AnywheRe Piano

Anisha Nilakantan, Caroline Liu, and Lee Poirier

Department of Electrical and Computer Engineering, Carnegie Mellon University

Abstract **—**A system capable of allowing a user to play music notes on an AR piano in any location for the purpose of music composition and arrangement. Includes pressure for volume sensing, computer vision detection for piano keys, phone for camera, and desktop computer for user interface.

Index Terms**—** Arduino, Android, arrangement, chord, Chromatic Tuner, composition, computer vision, data structure, decibel, Gantt chart, gloves, iPhone, Kivy, microcontroller, MIDI, piano, portability, pressure-sensors, sensor, synthesizer,

I. INTRODUCTION

Our project is centered around portability for music composition and/or arrangement. Arranging music is taking an existing song and writing sheet music for it by ear for different voice parts or instruments, while music composition is writing original pieces. Both of these uses similar software, such as Musescore (a free, accessible software commonly used by arrangers), to create sheet music.

Music composition and arrangement is significantly easier when you have a piano to play chords on. At Carnegie Mellon, arrangement is often done by people not in the College of Fine Arts, such as for a cappella or Greek Sing, and thus don't have easy access to pianos. Although lightweight keyboards do exist, they are bulky to carry to and from campus. Virtual piano sites only allow users to play one note at a time, making it hard to hear a chord, and both typically only play notes at one volume. On a real piano, users can play each note in a chord at different volumes depending on how hard you are pressing each key, and quickly go up and down in volume.

Our goal is to solve many of these challenges for students, focusing on making a portable device that allows users to hear chords with overlapping notes simultaneously, and play notes at different volumes without having to manually change settings. Our product is an augmented reality (AR) piano that can be played anywhere for accessibility to all students. The user plays the piano on a paper printout of a piano layout while a phone app and a glove of sensors are used to help the computer analyze what notes the user is playing at what volume.

Through our device, the user can create chords at different volumes and octaves while offering a portable advantage and offer a more realistic piano sound than current virtual pianos. This allows students and young professionals to arrange and compose on-the-go and ease the arranging process.

II. USE-CASE REQUIREMENTS

The user seeks to have a portable, affordable piano with good quality sound that acts like a real-piano in that there is pressure-based volume control of the keys. The user carries around our product with them, so it must remain lightweight and be able to fit into their backpack. We have assumed our main user to be a typical student at Carnegie Mellon who has access to their phone and laptop; as such, these devices are utilized so that the user needs to carry around minimal additional hardware. Most people already bring their phones and laptops everywhere, so relying on the technology they already use makes our product even more portable. Achieving affordability is much easier too since the design is simpler and more optimized.

The user must be able to use the portable piano for at least three hours consecutively; this is how long it typically takes to make a basic arrangement of a song and frame the chord progressions in an outline. Further, the product must allow maximum freedom of movement to play the piano. Therefore, any gloves must be wireless.

The user must have access to at least a 49-key range; this covers most notes that would be sung by a human, allowing the user to arrange and compose for all voice parts in a single session. They must be able to hear each note as they play it on the piano. It takes a piano player, sitting around three feet from a. piano, around 2.7 milliseconds to hear a note once they play it. Given that the device is wireless and requires time to output sound, we feel that hearing sound in less than 100 milliseconds would be acceptable, as it would feel instantaneous to the user with no perceived delay.

The notes they play should output with 98% accuracy of what they expect. Since users may themselves also make mistakes on what note they play, we feel that 98% accuracy can be reasonably expected from our system.

The glove must be volume-controlled by the pressure the user plays the key. At least three discrete volumes must be heard: soft, medium, and loud, so that there is a perceivable difference between the levels.

The user must be able to play chords, and hear all notes in the chords be played simultaneously. Since a chord in arranging consists of three to four notes, the user must be able to play and hear four notes at once, at least. Notes with different volumes within a chord should be played at their individual volume, so that the user can replicate louder or softer sections contributing to that note in the chord.

The user would also like to be able to see what notes they are playing on the screen and what volume they are pressing at by having the note color-coded. Each note has a distinct color on the screen so they can visually make sure they are playing the right note. Further, the shade of the color is adjusted to reflect the volume the note is being played at. This way, they can visually identify if the output was at the right volume for what they intended. For example, a C might be orange. If the user plays the note loudly, it is a dark orange; else, a lighter orange.

III. Architecture and/or Principle of Operation

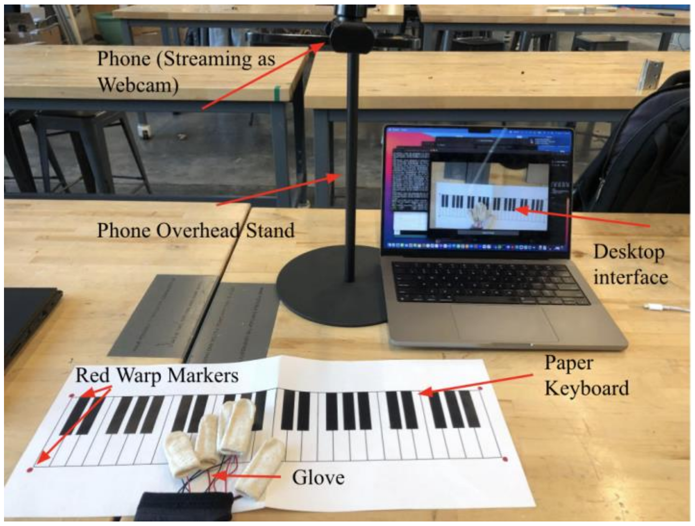


Figure 1:Overall Physical System

Our solution comprises of a set of gloves with pressure sensors on the fingertips, a paper keyboard, and a ring light with a phone mount. By creating portable gloves and a paper keyboard, we minimize the weight of the system. We use a phone and laptop as these devices are usually carried by people arranging or composing, as the software they use is on their laptop. We assume that the user is playing on a flat, horizontal surface, with no obstructing items that block the paper keyboard from sight of the phone's camera.

On the computer is an app that detects what key the gloves are trying to play on the paper keyboard, and play the note out of the phone's speakers at the appropriate volume.

Figure 3 describes the physical components of our system. The green shows materials that the subsystems are made of, and the blue shows the technological components that is integrated on our device.

On a high level, each glove is made up five pressure sensors (one on each finger), that connect to a microcontroller which is powered by batteries. Using Bluetooth Low Energy (BLE) communication, the microcontroller sends data about which finger, on which hand, is attempting to play a key, and how loud it is trying to play it. Figure 2 shows the glove, without the attached battery pack. BLE allows the user to get fast enough transmission rates (data being sent from the glove to the laptop at least every 10 ms), while requiring low enough power that an attached battery pack is able to power the glove wirelessly.

The phone is mounted to a ring light stand such that the user can see the interface of the app while the phone camera can see the paper keyboard. The stand is essential so that the camera can get a good angle that views the entirety of the paper keyboard. It streams a video feed of the gloves on the paper keyboard. The paper keyboard itself comprises of two pieces of paper taped together; this is needed to print out keys that are the same size as a real piano. The paper keyboard has four octaves, and each corner of the entire keyboard layout has red dots to help identify its location.

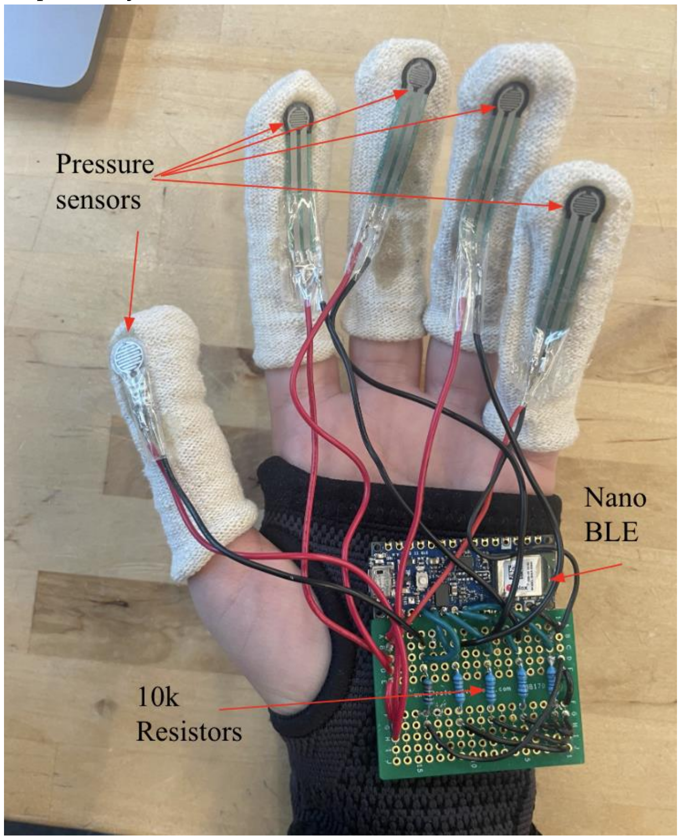


Figure 2: Gloves Subsystem

Essentially, the gloves send information about how loud to play a key, and the CV tells the computer what pitch to play.

Figure 4 explores the software structure of the app. The diagram here depicts the flow of the app on the computer; Kivy can be launched natively and directly interact with the speaker and BLE transmission. The app itself walks the user through the stages of connecting the gloves (left and/or right), and then calibrating their piano. We assume that the user does not move the paper piano, but if they do, they can recalibrate by returning to the calibration screen at any point.

First, the gloves send over the pressure information of any finger that is pressed down. This information is received by the app where CV processing occurs to determine which key was pressed (using information from the paper keyboard). Putting together these pieces of information, the piano sound is rendered by the computer, which is outputted through the computer's speakers.

Our design has changed slightly from our design report. Firstly, our app was initially going to run on our phone. However, in order to get a good angle on the paper piano to perform the warp, we opted to use an overhead stand for the phone. This means that the user can no longer really see the phone interface while it is using the camera. Therefore, we moved our interface to be on the laptop. This way, the user can see both the visualizer and their hands playing the piano, while sitting down comfortably like they would at a real piano.

Changing our app into the desktop interface allowed us to get rid of any need for Xcode, so we wrote our whole app in Python using Kivy language. It required using the Bleak package to connect to the BLE from the app itself on the laptop. With these changes and mounting our phone overhead, we also are now able to design a system that can play all 4 octaves at once, since we can get them in the frame of our camera view. Previously, we were limited to playing at most 2 octaves at a single time, due to the camera view and the limited space to visualize the keys on the phone interface.

Further, our last design report played a heavy emphasis on audio processing and attempting to write MIDI files. We removed this as our audio processing now uses WAV files to play notes, using an inbuilt mixer (via Pygame) to synthesize sounds. This let us focus on the timing component of integrating our pressure input with the CV hand positions that is integral to our use case. We did keep the element of MIDI files as an additional feature. Users can choose to set a tempo and write to a MIDI file as they play, and then the final file is saved onto their desktop can be opened in apps like Musescore. This is a nice feature that allows users to save their chords—however, it is no longer a focus in our use case or design requirements.

IV. Design Requirements

The gloves (pressure sensors & micro-controller) and phone holder is be at most 2 lbs in total. The total cost of creating the gloves (pressure sensors & micro-controller) and the ring light with phone mount is at most $100 in total. This keeps the device portable and affordable. Though a typical portable 49-key piano might be cheaper at around $60, the quality of sound is poor. Portable pianos of better sound quality cost at least $130, making our option cheaper than the current market option.

To have an interface on the laptop, allowing the user to see what keys they are playing and how loud, our design needs to be made on a platform that can be launched from any laptop. Therefore, we used Kivy, a cross-platform app development tool that runs our interface in Python. This also allows us to implement computer vision in a language we are already familiar with (Python). In order to view the paper piano, another camera is required, held up on a stand. To minimize cost to the user, we are using their phone as a webcam to their computer. This allows users with both Androids and iPhones to use the interface; on iPhone, for testing purposes, we are using the Continuity Camera on iOS16 and macOS Ventura to use our phones as a webcam, requiring no additional set-up.

The note playback time from when the user hits the key and hears it must be at most 100ms. This can be achieved through the use of a BLE microcontroller (Arduino Nano BLE) connected to the computer. For the duration of the BLE information being transferred, the data is sent at least every 10ms and each thread to receive the data is checked at 30Hz, which is how often data about where hands/fingers are on the piano is updated. This allows around 40 milliseconds of slack time, in case of concurrency blocks for other processes going on such as rendering the visualizer of the piano on the interface.

The piano keyboard layout has red dots which serve as colored markers for our piano with 4 octaves (49-keys). This helps the CV algorithm identify where the printed piano keyboard layout is in the video stream. 49 keys were chosen as this is typically how many keys are on portable pianos that cost about $100.

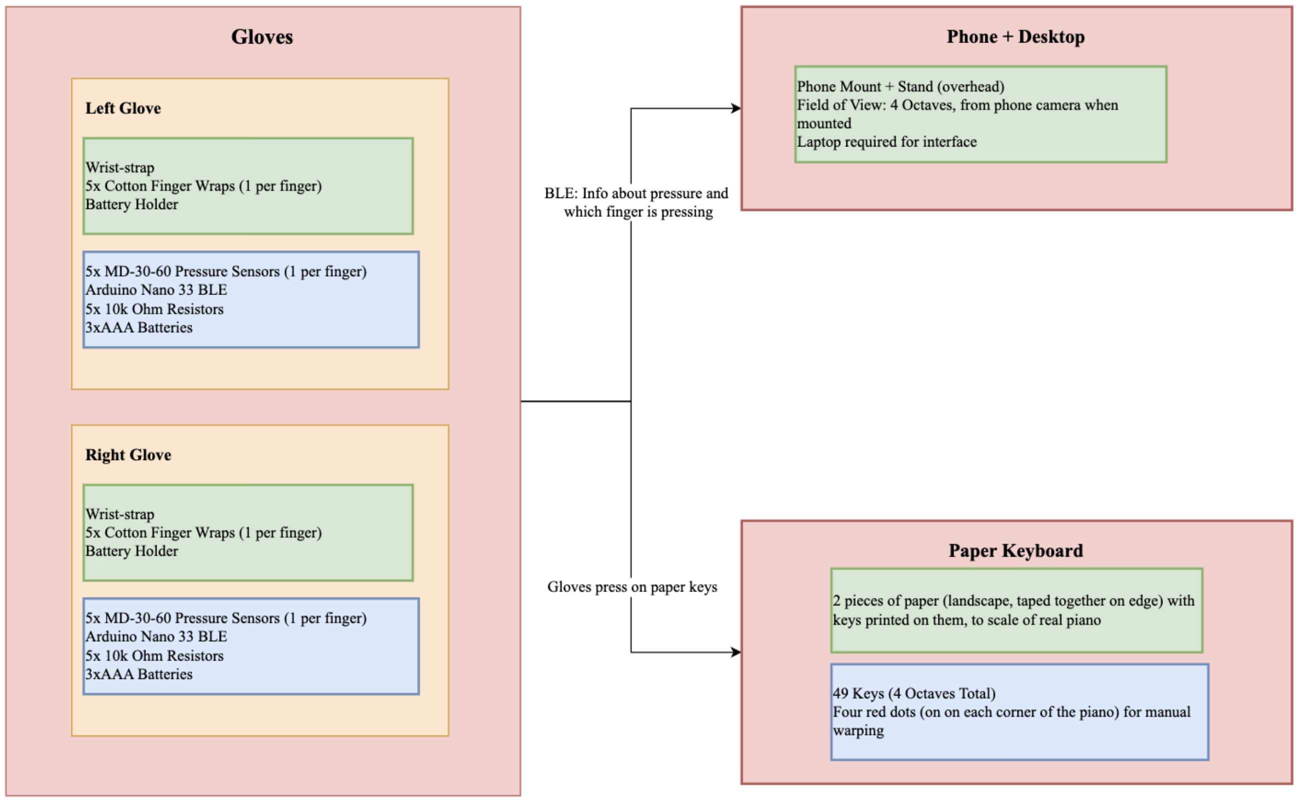

Gloves
Left Glove
Wrist-strap
5x Cotton Finger Wraps (1 per finger)
Battery Holder
5x MD-30-60 Pressure Sensors (1 per finger)
Arduino Nano 33 BLE
5x 10k Ohm Resistors
3xAAA Batteries
Right Glove
Wrist-strap
5x Cotton Finger Wraps (1 per finger)
Battery Holder
5x MD-30-60 Pressure Sensors (1 per finger)
Arduino Nano 33 BLE
5x 10k Ohm Resistors
3xAAA Batteries
BLE: Info about pressure and which finger is pressing
Gloves press on paper keys
Phone + Desktop
Phone Mount + Stand (overhead)
Field of View: 4 Octaves, from phone camera when mounted
Laptop required for interface
Paper Keyboard
2 pieces of paper (landscape, taped together on edge) with keys printed on them, to scale of real piano
49 Keys (4 Octaves Total)
Four red dots (on on each corner of the piano) for manual warping

Figure 3: A Block Diagram depicting the Physical Structures (green) and Technological Components (blue) of 3 Interfacing Subsystems (Pink)

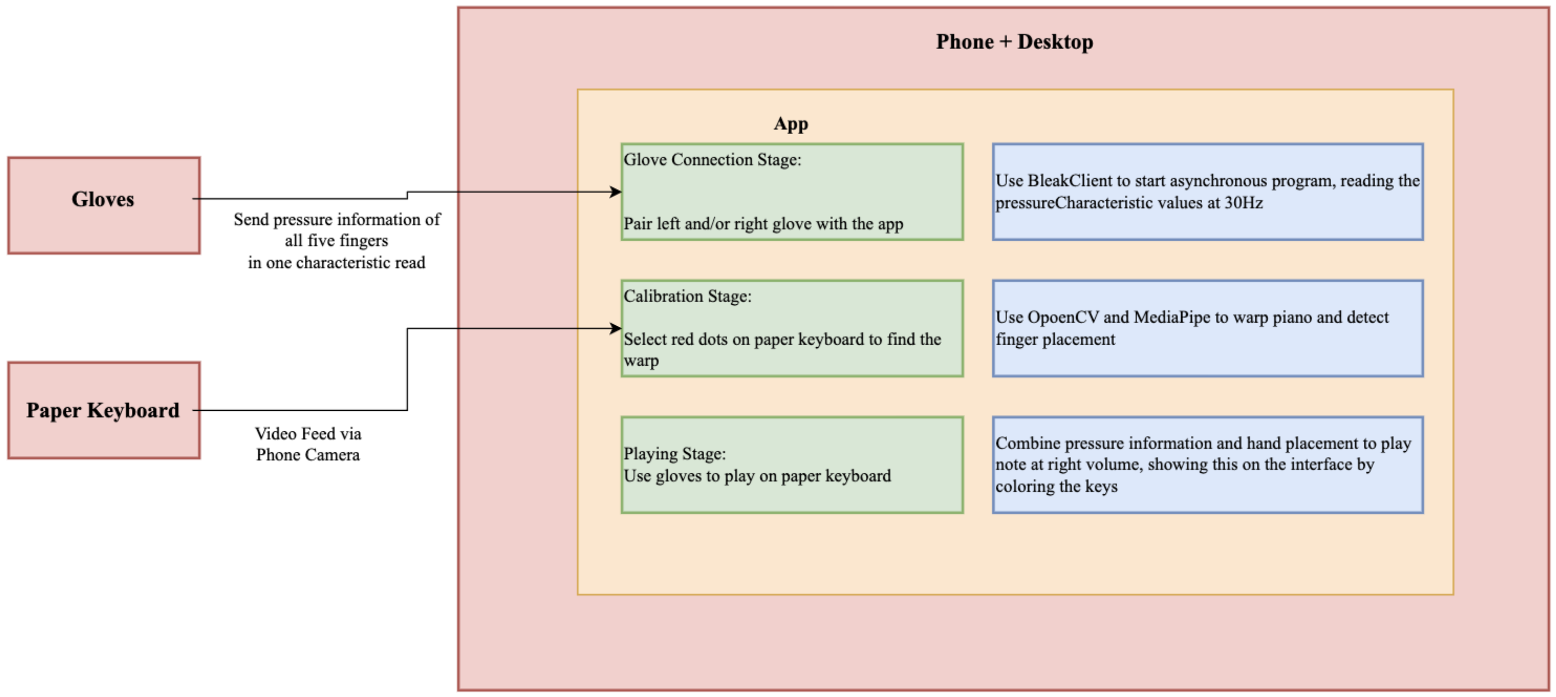

Phone + Desktop
App
Gloves
Send pressure information of all five fingers in one characteristic read
Paper Keyboard
Video Feed via Phone Camera
Glove Connection Stage:
Pair left and/or right glove with the app
Use BleakClient to start asynchronous program, reading the pressureCharacteristic values at 30Hz
Calibration Stage:
Select red dots on paper keyboard to find the warp
Use OpoenCV and MediaPipe to warp piano and detect finger placement
Playing Stage:
Use gloves to play on paper keyboard
Combine pressure information and hand placement to play note at right volume, showing this on the interface by coloring the keys

Figure 4: Software Block Diagram

Different volume levels are determined if pressure sensor output voltage is within three distinct intervals which can only be fully determined through testing. We are also be able to tell which finger is pressing at what volume, since each of the 10 pressure sensors on each finger are unique to each other when they send information to the microcontroller.

Gloves need to be calibrated when beginning to play. This means that if the gloves are faced in the upright orientation where the sensors might be pressed by the weight of the glove, it should instead be reset to detect nothing when there are no external forces on it. These pressure sensors are less than the size of your finger which is less than 14mm. This allows the user to have a comfortable experience wearing the gloves, and let them play the piano as naturally as possible.

On the software side, it is programmed to have an array that stores the value of which note is being played simultaneously. Software is programmed to store all 49 total keys which encompasses all 4 octaves in an array data structure. The CV algorithm must be able to detect the note in under 40 ms. To summarize our timing requirement, we can analyze the worst-case scenario. If CV takes 40ms to find the location of the finger, and it takes 33 ms to read the newest data from the BLE, it takes 77 ms to output the note. This is under 100 ms. We do not consider the 10ms it takes for the BLE to send over the information, as our thread checks whatever is currently in the BLE at the point of checking. If it only checks the BLE at the end of the 40 ms it takes for the CV to run, it adds on the full 33 ms, so 77 ms is our worst-case scenario.

Computer vision edge detection must detect key locations and outlines 100% of the time. Computer vision finger detection must detect the location of fingers based on key detection outlined above 98% of the time. We want to be as accurate as possible, but as the piano is flat, there may be edge cases where the algorithm cannot tell which key the finger is distinctly on.

Finger detection accounts for all 10 total fingers to analyze. To calibrate the computer vision, the phone mount holder must be a top down 90 degrees from the horizontal in order to view the entire piano layout.

The battery must have at least 3.3V, with enough current to power the Arduino Nano BLE and the pressure sensors. We want our battery to last at least three hours, which is an approximation of how long it takes to arrange a song. The Design Trade Studies will further explore the battery requirements.

V. Design Trade Studies

We encountered several trade-offs in our design, especially regarding picking our microcontroller + BLE modules, power supply, how to write our interface, and how to create our gloves and identify the user's fingers.

A. *Microcontroller + BLE Module*

We considered several different microcontrollers and BLE modules. We want our glove to be as lightweight as possible, and minimize the bulk carried on a user so as to increase portability and keep the weight of the entire system under 1 pound. We also wanted to use BLE to transmit data, as regular Bluetooth modules are more difficult to pair with iPhones which would limit the cross-platform independence outline in order design requirements. At the same the time, the microcontroller only needs to sense the pressure, threshold the value and encode it, and transmit, so familiarity with the microcontroller and module would be beneficial as its programming is not the main focus of our system.

Initially, we wanted to use the STM32F103 with an HC-05 Bluetooth module due to Lee and Nish's prior experience with STM32; however, we discovered that HC-05 is in fact low-power rather than actual BLE. Thus, we analyzed three different options, as in the table below (Table I):

TABLE I. MICROCONTROLLER ANALYSIS

	Microcontroller + BLE		
	STM32WB[7]	***nRF52840[6]***	***Arduino Nano 33 BLE [4]***
Size	7 x 11.3 mm	7 x7 mm	18 x 45 mm
Processor	ARM Cortex M4	Arm Cortex M4	Arm Cortex M4
Power	3.6V	1.7V	3.3V
Experience	With STM32	None	With Arduino
Weight	7.2 g	0.6 g	5 g

The Arduino Nano 33 BLE is much larger than the other two options. In fact, the Nano has the nRF52840 on its board. However, it requires less power than the STM32WB, and all three members of the group have worked with Arduino before. Furthermore, though we have experience with the STM32, the setup and debugging process seemed to far exceed what we were using the microcontroller for. Since we are not implementing threading on the microcontroller, a simple Arduino would suffice. We also wouldn't need to build a Printed Circuit Board for I/O with the pressure sensors, whereas if we used the STM32WB or nRF52840, we would need to account for the size of the PCB and test it.

To account for the size of the Nano, we placed the Nano horizontally, so that it fits within the user's wrist, still. Now, we will analyze how to power the Nano and pressure sensors.

B. Batteries

We considered coin cell batteries (CR2032), AAA, and AA batteries to power our system. We wanted to make sure our product lasts at least 3 hours.

The Nano and pressure sensors each require 3.3 V to operate. CR2032 can supply these 3.3 Volts with only one cell – however, it operates at around 235 mAh, assuming that you only draw around 15 mA at a time (optimal conditions). We first calculate how long each glove would run using a CR2023 cell.

To play loudly on a Kawai (a brand known for heavy keys) requires about 40 N, while a soft note is about 4 N. On a paper piano, we don't need to flip up the hammer of the piano, so we can scale our "loud" "medium" and "soft" to average about 10 N for medium notes [5]. The following equations calculate battery life when playing medium loudness notes.

The force sensitive resistor draws 0.45 mA when a 10N force is exerted upon it [8]. Since we have scoped our design to be able to play at least four notes at once (a full chord or tetrad), we can see in Equation (1) that we need at least 1.8 mA/chord.

$$0.45\,mA * 4\,notes = 1.8\,\frac{mA}{chord} \tag{1}$$

However, each I/0 pin on the Arduino Nano requires 15mA, which means that our system needs 75 mA per hand for at least 3 hours before needing to change batteries. A typical coin battery has 235mAh, AAA batteries have an mAh of 1200, while AA batteries have 2500 mAh. From equation (2) we calculated that coin batteries can last up to 3.1 hours, while AAA batteries can last up to 16 hours and AA batteries can last up to 33 hours given the amount of current needed to power our system.

$$\frac{235mAh}{75mA} = 3.13\,hours \tag{2}$$

$$\frac{1200mAh}{75mA} = 16\,hours \tag{3}$$

$$\frac{2500mAh}{75mA} = 33\,hours \tag{4}$$

It would not be feasible to have coin cell batteries that need to be replaced every 3 hours, entirely. While AA batteries hold more charge and can last longer than AAA batteries, AA batteries are also heavier and bulkier. AA batteries weigh 0.8 ounces each and AAA batteries only weigh 0.3 Oz each. Since our requirement is that we should be able to compose at least one song (~15 hours), we decided that the extra 17 hours that the AA batteries could give us would not be crucial.

C. App Interface

We went back-and-forth on what to build our app interface with. Initially, we wanted to launch the app from an iPhone, but code as much of our interface in Python as we are familiar with OpenCV in Python. However, iPhone apps need to be written in Xcode using Swift and Objective-C (languages none of us are familiar with); therefore, we planned to build most of our app in Kivy, a Python cross-platform interface, and use the Kivy-ios library to wrap this app in Xcode and launch it from our phones.

However, we found out that Kivy-ios is limited. Although apps can be launched from the iPhone, the wrapping prevents Kivy from touching iPhone hardware such as the camera, speaker, and BLE transmission access. These three components were integral to our app, so we started learning Swift and writing our app.

At the same time, we also decided to mount our phone from the overhead angle to get a better angle on our paper piano. This would render our phone app useless as the user wouldn't be able to see the app. Therefore, we moved our entire platform to the Desktop/Laptop, and were able to code in Kivy and access our camera, speakers, BLE, and write our OpenCV in Python. Previously, we would have had to integrate a Python interpreter into our Xcode app, which would slow it down considerably, or rewrite OpenCV in C++. Although it came it a cost of rewriting our interface, we believe we saved a lot of time in integration of BLE and OpenCV

D. Gloves

For the gloves, we decided to use the cotton finger wraps instead of the latex finger cots because we want our design and parts to be reusable after every use. This would reduce the amount of waste we would produce and also would feel better on the user's fingers.

There were also complaints about how the latex finger cots would be really tight against the user's fingers and were uncomfortable for long periods of time. Since we expect our user to be using the cots for multiple hours, we would want to have as comfortable as a device as possible.

We also considered various methods to create a frame for the glove so that the circuit would have a place to sit firmly and securely. Dish gloves had the strong material we were looking for but they never came fit to size and so we wouldn't want the fingers at the end not snug when you would be pressing on the pressure sensors. We considered using latex gloves again, but did not use them for the same reasons as above. We then considered 3D printing out own gloves with a plastic material as durable as the dish gloves but would also be snug against fingers and palms of the hands. This didn't work out as 3D printing was too risky to try out especially this late in the work schedule.

We lastly landed on purchasing lifting gloves, which essentially cover a portion of the hand, mainly the palms. This would provide a place to glue in the microcontroller and batteries. The lifting gloves also had a Velcro strap that helped the user tighten the glove so that it would be a better fit. Lastly, it did not interfere with the use of the cotton finger wraps where the pressure sensors would be located.

E. *68k Ohm Resistor vs 10k Ohm Resistor*

Instead of using a 68k resistor for our circuit, we changed to using a 10k resistor because we found that using the 68k resistor limited the range of the pressure sensor to sense certain thresholds for various volumes. The 68k resistor led to medium touches outputting the max value on the sensors, whereas the 10k resistor allowed us to read high force pressure from the sensor and better threshold our loud, medium, and soft touches.

F. *Finger Identification*

Originally to identify distinct fingers we were thinking of putting colored fiducials on each of the fingers to conduct color thresholding and feature matching CV algorithms. We decided to change it to using an external module called Media Pipe so as not to reinvent the wheel. It is faster, more efficient, and largely more accurate to find the locations of the fingers. The tradeoff to using Media Pipe is that if too much of the fingers are covered by the gloves, then the algorithm that we rely on may have trouble discerning the fingers. This can be troublesome as the only way to debug this is in the design of the gloves since you can't modify the Media Pipe package.

G. *Camera Angle*

We wanted to attach a phone mount to our laptop since the mount would be lightweight and miniscule, but because of the distance and angle away from the paper piano, the mount would have to be at a 45-degree angle below the horizontal. We decided to change this to a ring light stand with a phone mount because it could provide a top-down 90-degree vertical view of the paper piano. This is important because it would make identifying the paper piano keyboard easier since you wouldn't have to warp the image from a stretched image (at the 45-degree angle) to a size-fitted image (top-down view).

Even though we still have to warp. It is a process that only requires zooming in and eliminates any stretching of the piano keys that may have emerged from taking the input frames at such a low angle.

The ring light was a useful addition since it would help keep lighting environments consistent which would be helpful for identifying the contours of the piano keyboard. We wouldn't want it too overexposed while not too underexposed in terms of lighting.

We also considered using a goose neck functionality of the stand with the phone mount, however, there were very few options on Amazon that offered a goose neck long enough to stretch over the piano print out. Other alternatives have longer goose necks, but were not a desk stand but rather a desk clamp. We did not want to limit our users to only being able to play on surfaces that required an edge (this edge also cannot be too thick or thin for the clamp).

H. *CV vs Accelerometers and gyroscopes*

We decided to use computer vision as the algorithm for key detection rather than using additional sensors on the fingers such as accelerometers and gyroscopes to determine position. This was because it would have made the gloves much bulkier already since it had the pressure sensors on each finger to begin with. Additionally, it would be more expensive to buy 10x the sensor since we would need one for all 10 fingers.

Another reason is that the work would have better lined up with Lee since he had more experience with computer vision than the accelerometers and gyroscopes. Lastly, even though computer vision is never "perfect", it is still easier to implement than doing the math and trigonometry to obtain the position from those sensors.

VI. System Implementation

This section explores the implementation of each main subsystem: Gloves, Paper Keyboard, App Interface, Computer Vision Processing, and Audio Processing.

A. *Gloves*

Figure 3 denotes the components of the gloves. Referring to Figure 5, we can see top and bottom views of the gloves. The pressure sensors used are MD 30 60 force resistor sensors, which are approximately ½" in diameter and fit on the pad of each finger. In the case of the thumb, the sensor is be placed on the side of the thumb, where it would most naturally strike a piano without inducing awkward posture of the hands. The sensors are connected to the Arduino Nano that sits on the back of the hand, held together by the wrist strap. The batteries currently sit on the on the underside of the wrist.

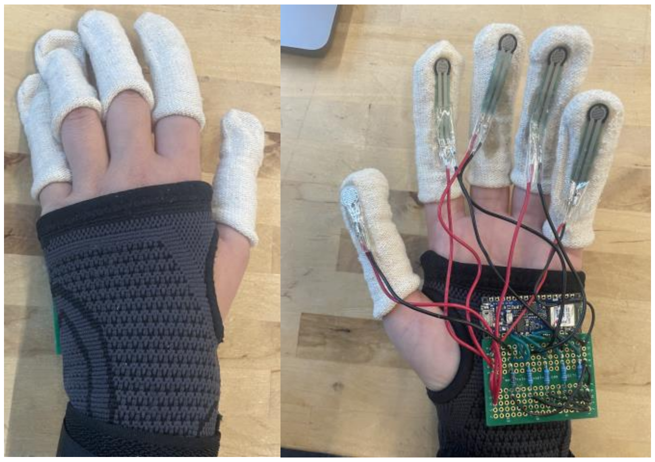


Figure 5: Top and Bottom Views of the Gloves

These sensors work on a logarithmic scale, as seen in Figure 6, so we have to carefully threshold where we want our “loud, medium, and soft” forces to lie; very low forces are difficult to distinguish from each other. The Arduino Nano reads the resistance of the pressure sensors, determining if the circuit (shown in fig 7) is open (no force, or very little force), or encode the volume depending on the read value.

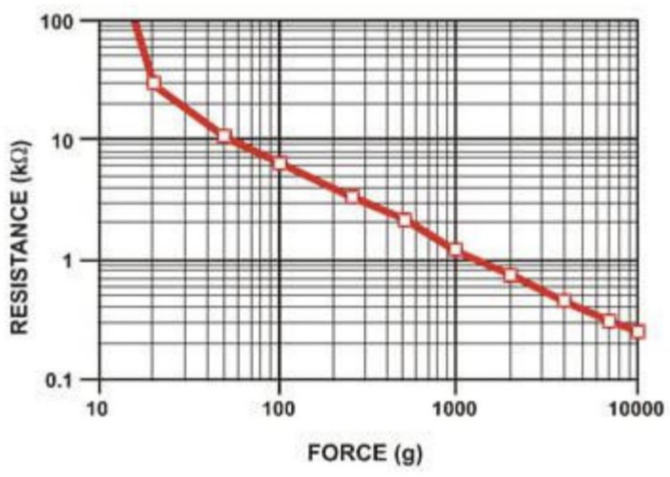


Figure 6: MD 30 60 Force vs Resistance [8]

The BLE Flow diagram (Figure 8) outlines how information is sent from each glove to the laptop. The pressure sensors output a voltage value between 0 and 5. The pressure sensors work as a variable resistor and limit the amount of voltage that is able to flow through. Having a higher pressure means that there is a higher resistance which correlates to a higher voltage read. We have thresholded these values based on testing of what constitutes a soft, medium, and hard touch. We translated these into values: 0 for off, 1 for soft, 2 for medium, 3 for hard. Each of these values was stored as a char, so that it takes up exactly one byte.

We combined all this information into an 8-byte long, to hold 5 bytes of information (1 byte per pressure sensor/finger) with 3 bytes of padding. Since each pressure sensor always sends its value to the same spot in the long, we are able to extract which finger has what value by reading the entire long.

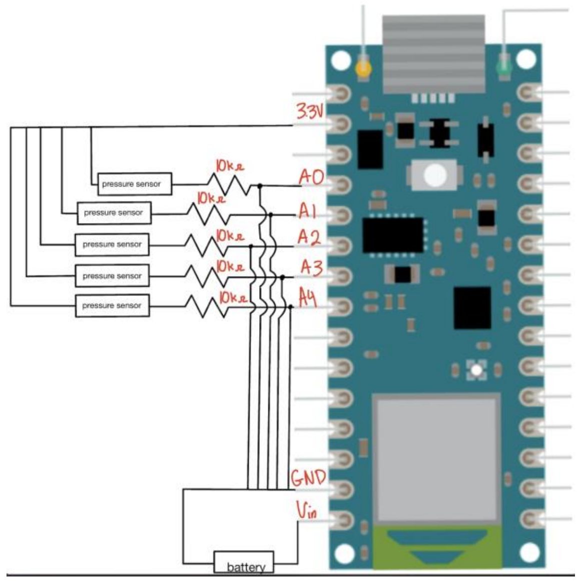
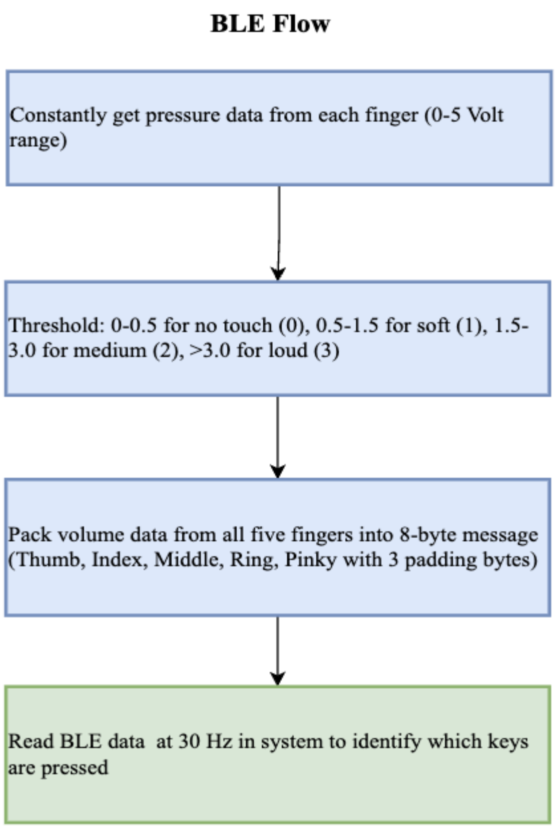


Figure 7: BLE Nano Circuit Diagram


BLE Flow
Constantly get pressure data from each finger (0-5 Volt range)
Threshold: 0-0.5 for no touch (0), 0.5-1.5 for soft (1), 1.5-3.0 for medium (2), >3.0 for loud (3)
Pack volume data from all five fingers into 8-byte message (Thumb, Index, Middle, Ring, Pinky with 3 padding bytes)
Read BLE data at 30 Hz in system to identify which keys are pressed
Figure 8: BLE Flow

BLE works by first pairing to the app. Once paired, the app searches for all the services the BLE offers; in this case, it offers one service, the pressure service, with one characteristic, the pressure characteristic. The app calls read on the characteristic, and reads the entire 8-byte value. We chose to condense information from all the sensors on a glove into one characteristic to minimize read time from the BLE. With one read, five sensors can be read.

Each glove is read 30 times per second. This is because our CV updates the hand positions at minimum 30 times per second, in order to be able to view our video stream, so we need to read our value at least as frequently. The read reportedly can take up to 30 ms in worst case scenario [4], but the two reads (one per glove) occur simultaneously.

We know now the pressure of each finger after the read, and their positions from the last CV run. Therefore, we can play the note at the right volume.

B. *Paper Keyboard*

Our 49-key paper keyboard has 4 red dots that serve as colored markers on each of the corners so that the computer vision algorithm can determine the location of the paper keyboard when computing on frames from the video stream.

C. *Interface*

Figures 9 and 10 show our interface. Figure 9 depicts the standard screen the user sees. The four octaves they are playing shows up on the screen. Keys that are pressed light up, and the name of the key appears above the keyboard so the user can verify that the app has detected and is playing the correct notes.

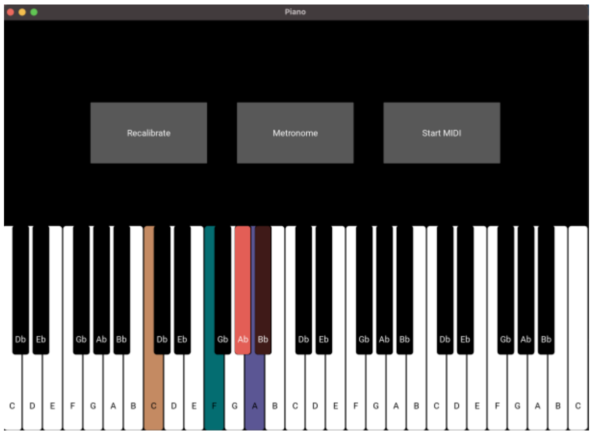

Piano
Recalibrate
Metronome
Start MIDI
Db Eb Gb Ab Bb Db Eb Gb Ab Bb Db Eb Gb Ab Bb Db Eb Gb Ab Bb
C D E F G A B C D E F G A B C D E F G A B C D E F G A B C

Figure 9: Rendering of Piano Interface

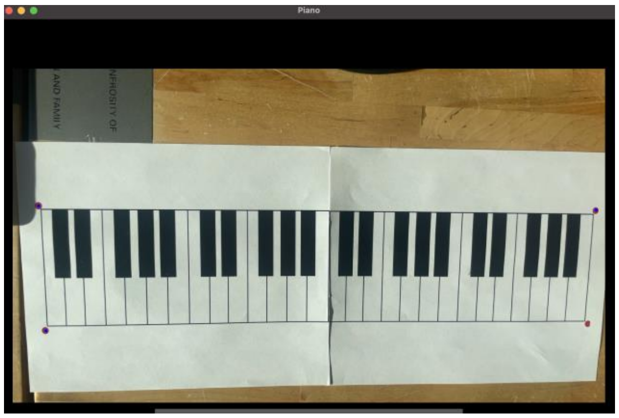


Figure 10: Rendering of Image Capture Interface

Every key has its own color, as previously mentioned. The same note in different octaves has the same color, as they are far enough apart visually to discern between them. For the same reason, black keys are all red, since they are separated by white keys so they won't be confused. This was made as having 13 different colors on the screen was reportedly too much, according to our initial user studies. Playing louder induces a deeper color.

We considered not having every key be a different color, and simply coloring the note green when soft, yellow when medium, and red when loud. However, when implemented, the users preferred the version with many different colors. Although the volume output wasn't intuitive to interpret with having different shades of the base color of the key, they enjoyed playing the piano and seeing aesthetically pleasing key colorings with a wide range on the screen. Ultimately, we found that we needed visual volume cues while testing the device, but now that the volume has been tested, the user does not need to verify how loud their note was while arranging.

At the top, of the screen, we see three buttons: calibrate, tempo, and create MIDI. Calibrate is shown in Figure 10. This is a step the user must go through every time they set up the app or shift the camera. The video feed is shown to the user. They capture and image of just the piano, with no gloves or hands covering the keys. Once the image is captured, they are able to select the 4 red dots, and the warp is rendered as they are taken to the Piano screen. The piano screen does not show the video feed, as it is not necessary for the user. The user is free to tape down the keyboard. If the keyboard shifts, they will see the notes on the interface registered incorrectly, and will be able to recalibrate the screen.

The interface also accounts for users that may only want to use one hand to play (as they might be using the other hand to compose on their laptop as they listen to the piano). The user can choose which gloves to pair in an initial screen upon launching the app. They individually pair the left and right gloves.

The app uses asynchronous functions to check the left and right gloves. Technically, the app is continually reading the values from each sensor, which may take up to 30 milliseconds in a worst-case scenario. It also checks at minimum 30 times per second, due to how we have scheduled the async task with the Kivy Clock. The Kivy Clock also schedules the CV algorithm to check hand placements of the user 30 times a second, which can be bumped up to 60 times a second. We chose to do 30 times a second initially to account for a worst-case scenario and see if we can still meet our timing requirements.

After the CV is run and we know where the user's fingers are, the latest call from the BLE reading is taken and fed to the piano. The respective PianoKey is called with the appropriate volume setting, and the sound is outputted. It is important to note that may receive multiple pressure readings for notes that are held down; this is accounted for in our software. The note is played with the volume of the first reading, and won't be played again until we see that the finger has been lifted off the paper (0 values for pressure) and is pressed again. This is because real pianos don't loop notes as someone holds down the key.

The tempo and create MIDI buttons essentially allow the user to save chords. They can set a tempo, which is necessary to create a MIDI file. From there, any time they press a note, the message is also written to a file using the Mingus MIDI library.

At the end of the session, the user can save the file to their computer that is running the app as a .mid file.

D. *Computer Vision Processing*

Figure 11 shows a flow of the CV processing that occurs in the app. Upon receiving the video input of the gloves playing the keys, the computer vision uses an external package called Media Pipe which relies on a machine learning algorithm to identify the positions of all fingers on the gloves to identify their location in space. We program it so that it only searches for fingers for which it has received pressure information via BLE.

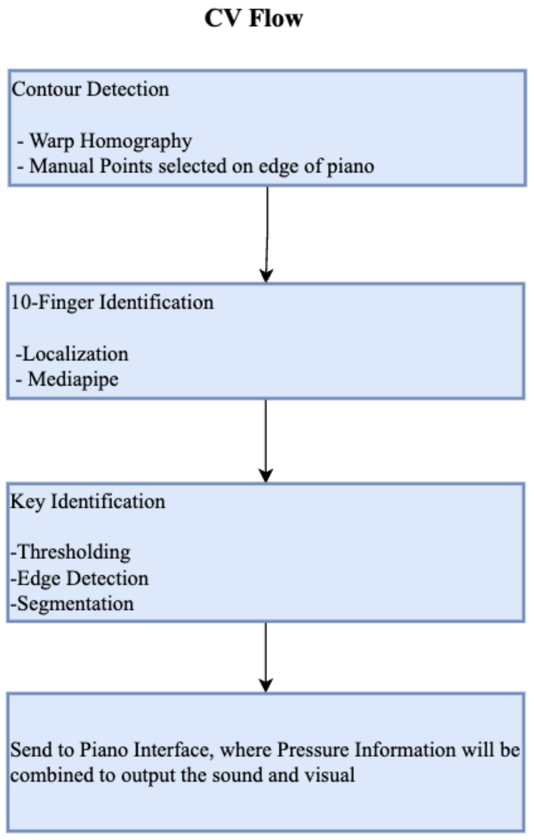


Figure 11: CV Flow

Once the finger has been located in space, the CV identifies if the finger is within the bounds of the paper keyboard.
If a finger has pressure information (it is pressing something), but the finger is not on the bounds of the keyboard, nothing will be played; this allows the user to use their hands for other activities such as typing or using a mouse, alternating between that and playing the piano.

If within the bounds of the keyboard, CV will determine which key the finger is on by using contour and edge detection to delineate between keys [1]. If the finger is located exactly between two keys, our algorithm will consistently take the higher key. At this point, the pitch each finger is playing has been identified.

The next step is to match the pitch with the pressure/volume information received from the glove. After combining this data, it then outputs the desired sound out of the computer' speaker.

VII. Test, Verification and Validation

For our testing, verification and validation, we all worked together to ensure that we didn't miss out on any edge cases/tests that might have influenced the success of our project and to test the robustness of our design requirements for our use-case.

A. *Tests for Note Accuracy*

The main purpose of our project depends on the accuracy of the notes we play. This means that this initial test is our most important test. Thus, we performed rigorous tests to test its robustness. This test involves playing each individual note separately. We tested the accuracy of each note within our 4-octave range, so any note played should be correct. We would compare each note played with a chromatic tuner.

We played all 49 keys on our piano as inputs with a goal of 98% accuracy (48/49 keys) which resulted in 100% accuracy.

B. *Tests for Chord Accuracy*

We want to ensure that our notes stay accurate when played together. Thus, on top of just note accuracy, we also tested for chord accuracy. This involves a similar test as our note accuracy tests, but involves chords instead.

We played all 49 keys on our piano as inputs with a goal of 98% accuracy which resulted in 100% accuracy again.

C. *Tests for Multi-note Volume Comparison*

Another important requirement we have is having multiple volume outputs for how hard the user presses down on a certain key. We tested volume by having five users play "loud, medium, or soft" notes (10 per note), marking what they expect vs what the actual output is. We also made sure that different notes play "loud" at the same volume, by playing successive notes at the same pressure and measuring and comparing their decibel output level, on our phone, to each other. Any errors here was fixed by mitigating our volume thresholds in our interface. We also allowed the users to have a few minutes to get used to the gloves and the pressure.

Of the 50 soft notes tested across subjects, we received 100% accuracy. Of the 50 medium notes, we achieved 88% accuracy, and of the 50 loud notes, we achieved 100% accuracy. This is results in an average accuracy of 96%, and the thresholds will continue to be tuned.

D. *Tests for Multi-note Volume Threshold*

When playing multiple notes simultaneously, we didn't want to overpower the speaker if all the notes played are "loud". We wanted to ensure that the maximum decibel value of all of the notes played together is less than 70 dB, which is a few decibels under the maximum phone volume limit. If it had exceeded this threshold, we went back to our interface and toned down the volume output for the "loud" notes until it fitted our requirement.

We played 49 "loud" notes as our inputs as inputs with a goal of 98% accuracy which resulted in 98% accuracy.

E. *Tests for Playback*

Because we are playing multiple notes at the same time, we want to have a fast-enough playback time for the notes played. First, we tested our playback speed, playing at least 8 notes. We started the time when the Arduino BLE Nano registers a pressed key, and then checked to see how long it took to reach the app and call the command to play the speaker. The goal was for all of this to happen in under 100ms. If it didn’t, we altered how our apps’ threads handle input and prioritize better.

We played 8 notes as our inputs as inputs with a goal of under 100ms which resulted in the greatest time of 61.3 ms, with an average of under 40 ms. We expected a worst-case scenario of 77 ms. We think it was under 77 ms because our BLE runs asynchronously and can be checked at any point, so the reads are taking less than 30 ms when we use the data 30 times per second, immediately following our CV process.

F. Tests for Edge Cases

With computer vision, we expected that there could be some unexpected behavior when it came to playing at the edge of a note. Thus, this test evaluates the robustness of our CV algorithm. We used one finger to play two adjacent notes, which includes pressing the edge between a black key and the neighboring white key and pressing two adjacent white keys. We wanted to ensure that only one note was played in these scenarios, so that it wouldn’t play something that the user didn’t want. Throughout our different tests, we wanted the note played to be consistent. This meant that when one finger pressed two keys, we wanted the highest note to be played. We wanted this to also have a 98% accuracy, just like our normal note accuracy test.

We played 10 sets of two adjacent keys (black/white and white/white) on our piano as inputs with a goal of 90% accuracy which resulted in 100% accuracy.

G. Tests for Typing/Using Fingers with Glove on

With music composition, we expected that our user would be using their computer while their gloves were on. Thus, we needed to ensure that when the user was not actively playing, they didn’t trigger any notes by just using their hands/fingers. We tested this by pressing the table surrounding our paper piano and seeing if that would trigger any notes. We wanted to have a 98% success rate, meaning that we wanted our system to not play a note if something was pressed in the surrounding areas and also to play a note if it was pressed on the keyboard 98% of the time. In addition, if the user’s fingers were not in sight of the camera, it should have never played a noise even if pressed.

We played all 49 keys on our piano as inputs with a goal of 98% accuracy which resulted in 100% accuracy.

H. Tests for Battery Life

We tested the battery life that powers our Arduino Nano board and we wanted our battery to be able to operate at 3.3V for at least 3 hours without replacement. In order to test this, we let the Arduino sit for three hours with the battery plugged in and transmitting BLE data. At the end of the three hours, we tried to use our glove and see if it was still powered.

Since the glove was still powered at the end of the three hours, we considered this a successful trial.

I. Tests for Warp Consistency

Whenever the user moves the printed paper piano keyboard layout, the points use to warp to the corners of the piano may be different. The user would have to calibrate their piano keyboard by choosing a new set of points and then warping again. This test is designed to validate how well the warping function works since it will be used often whenever a user needs to calibrate their piano keyboard again.

We ran the warp function under 25 test images of different lighting environments and different locations of the piano keyboard as our inputs with a goal of 95% accuracy and with a result of 100% accuracy now.

J. Tests for Key Segmentation

For the same reason as above in the warp consistency test case, every time the user calibrates their system by choosing a new set of points for their piano keyboard, the key segmentation function will also be run again. Since we can expect this to occur frequently, this test is designed to validate how well the key segmentation algorithm works.

We ran the segmentation function under the same 25 test images of different lighting environments and different locations of the piano keyboard as our inputs with a goal of 95% accuracy and with a result of 84% accuracy. The thresholding stage of segmentation is what causes this segmentation to fail. Therefore, we are working on the lighting accuracy, and if we are unable to adjust for lighting, we will offer the user a check to make sure that the thresholding stage was accurate before they try to play the piano. This and the key identification test are linked, and have the same solution.

K. Tests for Key Identification

For the same reason as the previous two test cases, this one verifies the accuracy of the next step in the computer vision process which is identifying the key from where the fingers are hovering over the piano in the video stream. This is before the gloves are incorporated with the CV subsystem.

We ran the segmentation function under the same 25 test images of different lighting environments and different locations of the piano keyboard as our inputs with a goal of 100% accuracy and with a result of 84% accuracy.

L. User Testing

We decided to add some user experience testing. We had 5 people try our entire system together, and walked them through pairing the device, calibrating the screen, and then playing the piano. The MIDI feature was not tested as it was not ready at the time of user tests.

In our first round, users tested the version with the visualizer that had each key a different color. We received some feedback that users wanted to try a version where there were only three colors the keys would change to, representative of the volume, as the understanding the volume was a little bit confusing with so many different colors. We implemented this and asked those people to test again. While they agreed it was easier to understand volume now, they preferred the older version as it was more fun to use and they didn’t know what to do with the volume information other than checking if our device was running correctly.

We also received feedback that we should work on making our interface more intuitive by adding help text so the user understands how to calibrate the image and pair the devices. This is currently being added in. They also reported that the paper piano really needed to be taped to the surface in order to comfortably play the piano, which was a valid point.

A few times, the warp didn't work. The warped images are saved in our code for debugging, but not shown to the user until they try to start playing the piano. The stage that frequently messes up our warp is in thresholding. Given that we have not yet hit our 95% accuracy mark, they requested that there be a check for the thresholding stage. As such, we are adding in a screen that shows the thresholded image, and asks the user to check it against an example thresholded imag before proceeding to the piano.

Overall, users enjoyed using our device. 4/5 said that they would use this for arranging music, though the glove seemed a bit fragile to be carrying around. When asked about the time delay between playing a key versus hearing it, users did not seem to notice too much of a time lag. They felt that it was overall a portable, nice-sounding piano interface.

VIII. Project Management

A. Schedule

In the appendix, Figure 12 shows our full GANTT chart for our project. Because we changed project ideas later in the semester, our GANTT chart starts on when we first started doing work on this idea (moved from "Airport Tag" idea to this "Piano AR" idea). We also added an additional two weeks to include the last week of classes as well as the first week of finals before Monday May 8th. We lastly removed audio processing as a task from our schedule and changed to implementing the app on the computer rather than the phone so some of the tasks on the left-hand side were also modified and updated to reflect this change.

B. Team Member Responsibilities

We split up the primary responsibilities of this project based off of each member's experience. While there might be some overlap where other members help out each other on their work, we assigned each member as the main lead for their primary responsibility.

Caroline focused on the gloves of our product. This included soldering the glove the together and onto the wrist device, making the glove adjustable for different hand sizes of the user, and working to send the BLE information to the laptop.

Lee focused on the computer vision aspect of our project because he has a lot of experience in CV. This includes warping and thresholding the paper piano, and implementing code to check where all 10 fingers are simultaneously and process this.

Nish has more experience in using the program Kivy, so she worked on the app interface. She worked on integrating the CV and glove information together, using asynchronous models and Kivy Clock, and the interface flow.

All of our members were involved the testing process.

C. Bill of Materials and Budget

Our Bill of Materials is listed in Table II (Appendix).

In our bill of materials, the items that we did buy at the beginning of the semester but did not use was the airplane phone mount holder with multi-directional dual 360-degree rotation. What we did not plan for in our design report but realized we needed to complete the project was the weight-lifting straps for the gloves and the ring light stand with the phone mount.

D. Risk Management

No one in the team has ever previously used Xcode for this Arduino BLE Nano, so our primary risk was getting it set up to wrap over the Kivy interface. To mitigate this risk, we planned on creating a dummy app to access the BLE and the speaker, and make a mini Kivy app wrapper to test. Instead, we ended up not using Xcode since we switched to running the app on the computer so this risk was eliminated this way.

Another risk was that Lee was in charge of the CV, but had a Windows OS on his personal computer and therefore couldn't test the work on Xcode, as Xcode only works on a Mac OS. To mitigate this risk, we decided that Lee can just send video and test frames as images to do vision processing on. Ultimately, we followed through with sending test images by taking photos with my pheon and running my program to test its functionality locally. However, it wasn't that we couldn't test it on Xcode, but more so because accessing the video stream from an apple iPhone to a Windows OS wasn't implemented yet.

The risk of having pressure sensors from Amazon is their robustness and quality. There is a risk of how the range of sensitivity performs according to our design requirements. To mitigate this risk, we planned on testing the current Amazon sensors to check if this risk existed. If they did not perform as expected, we would have ordered from Digikey for more robust ones. We ended up mitigating this risk when some of our pressure sensors did actually break even near the end of the project, we just ordered more from amazon last minute since amazon shipping proved to be quick enough for our needs.

IX. Ethical Issues

Some of the biggest ethical issues related to our project was with user privacy concerns since we were using a camera for computer vision. To mitigate this concern, the user can choose to point the camera wherever they desire and can turn it off whenever they want too. They are in control and the video stream isn't recording, but only processing frames in real time locally, which means it is still private.

If the product doesn't behave as intended, then the composer or arranger could go back to a real piano. Or in the adverse, but exceptionally rare case is if the battery shorts/explodes and injures the user. It is also not advised to use our product to learn the piano, as this may lead to hand injuries. However, our wrist-strap should also mitigate this ethical issue.

X. Related Work

There are a multitude of other work and projects dedicated towards creating a better piano experience for users. One project that relies on AR technology like ours attempts to create a piano experience at an actual real-life piano. In doing so their project goals strive to improve the user experience while playing on an actual piano, whereas for our project, we focus on portability and accessibility with the use case of composition and arrangement.

Their project creates an AR visual in a piano learning app on a real-life piano and draws out all the names of the notes on the piano for the user. In addition, the AR portion visualizes an animation of the notes coming toward them at the correct timing as to when the note is supposed to be played, similar to music rhythm video games such as “Guitar Hero” or “Piano Tiles 2”.

Some newer improvements to this technology was that it would also now just project a virtual piano keyboard for you so that you would be able to play it without an expensive real-life piano [3][10].

XI. Summary

Our system was able to meet most design specifications because we were able to get our system to a functional level.

A. Future Work

Some further design extensions are on creating a synth engine with MIDI capabilities. Another extension is adding a large range of volume levels for the users which would be achieved by increasing the accuracy for volume thresholding. Lastly, an extension would be added to the interface for having a capability for the pedal which allows you to have more accurate resonance for loud and soft notes.

B. Lessons Learned

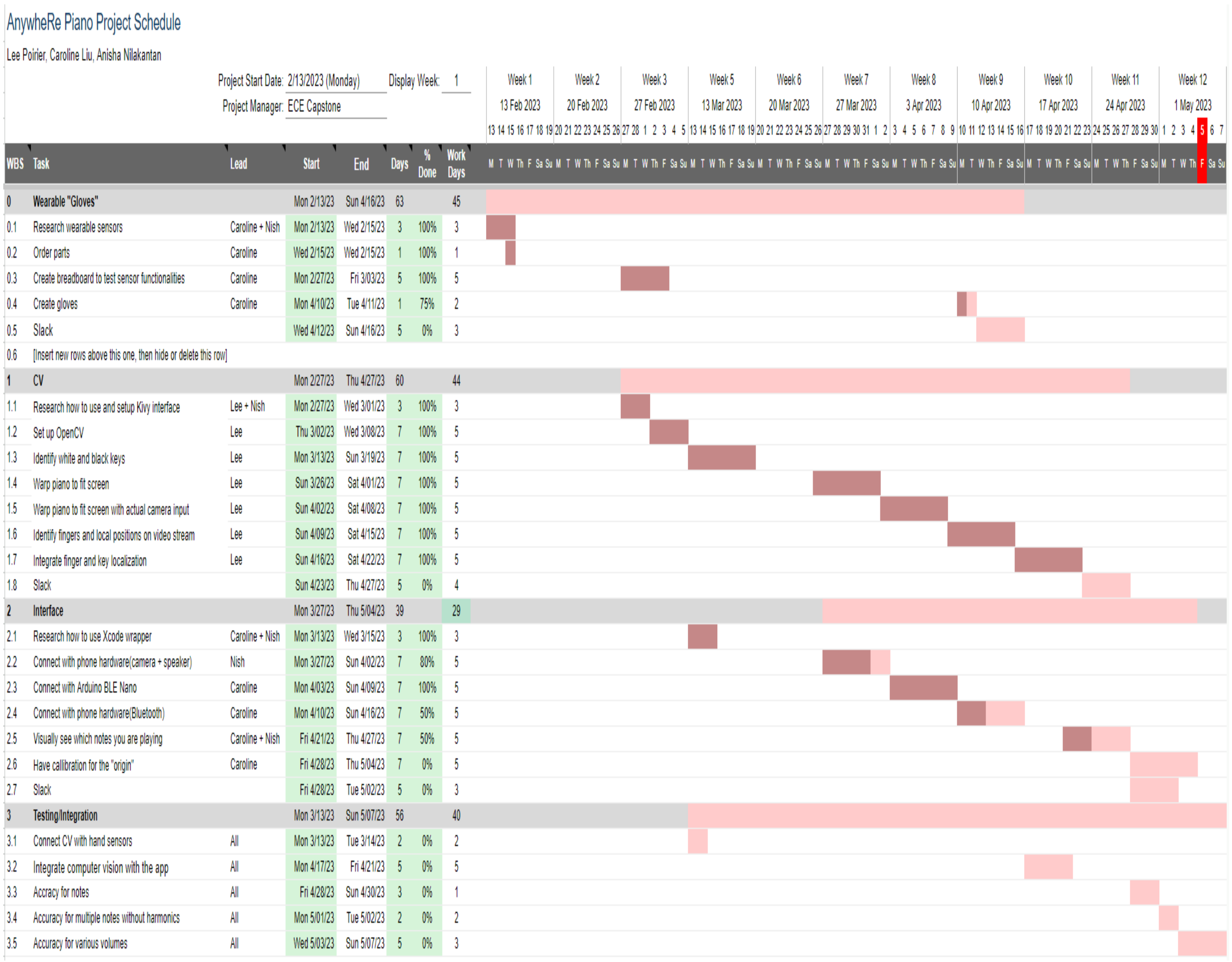
It is not a polished product though as there were some limitations that affected the overall design. The performance is affected under various lighting environments since there exists no perfect threshold values to identify the red markers. The outsourced pressure sensors proved to be more fragile than expected and lacked durability. Lastly, the audio processing goal was not reached as well due to the complexity of the task and it being not fundamental to the functionality of the design.

Glossary of Acronyms

AR - Augmented Reality
BLE – Bluetooth Low Energy
CV - Computer Vision
MIDI – Musical Instrument Digital Interface

References

1. geeksforgeeks.org, “OpenCV Python Tutorial,” Month, year, doi: 2.2023
2. Ravinath Wanni Arachch, “How to use iPhone camera for video capture in your openCV project” Feb, 2022.
3. Kyle Melnick, “AR Piano App For Quest Doesn’t Require An Actual Piano” Aug, .2022.
4. “Arduino Nano 32 BLE” in Arduino, Mar, 2023.
5. Hiroshi Kinoshita and Shinichi Furuya, “Loudness control in pianists as exemplified in keystroke force measurements on different touches,” in Journal of the Acoustical Society of America, May 2007.
6. “nRF52840” in Nordic Semiconductor, Feb, 2019.
7. “STM32WB55xx” in STMicroelectronics, Aug, 2022.
8. Lady Ada, “Force Sensitive Resistor (FSR)” in Adafruit Industries, Dec 2022.
9. Alan, “Making A Synth With Python — Controllers” in Python in Plain English, Mar 2021.
10. “Virtual Piano” in Crystal Magic Studio Ltd, 2023



AnywheRe Piano Project Schedule

Lee Poirier, Caroline Liu, Anisha Nilakantan

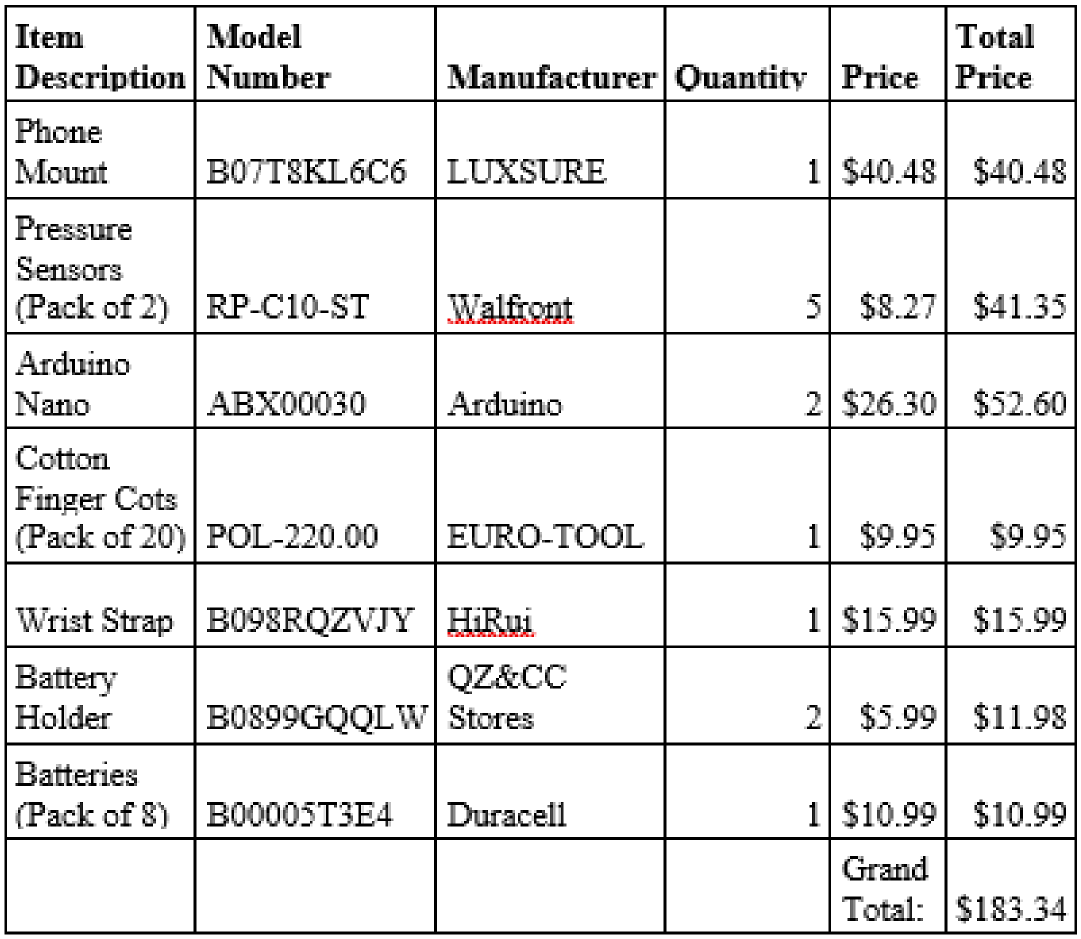
Project Start Date: 2/13/2023 (Monday) Display Week: 1

Project Manager: ECE Capstone

WBS	Task	Lead	Start	End	Days	% Done	Work Days
0	**Wearable "Gloves"**		Mon 2/13/23	Sun 4/16/23	63		45
0.1	Research wearable sensors	Caroline + Nish	Mon 2/13/23	Wed 2/15/23	3	100%	3
0.2	Order parts	Caroline	Wed 2/15/23	Wed 2/15/23	1	100%	1
0.3	Create breadboard to test sensor functionalities	Caroline	Mon 2/27/23	Fri 3/03/23	5	100%	5
0.4	Create gloves	Caroline	Mon 4/10/23	Tue 4/11/23	1	75%	2
0.5	Slack		Wed 4/12/23	Sun 4/16/23	5	0%	3
0.6	[Insert new rows above this one, then hide or delete this row]						
1	**CV**		Mon 2/27/23	Thu 4/27/23	60		44
1.1	Research how to use and setup Kivy interface	Lee + Nish	Mon 2/27/23	Wed 3/01/23	3	100%	3
1.2	Set up OpenCV	Lee	Thu 3/02/23	Wed 3/08/23	7	100%	5
1.3	Identify white and black keys	Lee	Mon 3/13/23	Sun 3/19/23	7	100%	5
1.4	Warp piano to fit screen	Lee	Sun 3/26/23	Sat 4/01/23	7	100%	5
1.5	Warp piano to fit screen with actual camera input	Lee	Sun 4/02/23	Sat 4/08/23	7	100%	5
1.6	Identify fingers and local positions on video stream	Lee	Sun 4/09/23	Sat 4/15/23	7	100%	5
1.7	Integrate finger and key localization	Lee	Sun 4/16/23	Sat 4/22/23	7	100%	5
1.8	Slack		Sun 4/23/23	Thu 4/27/23	5	0%	4
2	**Interface**		Mon 3/27/23	Thu 5/04/23	39		29
2.1	Research how to use Xcode wrapper	Caroline + Nish	Mon 3/13/23	Wed 3/15/23	3	100%	3
2.2	Connect with phone hardware(camera + speaker)	Nish	Mon 3/27/23	Sun 4/02/23	7	80%	5
2.3	Connect with Arduino BLE Nano	Caroline	Mon 4/03/23	Sun 4/09/23	7	100%	5
2.4	Connect with phone hardware(Bluetooth)	Caroline	Mon 4/10/23	Sun 4/16/23	7	50%	5
2.5	Visually see which notes you are playing	Caroline + Nish	Fri 4/21/23	Thu 4/27/23	7	50%	5
2.6	Have calibration for the "origin"	Caroline	Fri 4/28/23	Thu 5/04/23	7	0%	5
2.7	Slack		Fri 4/28/23	Tue 5/02/23	5	0%	3
3	**Testing/Integration**		Mon 3/13/23	Sun 5/07/23	56		40
3.1	Connect CV with hand sensors	All	Mon 3/13/23	Tue 3/14/23	2	0%	2
3.2	Integrate computer vision with the app	All	Mon 4/17/23	Fri 4/21/23	5	0%	5
3.3	Accracy for notes	All	Fri 4/28/23	Sun 4/30/23	3	0%	1
3.4	Accuracy for multiple notes without harmonics	All	Mon 5/01/23	Tue 5/02/23	2	0%	2
3.5	Accuracy for various volumes	All	Wed 5/03/23	Sun 5/07/23	5	0%	3

Figure 12: Gantt Chart

TABLE II. BILL OF MATERIALS



Item Description	Model Number	Manufacturer	Quantity	Price	Total Price
Phone Mount	B07T8KL6C6	LUXSURE	1	$40.48	$40.48
Pressure Sensors (Pack of 2)	RP-C10-ST	Walfront	5	$8.27	$41.35
Arduino Nano	ABX00030	Arduino	2	$26.30	$52.60
Cotton Finger Cots (Pack of 20)	POL-220.00	EURO-TOOL	1	$9.95	$9.95
Wrist Strap	B098RQZVJY	HiRui	1	$15.99	$15.99
Battery Holder	B0899GQQLW	QZ&CC Stores	2	$5.99	$11.98
Batteries (Pack of 8)	B00005T3E4	Duracell	1	$10.99	$10.99
				Grand Total:	$183.34