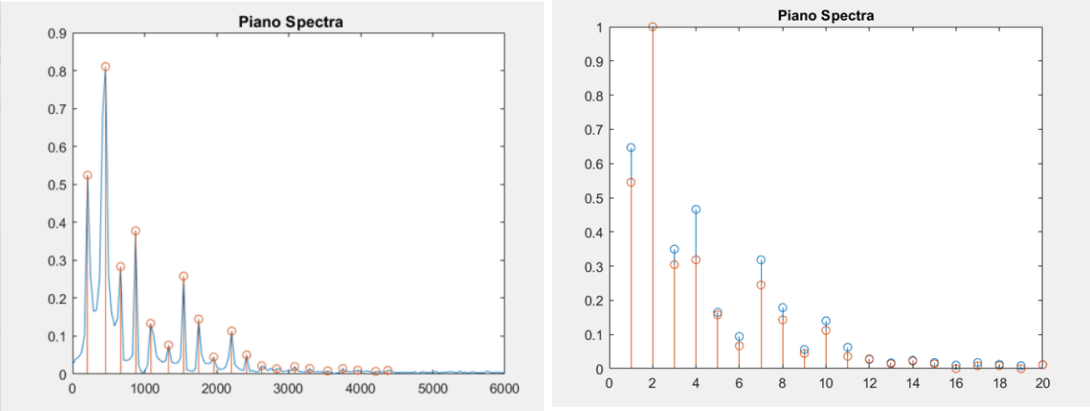


Figure 44: Spectral Comparison in MATLAB

6.3.3.2.2 Results

Below is a graph and table representing the Fourier error of all instruments over the operating range of this device.

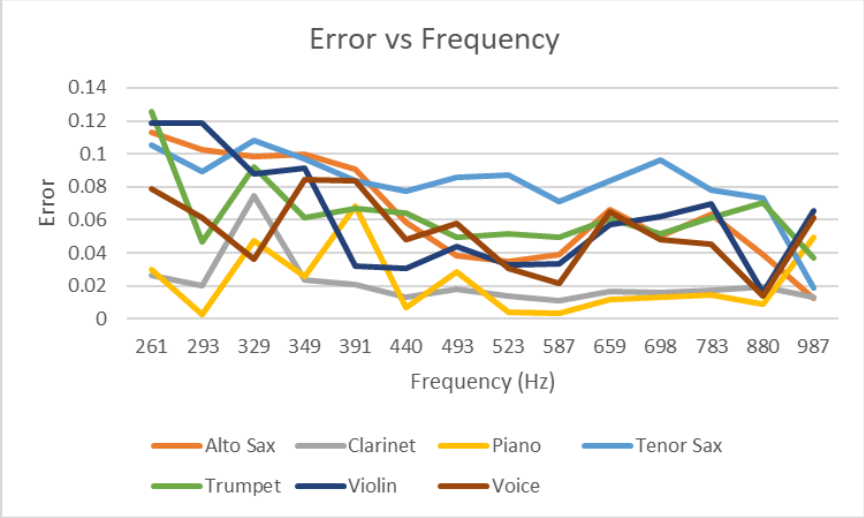


Figure 45: Fourier Spectrum Error vs Frequency for All Instruments

Instrument	Average Fourier Spectrum Error
Alto Sax	6.3%
Clarinet	2.1%
Piano	2.3%
Tenor Sax	7.8%
Trumpet	6.2%
Violin	5.8%
Voice	5.2%
All Instruments	5.09%

Table 10: Average Fourier Error for All Instruments

The average error for each instrument was all under 10% as desired. However, the error for every instrument was not always less than 10% for the full range of the instruments. At the lowest frequencies, alto sax, tenor sax, trumpet, and violin were all over 10%, with the maximum Fourier error being 12.6% for trumpet at lowest frequency. The error decreasing with an increase in frequency is expected with Direct Digital Synthesis. Direct Digital Synthesis performs poorly at lower frequencies when the sample array is of a low resolution, as in when it has a low number of elements compared to the output frequency [80]. Here the output frequency is 48 kHz and the instrument sample arrays are 1000 elements each, so the resolution is fairly low, explaining the error. This error could be improved by getting a microcontroller with more memory and increasing the array resolution.

Note that the instrument that consistently underperforms is the tenor saxophone. Considering the spectra of all sampled instruments, the tenor saxophone stands out because it has harmonic components up to the 10th harmonic that are relatively comparable in magnitude to the fundamental frequency component. More research is needed to determine the cause with absolute certainty, but on the datasheet of the PAM8302A amplifier [48] used, it is shown that the percentage total harmonic distortion is roughly quadrupled at frequencies between 2kHz and 8kHz, so having high amplitude components at those frequencies for a tenor sax likely increases the overall signal's distortion compared to other instrument signals.

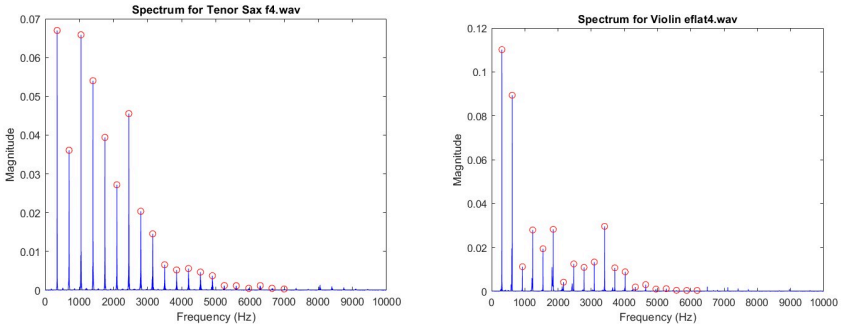


Figure 46: Tenor Sax Spectrum Compared to Another Instrument

6.3.3.3 Spectrum Power Error

Controller Communication Latency

For the control software developed for the Raspberry Pi central controller, the main focus was output lag. Lag for this part of the project was defined as the time it took for all output processes. This was a focus because in music timing is very important, so the goal was to have imperceptible delay between any outputs. The human perception of lag of 13 milliseconds [27], so the goal was to have a delay shorter than that.

6.3.3.4 Procedure

To measure the lag in the Raspberry Pi code, the time() function was used. There were three different outputs operated in the code. To give a worst case measurement, the time was measured before and after all outputs were operated. Given that the next output would not start until the last one was established, this timing would be a measurement of all of these communication processes. The lag was recorded as the time difference between the two measurements. Average lag was calculated to be the representative statistic for this metric.

6.3.3.5 Results

Average Lag	1.48 milliseconds

Table 11: Average Lag

The average lag was recorded to be much less than the 13 milliseconds that was desired, so this was an indication of imperceptible lag in the control code's output.

6.3.4 Voltage and Current Measurements

6.3.4.1 Procedure

Expected voltage and current measurements will be taken to ensure the safety of the microcontrollers and other components of the circuit. A multimeter will be placed in parallel with the areas of interest to measure the proper voltage. The measurements will be taken after the full wave rectifier and at the supply nodes (9V, -9V, 5V, 3.3V). Additionally, voltage measurements will be taken at the point of consumption to measure the power loss between the two PCBs.

6.3.4.2 Results

Below is the voltage and current measurement results for the whole system.

6.3.4.2.1 Component Results

This section outlines the voltage and current measurements for all the components in the Sheet Music Trainer. Table 12 displays the measurements for the Arduino, LCD, and LEDs. The measurements produced were as expected and the power consumption remained in the range of the power supply's capability.

Component	Voltage	Current	Power
Arduino	5.06 V	0.6764 A	3.329 W
LCD	5.03 V	25.683 mA	0.1218 mW
LEDs	5.02 V	0.0320 A	1.6064 W

Table 12: Component Voltage, Current, and Power Measurements

Figure 44, Figure 45, and Figure 46 display current measurements during certain periods of operation for the Raspberry Pi 4 controller. Figure 44 displays current measurements during the time it took the Pi to boot up. The maximum current measured was 0.98 amps. The average current during this time was 0.9183 amps. From operating at 5.02 volts the Pi during this time consumed around 4.61 watts.

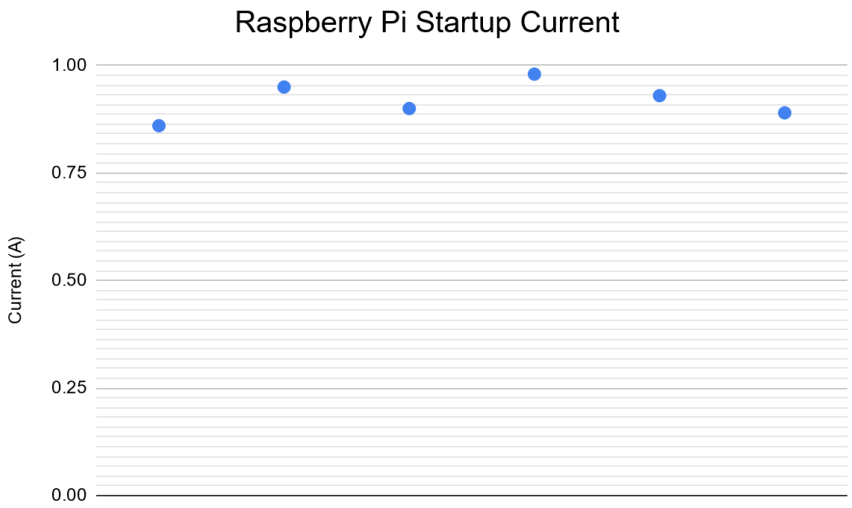


Figure 47: Raspberry Pi Startup Current

Figure 45 displays the current of the Pi while not processing or booting up. For this state, it was labeled as idle. The maximum current during this period was 0.62 amps and on average 0.545 amps. From operating at 5.02 volts the Pi during this time consumed around 2.74 watts.

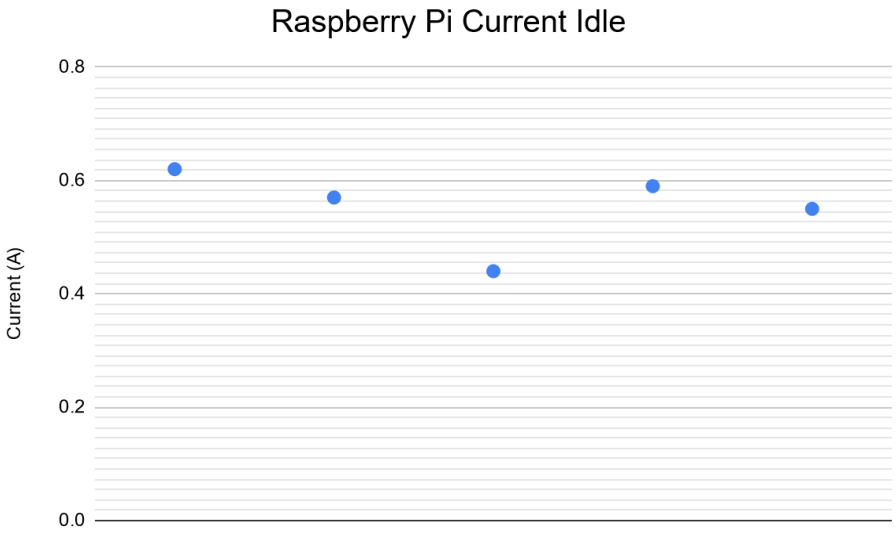


Figure 48: Raspberry Pi While Idle

Lastly, Figure 46 displays current measurements while the Pi is processing the music piece selected. The maximum current at this state was 0.85 amps and the average current was 0.7417 amps. From operating at 5.02 volts the Pi during this time consumed around 3.72 watts.

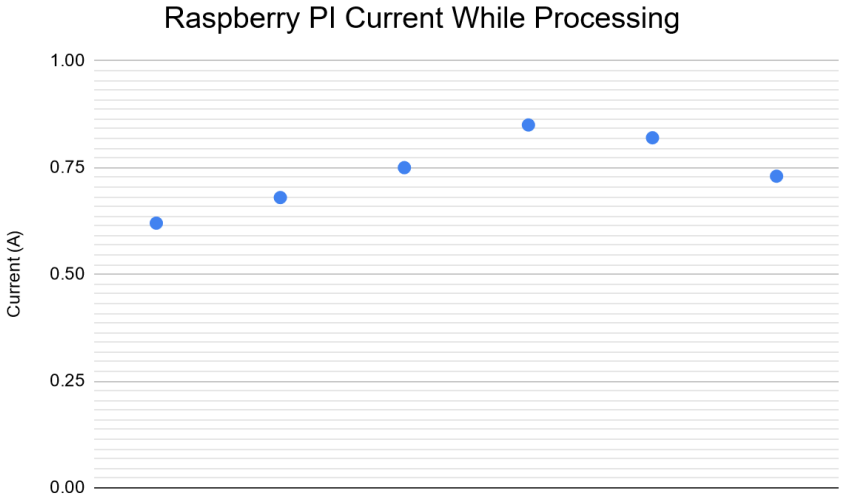


Figure 49: Raspberry Pi While Processing

6.3.4.2.2 Power Connector Results

This section reviews the voltage measurement results for the rails that connect to the audio PCB. Below displays the voltage from the rectifier to the DPST switch that allows power to the board. The voltages are lower than expected and this can be due to the soldering of the larger diodes to the PCB along with standard power losses in the rectification process itself.

From Rectifier	Voltage (V)
Positive Rail	8.7048
Negative Rail	-8.725

Table 13: Dual Rail Results Directly from the Full Wave Rectifier

Table TCJ4 displays the input and output voltages from the two buck converters utilized in the design. It can be seen when comparing the voltage into the voltage supplied from the positive rail of the rectifier there are some losses. The drop in voltage can occur through the traces along the board along with the physical wiring to the buck converters themselves. However, it is important to note the buck converters are able to produce a steady output voltage if the input voltage is varied. This is vital for the devices being powered from the five volt line mentioned in Section 6.2.3.2.1.

Buck Converters	Vin	Vout
5 V Output	7.53 V	5.170 V
3.30 V Output	7.63 V	3.320 V

Table 14: Buck Converter Measurements

The Table below displays the results when measuring the voltage of the power connector pins while in the device's three modes of operation: battery powered, AC and battery powered, and in battery charging mode. It can be seen that the batteries produce 7.309 volts and -7.7413 volts for the audio board. This is lower than the expected value, however this can be due to the batteries going through multiple life cycles while testing along with batteries matching the battery with the lower voltage when operating in parallel. Additionally, even with the output not being as expected the positive and negative reference allowed the audio circuitry to operate as expected. The powering mode was able to produce a closer result through producing a 8.4183 volt positive rail and a -8.7616 volt negative rail. Both operating modes were able to produce a stable 5.01 volt and 3.32 volt output due to the buck converters. Lastly, when operating in battery charging mode, there is very little voltage to the power connector, thus meaning the devices are disconnected from the main supply while the batteries are charging.

Pout Connector	Just Batteries	Powering Mode	Battery Charging Mode
Pin 1	7.309 V	8.4183 V	0.1690 mV
Pin 2	5.0149 V	5.0169 V	0.1030 mV
Pin 3	3.32 V	3.3214 V	0.0080 mV
Pin 4	0.109 mV	0.0104 mV	-0.3250 mV
Pin 5	-7.7413 V	-8.7616 V	0.00183 mV

Table 15: Power Connector Rail Voltage Results

6.3.5 System Recharge Period

6.3.5.1 Procedure

The battery recharge system will be tested through comparing the time needed to charge the battery to the calculated time it takes. The calculated time it takes for the battery to recharge the battery is displayed in Equation 1.

6.3.5.2 Results

Below represents the values when testing the charging current to the backup battery system. When connected in operating conditions, the batteries experience the least amount of current and take almost a year to charge. When only disconnected from the negative rail of the transformer, the battery experiences a charge around 81 times greater than before. This allows the batteries to charge within four days. Lastly, when the center tap was disconnected the batteries experienced a current closer to the expected 1.2 A and would charge within the hour. All battery charging time measurements were calculated utilizing Equation 1.

Connected to Negative Rail and Center Tapped Ground		
I Charge	83.015 uA	
Time	7227.61 hrs	301.15 days
Disconnected from Negative Rails		
I Charge	6.796 mA	
Time	88.29 hrs	3.68 days
Disconnected from Negative Rails and Center Tapped Ground		
I Charge	0.987 A	
Expected Time	0.608 hrs	36.47 mins

Table 16: Battery Charging Results

Overall, in battery operating mode, the batteries were able to last at least an hour of operation. Testing did not last over an hour due to an hour being the standard recommended practicing time for a musician. However, when operating off AC power while with current from the batteries when needed, the device lasted four hours. The testing period did not last longer than four hours, thus meaning the device powered from both could last longer.

7. Team

The team consists of two electrical engineers (Jack Carnovale and Chloe Hale) as well as one computer engineer (Cameron Henning). Jack and Chloe will be handling the hardware modules, which includes power supplies, microcontrollers, and PCBs. Cameron will be handling the software module which includes a computer vision algorithm and a slight overlap with Jack in communication between the computer vision module and the Raspberry Pi.

7.1 Jack Carnovale

Jack Carnovale is an Electrical Engineering major with a concentration in power and a passion for controls. He has done two internship rotations with Eaton Corporation in controls and field service. During the school year, he works for the Student Electronic Resource Center. These experiences will be useful as he takes on his role in this project. Jack will be handling the controls and timing of the Raspberry Pi and all visual and audio outputs. He will also be responsible for designing the MIDI synthesizer from.a microcontroller. He will also be helping with enclosure design and hardware assembly.

7.1.1 Skills learned in ECE coursework

Because Jack's module in this project involves working on a microcontroller and embedded control, ECE 0202 Embedded Processors and Interfacing will be vital to his module. Knowledge from this class will be useful for the communication, processing, and microcontroller coding. The information from courses ECE 0101 Linear Systems and Circuits, ECE 0102 Microelectronics, and ECE 1212 Circuit Design Lab will be used for filter and amplifier design in the audio circuit, as well as the circuitry needed for configuring the electronic hardware for the microcontroller and user inputs. The prototyping and PCB design skills learned in ECE 1895 Junior Design Fundamentals will be also utilized in making the final circuit boards for the audio and microcontroller circuitry. Last, skills from ECE 0402 Signals, Systems, and Probability as well as ECE 1560 Digital Signal Processing will be used in using the Fast Fourier Transform to conduct spectral analysis and signal composition. Skills in musical signal processing learned in ENGR 0501 Music Engineering Laboratory will be useful as well. These three will be key in making audio that sounds like real instruments on the MIDI synthesizer.

7.1.2 Skills learned outside ECE coursework

For many aspects of Jack's portion of this project, a higher level of skill must be attained beyond what has been taught in the ECE courses. A more applied and thorough use of spectral analysis must be used for the MIDI playback of different instrument sounds. For this reason, Dr. Steve Jacobs will serve as Jack's faculty advisor for his module. Dr. Jacobs has a strong background in signal processing, and he has a musical background as well. Dr. George Stetten (BioEngineering) teaches the Music Engineering Laboratory and has a similar background to Dr. Jacobs, so he will be a useful resource as well who can be contacted easily through email.

For Jack's module, there will also need to be independent internet research into audio amplification and filtering, as well as programming a microcontroller. Through ECE courses, a basic understanding of these has been achieved, but for this project, a deeper study is needed. The biggest challenge will be the speed and time management of the signals to control lights, displays, and output audio. This will likely involve more research into embedded systems, interrupts, and communication protocols. Dr. Sam Dickerson and Dr. Jingtong Hu may be good resources for this independent research as well. Last, Jack has never coded a Raspberry Pi before, so there will need to be independent research in this area as well.

7.2 Chloe Hale

Chloe is an Electrical Engineering major with a concentration in power. Her one year of industry experience includes three co-op rotations at Acutronic USA Inc. There she helped the electrical engineering test create test panels for amplifiers, complete factory acceptance testing on multiple machines, and assist with proposal budgeting management. She will be responsible for designing the power supply for the system along with the required hardware for the Raspberry Pi and input and output sensors. The power circuitry will consist of a 120 volts input with a rechargeable 18 volts battery backup system. The Raspberry Pi controller circuitry will consist of individually accessible LED, LCD, SD card reader, rotary encoder, and input buttons. Along with the electrical design, Chloe will help design the enclosure for the system.

7.2.1 Skills learned in ECE coursework

A list of relevant courses Chloe has taken during her four years at college include ECE 0101, ECE 0102, ECE 1701, ECE 1771, ECE 1212, and ECE 1895. The base for any circuit design falls back upon the person's knowledge and understanding of the basics of circuit analysis. ECE 0101, Linear Circuits and Systems, and ECE 0102, Microelectronic Circuits, helped create and solidify Chloe's circuit analysis from RLC circuits to transistors. Resistors, capacitors, and transistors are included in the power design module. ECE 1701, Fundamentals of Electric Power Engineering, taught Chloe the needed skills to understand power consumption analysis along with the basic operations of transformers. ECE 1771, Electric Machinery, expanded her knowledge of the operations of transformers and introduced alternating current (AC) to direct current (DC) conversions. A transformer along with a full-wave rectifier is included in the power module design.

Additionally, ECE 1212, Electronic Design Lab, helped expand on the knowledge gathered in the courses mentioned previously. In ECE 1212, she was able to learn how to meet the design requirements given by the professor for component focused labs. This skill helped her meet design requirements for the controllers and sensors along with creating an indication system for if the battery is charged. Also, the skills learned in ECE 1212 helped with her projects in ECE 1895, Junior Design. Junior Design gave Chloe the opportunity to learn the design process along with the general knowledge and skills to take a circuit on a breadboard to a printed circuit board (PCB). In general those skills will help her during the power supply design along with the Raspberry Pi circuit design. Lastly, Junior Design introduced skills like 3D modeling (Fusion360), 3D printing, and laser cutting. These skills helped her learn how to prototype an enclosure design. This skill learned from ECE 1895 will be utilized to create the enclosure for the device.

7.2.2 Skills learned outside ECE coursework

To help better understand the general problem the system solves, Chloe has years of experience being in the orchestra, string ensemble, and a performing quartet. This knowledge has helped her understand the necessity of this device to help others and beginner musicians. Additionally, to her music experience, Chloe has taken music theory classes along with a piano class at the University of Pittsburgh. With this background, she is able to help provide examples of music along with answer questions to help assist other members of the team. For technical background, during co-op at Acutronic, Chloe was exposed to the engineering design process along with her own projects. Chloe was tasked with designing two test amplifier panels along with designing a signal amplifier board. While creating the test amplifier panels, Chloe was able to learn how to follow industry standards as well as understand the device itself from the technical manuals. Chloe was first exposed to PCB design, using EAGLE, while creating the signal amplifier board.

As well as background knowledge through personal experience, research was needed for her to better understand the design needed. More research was needed to be able to understand the operating system and pins of the Raspberry Pi as well as the power intake of the microprocessor. Additionally, more research will be done to find the best way to test and verify the charging of the battery along the consistency in the power supply output. Along with experienced Alumni, the person within the faculty of the ECE department that would be best to talk to about this subject would be Dr. Grainger due to his knowledge of power electronics.

7.3 Cameron Henning

Cameron is a senior computer engineer with a passion for software development as well as the recent AI advances in today's society. He completed two rotations with Philips Respironics, working with other software developers to test chat bots, analyze outputs, create and label data, and assess the flow of programs. He then completed his last rotation at BNY Mellon, working with a team of backend and frontend developers to write tests for various applications within BNY Mellon's Market Technology. He will be taking the role of dealing with the software side of the project. He will be responsible for image detection, image processing, and computer vision techniques to correctly classify the musical variables of sheet music. He will be working with a camera combined with OpenCV to analyze sheet music and load it into the algorithm.

7.3.1 Skills learned in ECE coursework

Some of the relevant coursework that will be applied to this project are ECE 302, ECE 1395, ECE 1895, and ECE 1140. ECE 1395 will play an important role in Cameron's part of the project, because this is where most of his machine learning knowledge comes from. Image processing, support vector machine (SVM), and coding in Python are all important topics that came from this class. ECE 302 and ECE 1895 had focused on using GitHub for version control. ECE 302 gave Cameron the knowledge of different data structures and algorithms that could be applied to his computer vision techniques. ECE 1140 played a significant role in using visual studio and package management.

7.3.2 Skills learned outside ECE coursework

From the introduction, Cameron has had co-op experience with two different companies: Philips Respironics and BNY Mellon. Philips Respironics had taught Cameron to work with tagging data and creating data, which is very important when it comes to training models in machine learning. Cameron also practiced his Python knowledge by creating an automated data documentation system, which can be applied to automating musical data within this project. At BNY Mellon, Cameron learned various skills such as coding with Java, working with Spring Boot Applications, and testing with Mockito and JUnit. This can play a role in debugging errors and problems that may arise with the computer vision algorithm and logic. Cameron expects to do more research into OpenCV, as well as seeing whether tackling the project from an image processing perspective rather than a machine learning perspective would be beneficial and more efficient.

8. Schedule and Budget Plan

The sections below detail each team member's schedule for completing their module along with the budget plan for the project.

8.1 Project Schedule

Date	Jack Carnovale	Chloe Hale	Cameron Henning
9/11 - 9/15	Order parts & research	Order parts & research	Order parts & research
9/18 - 9/22	Conceptual design document	Conceptual design document	Conceptual design document
9/25 - 9/30 **Conceptual Design due 9/29**	Get arduino to play tones off of MIDI commands, Filter Design	Start connection assembly to prepare for breadboard testing, Begin power design testing, Parts come in, Finalize pins for Raspberry Pi	Figure out what descriptor to use for template matching, Start template matching sheet music with labeled images
10/2 - 10/6	Start gathering musical data, Fourier analysis and composition of samples, start audio circuit design	Continue and finalize design testing, Begin battery measurements and tests, Order LEDs and build controller design test setup, Gather other sensors for controller (LCD, potentiometer)	Continue testing and working on matching variables with labeled images, increase the accuracy of the images being matched
10/9 - 10/13 **First Check Off (Cohort II) on 10/11**	Implement instruments in code, run testing on audio circuitry	Start Raspberry Pi setup and begin testing in debug mode, continue battery/power testing.	Have some type of image processing/machine learning algorithm to correctly identify notes, translate the output data into binary/hex representation
10/16 - 10/20	PCB design for Audio and MIDI circuitry, Implement digital direct synthesis on Arduino with tones	Start PCB design as sections are verified, continue testing/debugging as needed	Import module to the Raspberry Pi

10/23 - 10/27 **Midterm Presentation (Cohort II) - 10/25**	Finalize PCB design and order, Finalize instrument coefficients on device	Continue PCB design and finalize, continue board testing as needed	Work with Jack to communicate the output data to the MIDI controller
10/30 - 11/3	Start writing central control code and testing I/O for RPI, assemble and test audio PCB	Send final parts order if needed, Begin enclosure design, Start PCB assembly (power portion)	Work with capturing images using the camera, Debug and configure resolution of images
11/6 - 11/10	Establish RPI communication with MIDI controller, Establish overall RPi code and musical timing in final shell	Finish PCB assembly for power section, start testing with PCB	More debugging using the Raspberry Pi
11/13 - 11/17 **Second Check Off (Cohort II) - 11/15**	Debugging and audio circuit redesign, Develop Audio Envelopes	Finish measurements and testing on PCB, start implementing Raspberry Pi on PCB	Computer vision algorithm should be finalized
11/27 - 12/1	Reassemble and retest audio circuitry, Integrate with Cam and Chloe, Help with Assembly	Debugging	Continue testing, finalize module connection with Jack
12/4 - 12/8 **Final Presentation (Cohort II) - 12/6** **Senior Design EXPO - 12/8**	Finalize poster and Present	Finalize poster section, start writing final report, start total system assembly	Work on final report, testing as needed, test algorithm to ensure proper accuracy
12/11 - 12/15 **Final Report due 12/15**	Finalize report	Finalize report, prepare for presentation, finalize total design	Finalize report

Table 17: Team's Individual Schedule

8.2 Project Budget

Below is the overall budget for the Sheet Music Trainer.

Section	**Part**	**Action**	**Possible Cost**	**Supplier**
Inputs/Sensors	Rotary Encoder	Have	$1.28	Digikey
	Scan Button	Have	$0.18	Digikey
	LCD Display	Have	$8.99	Amazon
Controllers	Raspberry Pi 4	**Have**	$34.98	Amazon
	Arduino Due	Have	$39.99	Arduino
Power Circuitry	Outlet with Rocker and Fuse	Have	$6.99	Amazon
	Transformer	**Bought**	$20.00	Digikey
	Buck Converter (1)	**Bought**	$10.49	Amazon
	Buck Converter (2)	Have	$13.99	Amazon
	Schottky Barrier Rectifier Diodes (x30)	**Bought**	$6.99	Amazon
	Diodes (1N4007) x2	Have	$0.27	Digikey
	18V relays (x2)	**Bought**	$1.42	Digikey
	1k resistor (x2)	Have	$0.30	Digikey
	100 uF capacitor (x2)	Have	$0.52	Digikey
Controller Circuitry	Op Amps	**Bought**	$1.26	Digikey
	LED Strips	Buy	$12.50	Digikey

	Potentiometer (x2)	Have	$2.10	Digikey
	12 Bit DAC	Have	$2.99	Amazon
	16 Bit ADC	Have	$7.99	Amazon
	4 Ohm Speaker	Have	$8.00	Grainger
PCBs	Power and Control PCBs	Buy	$8.00	JLCPCB
		TOTAL:	**$189.23**	

Table 18: Overall Project Budget

9. Conclusion

9.1 New Skills Acquired and Learning Strategies

9.1.1 Jack Carnovale

Throughout the course of the project, Jack used knowledge from past classes to develop his audio and control module. He has worked with filter design, Arduinos, and interrupts before. However, new skills were acquired in working with audio circuitry. He also learned about designing a variety of different audio amplifier circuit configurations [47] in designing his original amplifier circuit like single ended, bridge-tied load, and fully differential audio amplifiers. While these amplifier designs were common and useful, the audio sounded distorted with the original amplifier design. This required researching the causes of audio distortion and learning about impedance matching [76]. This led to a redesign of the audio module with a new amplifier that is impedance matched to the speaker. Direct Digital Synthesis was also a new concept learned as a means to play periodic waveforms at variable frequencies [73]. He also researched new ways to speed up controller processing [79] like using fixed point math instead of floating point math [75].

A lot of new skills acquired for Jack were involving using the Raspberry Pi. Jack had never used a Raspberry Pi before this project, and he was in charge of writing the central control code for it. Through troubleshooting with the team's original Raspberry Pi Zero 2, Jack learned and executed multiple ways to access a Pi that aren't through the HDMI port. He learned how to use SSH and VNC connection methods [77]. He learned how to write a file for a Pi and get it to run on boot [78]. Also, he learned the importance of modularization and abstraction of software. When developing software, it is important to be able to develop code as one function that can be used by other teammates through only knowing the functions general purpose and its inputs and outputs. These ideas allowed the process of integration of software to be relatively smooth.

9.1.2 Chloe Hale

Since the beginning of the project, Chloe was applying old knowledge gathered from her previous courses (as mentioned in Section 7.2). This applied for the AC to DC conversion with the full wave rectifier and the use of a transformer. However, the new skills acquired for the

portion of the project was safety with working with AC mains [69]. This was vital in the early stages of testing the project and important to take into consideration during the PCB design process [68]. The knowledge to do so was gained through trial and error as the adjustment of location of components along with trace widths needed to be made three times.

Additionally, Chloe had spent most of research time looking into ways to create a dual rail output along with a battery charging mechanism. This proved to be difficult since none were implemented along with most rechargeable designs including the batteries being isolated and not having a powering option. Also, safety of charging the lithium ion batteries needed to be learned and taken into consideration in the design process as well [70]. All new knowledge to be gained. Lastly, through assisting with raspberry pi coding, Chloe was able to get her first look into python programming.

9.1.3 Cameron Henning

At the start of the project, Cameron had very little knowledge or background in computer vision. He had a little bit of experience in machine learning from his co-op at Philips and Intro to Machine Learning class. However, the application of these skills was very new to him, forcing him to look into viable resources that could be used for the computer vision algorithm. He learned about a method of computer vision called template matching, which in some cases uses cross-correlation to match templates to objects on an image. There was very little research done on template matching with Optical Music Recognition, so looking into the documentation and learning how template matching worked was one of the skills Cameron had to learn. Cameron also used his knowledge of machine learning to look into various techniques when it came to organizing the data. He researched different methods and ultimately was able to apply his knowledge of clustering in order to properly sort the output data coming from the template matcher. Lastly, Cameron applied his knowledge of working with GitHub to organize his code efficiently and effectively, to prevent errors from creating roadblocks in his progress.

9.2 Future Work

Future work would be developing the concepts deeper that have already been implemented. For the Sheet Music Trainer it would be considered a newer model. The model would feature more LEDs to track the line the user is on along with more selections for tempo and wider ranges of notes. More dynamics and other variables provided by the NoteFlight website would be implemented in the software along with basic chords and the bass clef. Also, the newer model would include camera implementation along with bluetooth to allow an uploadable music database.

Additionally, the model would have a decreased time to charge along with an increase in the time the batteries can power the product. To decrease charging time would involve creating a charging circuit regulated through a buck-boost converter to account for voltage fluctuations. Additionally, the battery charging mode would involve one switch rather than two to create a less confusing model. Lastly, to increase the time running on battery power, the batteries would be replaced with a different lithium ion battery with a higher capacity. Lithium ion batteries would still be used due to being lightweight.

The future model would be improved in terms of its audio quality and options. Audio envelopes would be developed further towards perfection, and there would be experimentation with the measured harmonics to determine the ideal number of harmonics for each instrument, instead of just using the first 20. A microcontroller with a larger memory would also be

considered such that the resolution of all waveforms on the MIDI controller could be increased further. These would improve the sound quality from the microcontroller. Beyond that, more instruments would be analyzed and implemented on the device such that the musician using the device would be able to cater their practicing to a wider range of instruments.

References

1 "How to Read Sheet Music for Beginners: A Step-by-Step Guide," *moises.ai*. https://moises.ai/blog/tips/how-to-read-sheet-music/ (accessed Sep. 27, 2023).

2 L.Mirinjian, "Where Piano Meets Passion," *Lara's Music*, Mar. 01, 2017. https://www.larasmusic.com/still-cant-sight-read/ (accessed Sep. 27, 2023).

3 "Playing by Ear: 5 Reasons Why Reading Sheet Music Is Still Important," *Musicnotes*, Jan. 24, 2019. https://www.musicnotes.com/blog/5-reasons-why-playing-by-ear-why-reading-sheet-music-is-still-important/ (accessed Sep. 27, 2023).

4 "The purpose of sheet music," *Acoustic Notes*, Jan. 27, 2017. https://acousticnotesblog.wordpress.com/2017/01/27/the-purpose-of-sheet-music/ (accessed Sep. 27, 2023).

5 "National Arts Education Status Report 2019," *Arts Education Data Project*. https://artseddata.org/national_report_2019/

6 "Music-Making in America," *aswltd.com*. https://aswltd.com/gallup.htm#:~:text=More%20than%20one%2Dhalf%2052 (accessed Sep. 27, 2023).

7 Dianna, "ScanScore: A Music Scanner and Sheet Music Reader | SCANSCORE," *https://scan-score.com/en/* , Apr. 13, 2021. https://scan-score.com/en/sheet-music-reader/?gclid=CjwKCAjwsKqoBhBPEiwALrrqiLCTaAyiX8NJf1_xerCUFMmrhwwdqRH39X5k7-Kuq6MkKD5oKISUrBoCEWIQAvD_BwE

8 "MUSITEK - Music Scanning Software," *www.musitek.com*. https://www.musitek.com/ (accessed Sep. 27, 2023).

9 "Sheet Music Scanner," *App Store*, Sep. 25, 2023. https://apps.apple.com/us/app/sheet-music-scanner/id884984324 (accessed Sep. 27, 2023).

10 M. Halliday, "PlayScore 2 Sheet Music Scanner," *PlayScore*. https://www.playscore.co/

11 "Sheet Music Scanner | SCANSCORE Sheet Music Scanning Software," *https://scan-score.com/en/*. https://scan-score.com/en/ (accessed Sep. 27, 2023).

12 F. Jabr, "Why the Brain Prefers Paper," *Scientific American*, vol. 309, no. 5, pp. 48–53, 2013, Accessed: Sep. 27, 2023. [Online]. Available: https://www.jstor.org/stable/pdf/26018148.pdf?casa_token=DzfEZE16QXEAAAAA:6P2akEuJTZQ6qCodkJWxMAB2zCHv63Z8ITg_YpTyCMMaBbdYEW7hkWA77Ww_DI75tFU1NQeeDLlQAD8AKJNrd2piZsa98CQCfWQDWJuz89KxWsxNb0L2m

13 S. Miller, "Hearing Sheet Music: Towards Visual Recognition of Printed Scores." Accessed: Sep. 27, 2023. [Online]. Available: http://vision.stanford.edu/teaching/cs231a_autumn1213_internal/project/final/writeup/distributable/sdMiller_Paper.pdf

14 A. Nyati, "cadenCV: An Optical Music Recognition System with Audible Playback."

Available: https://firebasestorage.googleapis.com/v0/b/afika-nyati-website.appspot.com/o/design%2Fcadencv%2Fcadencv_afika_nyati.pdf?alt=media&token=a5aa2413-32c0-4bc7-8222-06342b822096

15 J. Calvo-Zaragoza and D. Rizo, "End-to-End Neural Optical Music Recognition of Monophonic Scores," *Applied Sciences*, vol. 8, no. 4, p. 606, Apr. 2018, doi: https://doi.org/10.3390/app8040606.

16 "MusicalPi – Source Code – Linwood Ferguson," *www.leferguson.com*. https://www.leferguson.com/musicalpi-source-code/.

17 J. Preis, "How to Read Piano Sheet Music," Hoffman Academy. https://www.hoffmanacademy.com/blog/how-to-read-piano-sheet-music/.

18 "How to Read Music," Instructables. https://www.instructables.com/How-to-Read-Music/?utm_source=base&utm_medium=related-instructables&utm_campaign=related_test.

19 Dyslexic Musician, Dynamics Anchor Chart. Accessed: Sep. 26, 2023. [Digital]. Available: https://www.teacherspayteachers.com/Product/Dynamics-Anchor-Chart-4369260?st=d6bded11fe7b7c5899b65d1b76038c9b

20 "MIDI tutorial for programmers," *Cmu.edu*, 2012. https://www.cs.cmu.edu/~music/cmsip/readings/MIDI%20tutorial%20for%20programmers.html

21 "What Is Computer Vision? | Microsoft Azure," *azure.microsoft.com*. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-computer-vision/#:~:text=Computer%20vision%20is%20a%20field

22 M. Bell, "Fourier analysis in Music - Rhea," *Projectrhea.org*, 2013. https://www.projectrhea.org/rhea/index.php/Fourier_analysis_in_Music

23 "Spectral Analysis - MATLAB & Simulink," *www.mathworks.com*. https://www.mathworks.com/help/dsp/ug/spectral-analysis.html

24 QOITECH, Ed., "Battery Life Calculator," DigiKey. https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-battery-life.

25 "MIDI Tutorial - SparkFun Learn," *learn.sparkfun.com*. https://learn.sparkfun.com/tutorials/midi-tutorial/all#:~:text=MIDI%20uses%20a%20clock%20rate.

26 "BPM tempo and delay to time conversion calculator tempo - sengpielaudio Sengpiel Berlin," *www.sengpielaudio.com*. http://www.sengpielaudio.com/calculator-bpmtempotime.htm

27 "man portable (US DoD Definition)," *www.militaryfactory.com*. https://www.militaryfactory.com/dictionary/military-terms-defined.php?term_id=3207.

28 Admin, "20 Benefits of Studying Music," CMS, Jun. 01, 2020. google.com/url?q=https://cmslv.org/20-benefits-of-studying-music/&sa=D&source=docs&ust=1695755337396970&usg=AOvVaw1sU9WE3Kc6gOk49h0wiB09.

29 "The Power of Music Education," KAUFMAN Music Center. https://www.kaufmanmusiccenter.org/the-power-of-music-education/#:~:text=Research%20 shows%20that%20 music%20training%20 boosts%20IQ%2C%20focus%20and%20

persistence.&text=The%20value%20of%20incorporating%20music,cognitive%20function%20and%20academic%20performance.
30 Electronics Tutorials, “Full Wave Rectifier and Bridge Rectifier Theory,” Basic Electronics Tutorials, Feb. 24, 2018. https://www.electronics-tutorials.ws/diode/diode_6.html
31 “What is the Difference Between, SPST, SPDT and DPDT?,” Moniteur Devices. https://moniteurdevices.com/knowledgebase/knowledgebase/what-is-the-difference-between-spst-spdt-and-dpdt/#:~:text=One%20of%20the%20most%20common.
32 D. Patel, Ed., “How to Access Negative Voltage Power Supply,” Maker Pro, Jan. 07, 2020. https://maker.pro/custom/tutorial/how-to-access-negative-voltage-power-supply .
33 J. Smith, “Create Your Own Battery Backup Power Supplies,” All About Circuits, Summer 02, 20216. https://www.allaboutcircuits.com/projects/battery-backup-power-supplies/.
34 “Amazon.com: ADRESUNO WS2811 IC WS2812B LED Strip Individually Addressable Light 144Pixels/m 144Pixels SMD 5050 RGB Pixel Strip DC5V (3.2FT 144LEDS Non-Waterproof, White PCB) : Home & Kitchen,” www.amazon.com. https://www.amazon.com/dp/B09PBJ92FV/ref=twister_B09SPJKCD5?_encoding=UTF8&th=1
35 DigiKey, “LCD 1602 2x16 Blue-White,” DigiKey. https://www.digikey.com/en/products/detail/universal-solder-electronics-ltd/LCD%25201602%25202X16%2520BLUE-WHITE/16821383?utm_adgroup=&utm_source=google&utm_medium=cpc&utm_campaign=PMax%20Shopping_Product_Medium%20ROAS%20Categories&utm_term=&utm_content=&gclid=Cj0KCQjw9rSoBhCiARIsAFOiplmSSASd1PzE-sN7oiu7900AGNJCsFJ1AfdjLJP4WvC7S0Lgj9G80WcaAi0 .
36 “LiquidCrystal - Arduino Reference,” www.arduino.cc. https://www.arduino.cc/reference/en/libraries/liquidcrystal/
37 “Amazon.com: XENITE Potentiometer 5PCS WH148 Single Linked Potentiometer with Cap B1K 2K 5K 10K 20K 50K 100K 500K 1M Handle Length 15mm Adjustable Resistance Electronic Component Potentiometer (Size : 10K Ohm) : Industrial & Scientific,” www.amazon.com. https://www.amazon.com/XENITE-Potentiometer-Adjustable-Resistance-Electronic/dp/B0C73Q4B7X.
38 JCPB1 A. Industries, “ADS1115 16-Bit ADC - 4 Channel with Programmable Gain Amplifier,” *www.adafruit.com*. https://www.adafruit.com/product/1085
39 Raspberry pi 4 model B - raspberry pi | digikey - digi-key electronics, https://www.digikey.com/en/product-highlight/r/raspberry-pi/raspberry-pi-4-model-b.
40 “ATMega328 - microcontroller - bootloader UNO,” *Arduino Online Shop*. https://store-usa.arduino.cc/products/atmega328-microcontroller-bootloader-uno.
41 “MIDI Basics.” Available: https://people.carleton.edu/~jellinge/m208w14/pdf/02MIDIBasics_doc.pdf
42 “Arduino Due,” *Arduino Online Shop*. https://store-usa.arduino.cc/products/arduino-due
43 “Fourier Transforms,” *blog.mbedded.ninja*, Jun. 02, 2018. https://blog.mbedded.ninja/programming/signal-processing/fourier-transforms/
44 “The Analog to Digital Converter - An Introductory Tutorial.,” *Circuit Crush*, Feb. 27, 2020. https://www.circuitcrush.com/analog-digital-converters-tutorial/
45 “MUSC 101 Unit 1 Sound Basics,” *people.carleton.edu*.

https://people.carleton.edu/~jellinge/m101s12/Pages/01/01SoundBasics.html#:~:text=The%20fundamental%20frequencies%20for%20the

46 Kubendran, R 2022, *Experiment #3 - Active Filters and Oscillators*, Lecture Notes, University of Pittsburgh

47 Calculating gain for audio amplifiers (rev. A) - texas instruments india, https://www.ti.com/lit/an/sloa105a/sloa105a.pdf?ts=1635839791291&ref_url=https%253A%252F%252Fwww.google.com%252F.

48 A. Industries, "Adafruit Mono 2.5W Class D Audio Amplifier - PAM8302," *www.adafruit.com*. https://www.adafruit.com/product/2130?gad_source=1&gclid=Cj0KCQjwvL-oBhCxARIsAHkOiu30lsNqoxLcuUM87FjKWLfuNnakNgpKsKCO8oWoOkFgSkiaGs5gnPcaAhaXEALw_wcB (accessed Sep. 27, 2023).

49 M. Tyagi, "HOG(Histogram of Oriented Gradients)," *Medium*, Jul. 24, 2021. https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f

50 "Image classification using Support Vector Machine (SVM) in Python," *GeeksforGeeks*, Mar. 21, 2023. https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/

51 Mithi, "Vehicle Detection with HOG and Linear SVM," *Medium*, Apr. 19, 2020. https://medium.com/@mithi/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a

52 "OpenCV: Harris Corner Detection," *docs.opencv.org*. https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

53 H. Zhao, "Battery Backup Circuit: A Comprehensive Guide in Making One," Circuit Board Fabrication and PCB Assembly Turnkey Services - WellPCB, Dec. 16, 2021. https://www.wellpcb.com/battery-backup-circuit.html.

54 A. Sharma, "battery charging time calculation» Freak Engineer," Freak Engineer, Jul. 22, 2021. https://freakengineer.com/charging-time /

55 "Average Cost of Private Music Lessons | 2021 Complete Guide," *Ensemble Music*, Mar. 15, 2021. https://www.ensembleschools.com/blog/lessons/cost-of-private-music-lessons/

56 Piano Lessons On The Web, "How to Read Sheet Music in One Easy Lesson," *YouTube*. Sep. 16, 2016. Available: https://www.youtube.com/watch?v=wJSQ9t0nG3Q

57 "How To Read Sheet Music: A Step-by-Step Guide," *Musicnotes*, Apr. 11, 2014. https://www.musicnotes.com/blog/how-to-read-sheet-music/

58 C. Riener and D. Willingham, "The Myth of Learning Styles," *Change: The Magazine of Higher Learning*, vol. 42, no. 5, pp. 32–35, Aug. 2010, doi: https://doi.org/10.1080/00091383.2010.503139.

59 "Experiment 3: Active Filters and Oscillators," in *ECE 1212: Circuit Design Lab*, Sep. 26, 2023

60 "OpenCV: Feature Matching," *docs.opencv.org*. https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

61 "OpenCV: cv::KeyPoint Class Reference," *docs.opencv.org*. https://docs.opencv.org/3.4/d2/d29/classcv_1_1KeyPoint.html

62 "OpenCV: Template Matching," *Opencv.org*, 2022. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

63 M. Kapadnis, "Feature Matching," *K. R. I. S. S*, Jun. 01, 2020. https://medium.com/k-r-i-s-s/feature-matching-11a912f44318

64 C. Pratt, “cal-pratt/SheetVision,” *GitHub*, Sep. 23, 2023. https://github.com/cal-pratt/SheetVision.
65 D. Tyagi, “Introduction to SIFT(Scale Invariant Feature Transform),” *Medium*, Apr. 07, 2020. https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40
66 K. S, “Object Detection: Multi-Template Matching,” Quantrium.ai, Jul. 28, 2021. https://medium.com/quantrium-tech/object-detection-multi-template-matching-2c9c9fc1a867
67 “2.3. Clustering — scikit-learn 0.20.3 documentation,” Scikit-learn.org, 2010. https://scikit-learn.org/stable/modules/clustering.html
68 Z. Peterson, “PCB Trace Width vs. Current Table for High Power Designs,” *Altium*, Nov. 27, 2023. https://resources.altium.com/p/pcb-trace-width-vs-current-table-high-voltage-design
69 J. Twomey, “The essentials of AC power safety,” *Electronic Design*, Mar. 08, 2017. [Online]. Available: https://www.electronicdesign.com/technologies/power/article/21802250/the-essentials-of-ac-power-safety
70 University of Washington, “Lithium-Ion Battery Safety,” *Environmental Health & Safety*, Nov. 2021.
71 “Nyquist frequency,” *www.st-andrews.ac.uk*. https://www.st-andrews.ac.uk/~wjh/dataview/tutorials/nyquist.html
72 “TL074 | Buy TI Parts | TI.com,” *www.ti.com*. https://www.ti.com/product/TL074/part-details/TL074CN.
73 “RCArduino: Arduino Due DDS - Part 1 - Sinewaves and Fixed Point Maths,” *RCArduino*, Dec. 01, 2012. https://rcarduino.blogspot.com/2012/12/arduino-due-dds-part-1-sinewaves-and.html.
74 “Digital Audio Basics: Audio Sample Rate and Bit Depth,” *www.izotope.com*.
75 “Fixed-Point vs. Floating-Point Digital Signal Processing,” *Analog.com*, Dec. 02, 2015. https://www.analog.com/en/technical-articles/fixedpoint-vs-floatingpoint-dsp.html#:~:text=The%20term%20
76 “Impedance Matching of Audio Components,” *hyperphysics.phy-astr.gsu.edu*. http://hyperphysics.phy-astr.gsu.edu/hbase/Audio/imped.html
77 “Raspberry Pi Documentation - Remote Access,” *www.raspberrypi.com*. https://www.raspberrypi.com/documentation/computers/remote-access.html
78 “Five Ways to Run a Program On Your Raspberry Pi At Startup,” *Dexter Industries*, 2015. https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/
79 “How can you speed up and secure microcontroller code?,” *www.linkedin.com*. https://www.linkedin.com/advice/0/how-can-you-speed-up-secure-microcontroller.
80 S. Leitner, H. Wang, and S. Tragoudas, “Design Techniques for Direct Digital Synthesis Circuits with Improved Frequency Accuracy Over Wide Frequency Ranges,” *Journal of Circuits, Systems and Computers*, vol. 26, no. 02, p. 1750035, Nov. 2016, doi: https://doi.org/10.1142/s0218126617500359. https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html#:~:text=48%20kHz%20is%20another%20common.