4-17-2018

Music Keyboard Based on Flexible Hybrid Electronics

Anthony Hanson
Western Michigan University, tony27hanson@gmail.com

A Musical Keyboard Based on Flexible Hybrid Electronics

WESTERN MICHIGAN UNIVERSITY

College of Engineering and Applied Sciences

ECE 4820

Jesse Echtinaw, Kyle Mann, Tony Hanson

Advisor and Sponsor: Dr. Massood Atashbar

4/27/2018

DISCLAIMER

This report was generated by a group of engineering seniors at Western Michigan University. It is primarily a record of a project conducted by these students as part of curriculum requirements for being awarded an engineering degree. Western Michigan University makes no representation that the material contained in this report is error free or complete in all respects. Therefore, Western Michigan University, its faculty, its administration or the students make no recommendation for use of said material and take no responsibility for such usage. Thus, persons or organizations who choose to use said material for such usage do so at their own risk.

Abstract

With the idea of creating a product using flexible hybrid electronics, the group decided to create a musical keyboard like a synthesizer. This is implemented using a printing technique called screen printing. This allowed the group to make several copies of the same sensors with a consistent thickness, roughness, and size. These keys, made of silver, are used as capacitive proximity sensors. When another conductive material comes within range of the key to change the polled capacitance by 2.7 pF, the note corresponding with that key is outputted.

In the interest of making this project a useful tool for someone playing the piano, a user interface was implemented that allows for volume control, octave control, and instrument change. Using a microcontroller, multiple sets of keys, as well as the user interface can all be incorporated together to make a compact, portable musical keyboard.

This design proved to be effective. The enclosure has a final volume of 2300 cm^3. Several changes were made to allow for a more portable case design including a holder for the keys and connectors. The case for the musical keyboard also includes an internal speaker and the user can either use the internal speaker or plug in an audio cable to listen through headphones or a more powerful speaker. The option to switch between battery power or a micro USB cable is available to help keep the design portable. The user can change instrument, volume, and octave at will as well as being able to play up to 24 notes simultaneously. This allows the user to compose meaningful music with the full range of a conventional piano.

Table of Contents

Need Statement

Conventional electronic keyboards for playing music are much too heavy and bulky to be able to take them on the go. The purpose of this project is to eliminate this problem by creating a keyboard using printed electronics with proximity sensors. This will allow the keyboard to be easily portable, with the ability to take it to unconventional places without it being out of place.

Introduction

The main goal of this project was to create an accurate and reliable musical keyboard based on capacitive sensing using printed electronics. For the layout of the keys themselves, they must be the size of standard piano keys. The keys must also be flexible to stay with the idea of portability and to showcase the capabilities of printed electronics. The case should be relatively small, and the microcontroller should be able to support functions like volume and octave control to give the user as much functionality as possible. To this end the group came up with some initial specifications that the project should meet.

In order to meet these design goals, the group partnered with Dr. Massood Atashbar, and the Center for Advanced Smart Sensors and Structures lab at Western Michigan University. This lab offers several different opportunities including the ability to work with specialized equipment such as: a laser etching machine, a gravure printer, a screen printer, as well as students and faculty that are knowledgeable in this area.

Specifications

1. Physical
    1.1. Size
        1.1.1. The key size should be the standard keyboard size (Width= 20mm).
        1.1.2. Case for components should fit in a box under 3,400cm^3.
    1.2. The keyboard must be flexible up to at least 45 degrees.
    1.3. There must be 24 keys on the keyboard.
    1.4. Entire system should not weigh more than 10lbs
2. Functionality
    2.1. Each key must play a specific note.
    2.2. The keyboard must be able to output sound.
    2.3. The keyboard should be able to change to multiple octaves.
    2.4. The keyboard's battery should have a playtime of at least two hours.
    2.5. Device must be able to turn on and off at the users will.
    2.6. User should be able to change the output volume of the device.
    2.7. Device pieces must be able to firmly lock together.
    2.8. The keyboard must run using a portable power source.
    2.9. Must be able to read and handle a voltage of around 0-5 V.
3. Safety
    3.1. When in use, the device must not administer a shock of any kind.
4. Cost
    4.1. Total cost of materials should be no more than $500.

Discussion

Layout Design

Before the final design that can be seen in appendix C, the group went through several different design possibilities. The first of which was meant for the laser etching machine in the Sensors Lab at Western Michigan. This design had a width of 4 inches to match the width of the copper that was to be etched on.

Some key features that were kept for the rest of the designs include a 1 mm gap between the keys and the associated wires. This helps to prevent crosstalk while pressing keys, as keys that are close to touching would experience changes in their capacitance when the user didn't intend to hit that key. The connection wires needed to have a gap of 1 mm, and a width of 2.54 mm from the middle of one wire to the middle of the next to fit perfectly with the 12 pin connectors that would allow the keyboard to interface with the microcontroller.

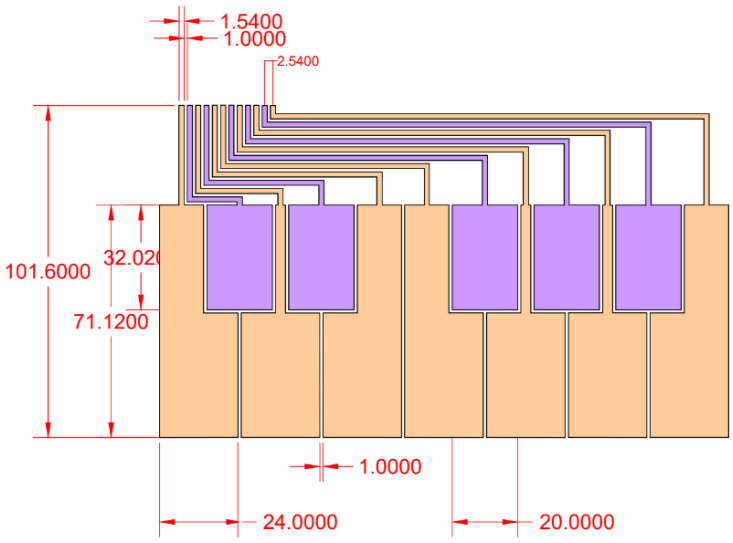


Figure 1 [Appendix A]

This design covers 12 distinct keys, 7 white keys and 5 flats/sharps keys. It starts on the left side with a C note, which is a common place for the right hand to start at and covers a full octave after that.

The next design, shown below, keeps many of the design decisions from the original keyboard, including the 1 mm gap and the key selections, but this was meant for screen printing, and as such, has different dimensions. The first design could be sent to the laser etching machine which could etch out the required design with precision. This design however, needs to be sent to a company who can create the screen for printing. The screen will be an 8” x 8” design on a stainless-steel mesh with an emulsion placed in all the areas where the design is not present.

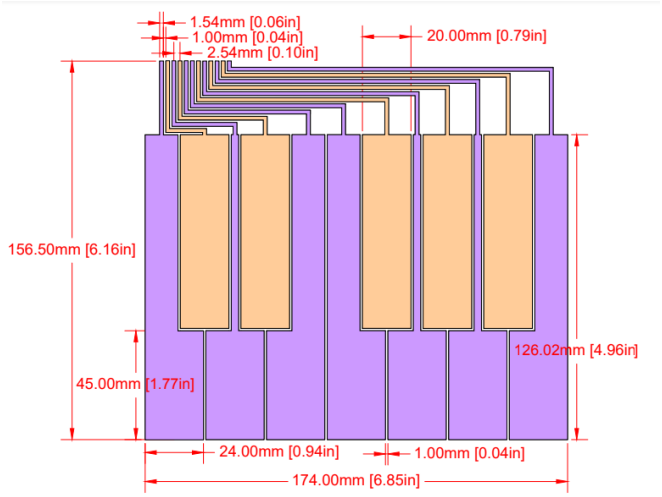


Figure 2 [Appendix B]

This design was not optimal however, since it was advised that the screen design be within a 6” x 6” area for the best quality with the available screen printer. It was also advised that 90^{o} angles in the wires are not optimal. This is because often the screen that’s returned from the manufactures will have defects in the sharp turns and make poor wire connections. The wire section for the connector needed to be elongated to

allow for the 12-pin connector to crimp correctly onto the wires without interfering with other traces.

Thus, the next design became the final design of the project, with rounded turns and on a 6” x 6” design. The connector now meets in the middle, which makes rounding the wires easier.

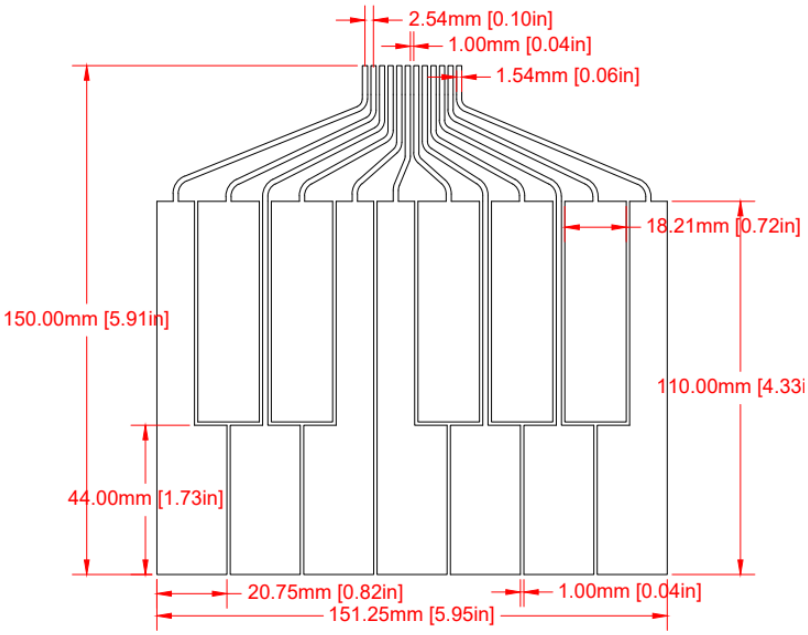


Figure 3 [Appendix C]

Hardware Design

The hardware section of the project included the implementation of the volume change, octave change, instrument selection, LCD screen, power source selection, and the audio selection.

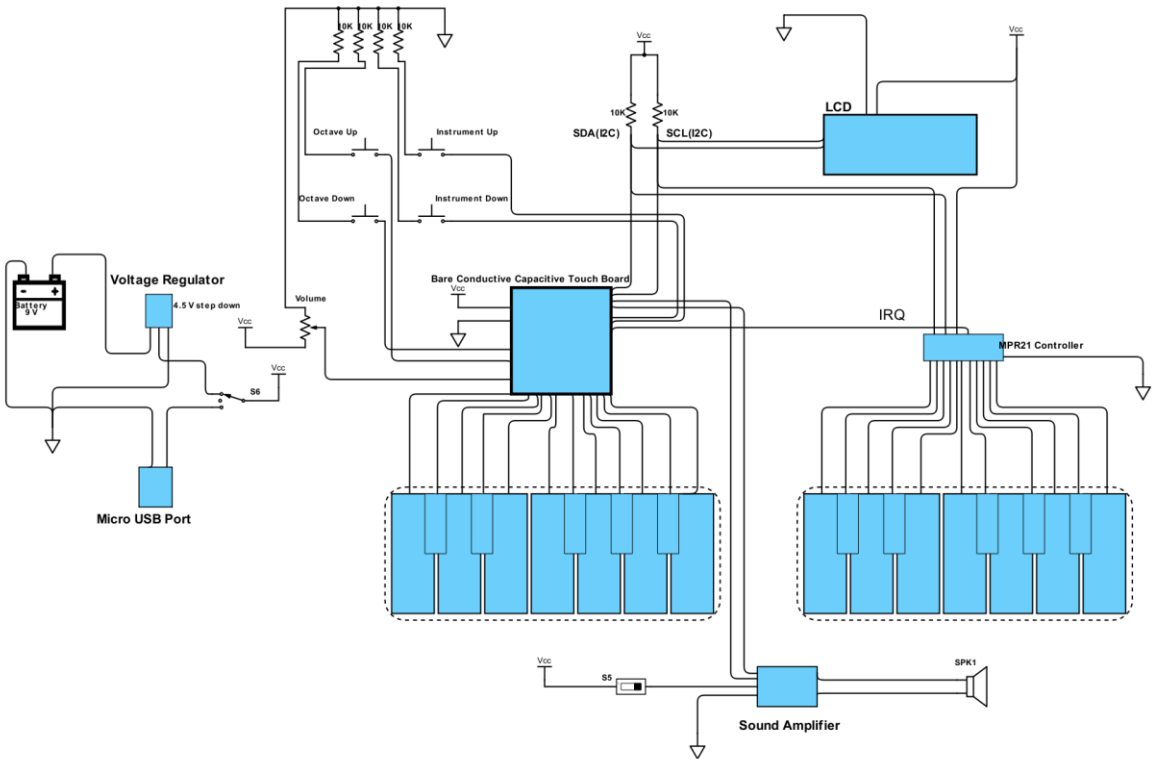


Figure 4 [Appendix E]

One of the key ideas behind this project was to make a keyboard that can be used anywhere while maintaining the functionality of a normal piano. To implement all these ideas while keeping the portability, some design compromises had to be met. It is not physically feasible to implement all 88 keys traditionally offered on a piano while also keeping it portable. As a compromise, the product offers the user the option to select from the entire range of the piano's octaves. Each of the two keyboard screens cover one octave. This allows the user to access two consecutive octaves at any moment, and they may shift up and down on the keyboard. This shift is controlled by two different switch buttons that are debounced in the software and are connected to 10 KΩ pull-down resistors.

The user may also control the changing of instruments by the switch buttons connected to pull-down resistors. This will cycle between 20 different instruments including:

1. Acoustic Piano	11. Tuba
2. Electric Piano	12. French Horn
3. Church Organ	13. Clarinet
4. Vibraphone	14. Flute
5. Xylophone	15. Sitar
6. Harmonica	16. Bagpipes
7. Electric Guitar	17. Woodblock
8. Violin	18. Synth Drum
9. Trumpet	19. Reverse Cymbal
10. Trombone	20. Bird Tweet

The volume control allows the user to change volume while playing with the use of a 10 kΩ twist potentiometer.

The instrument, volume, and the current octaves are all displayed on the LCD screen. The octave shows all the potential octaves as 0s with Xs where the current keys are. This way the user can see where they are relative to a standard size piano. The volume is shown as a bar across the screen. As the volume goes up, the bar increases in size and vice versa. This screen is controlled via I^2C pins on the controller and

connected to a common voltage source and ground with the other I^2C controlled devices.

The first set of keys are directly connected to the controller with the built in MPR121 controller and connected electrodes. The second set of keys are connected through an external MPR121 which is controlled via I^2C from the microcontroller, connected to the common voltage source and ground.

The user also has the option of switching between a mini-USB power source and a 9 V battery for the power source. This is accomplished on the side of the case with a 3-way switch that has options of: Off in the middle, mini USB on the top, and 9 V battery on the bottom. The 9 V battery is scaled down to 5 V by a voltage regulator, so it can be handled by the microcontroller.

There is also the option of changing between an internal and an auxiliary audio output. This is done with the use of a switch that turns on and off the internal speaker. The internal speaker is connected and controlled by a connection between the microcontroller and a PAM8302 audio amplifier controller.

All the components for this project are found on the parts list in appendix F. The final cost for one device can to be $134.52. This was well under our budget of $500.

Software Design

The software was all made in the Arduino IDE using C++. The main software drivers are the switches, the volume potentiometer, and the interrupts associated with the two different sets of keys.

I^2C is the driving force behind much of the project and makes a lot of it possible given the limited number of pins available. I^2C is essentially a master to multiple-slave

protocol that allows a board to control multiple devices through two pins, SDA (data pin) and SCL (clock line). The liquid crystal display (LCD) is controlled via I^2C, as well as the supplemental MPR121 controller and the MPR121 controller built into the main microcontroller.

Both MPR121 controllers are connected to interrupt pins on the microcontroller. When a key is pressed on one of the keyboards, the associated interrupt will trigger. This will send the code to the MPR121 header file, where the interrupt handler changes the return value of the touchStatusChanged method. When either the keyboard touch status changes, the software goes through a loop looking to see which key was pressed. Once found it will either send that key to the released or touched method in the MPR121 header files.

All the push buttons are debounced to detect only when a value has been changed for more than 50 ms. This allows the user to accurately change instruments or octaves when pressing the button without making the octave or instrument change multiple units when they only intended to press it once.

On every loop of the code, the value of each displayed variable (volume, octave, and instrument) will be polled to see if they changed. If they did, then the LCD is cleared, and the new values are displayed through I^2C.

The flow chart of the code as well as the code itself can be found in appendix G and H respectively.

Capacitance

The sensors made are proximity touch sensors. These touch sensors change in capacitance the closer a contact becomes to it. The keys measure a non-touch

capacitance of 11-13 pF depending on the key. When a finger applies a touch to the sensor, the capacitance increases by a value of 24-27 pF. These values can be found in appendix D. When this happens, it triggers an interrupt to the microcontroller letting the controller know that there has been contact on one sensor.

The MPR121 measures the capacitance of the keys by sending a 16uA signal to a key for 0.5 us to charge the capacitor. Then the capacitor discharges for 0.5 us. After this the next key in the sequence is charged. The MPR121 looks at the peak voltage that the capacitor hits and converts this to a 10-bit ADC number. This 10-bit number is sent to the microcontroller if the number exceeds the threshold set to 20 V_{ADC} which is about a 2.7 pf difference.

Each key is measured ever 6 us, and if the capacitance changes from the base capacitance, it triggers an interrupt. When this happens, the microcontroller will play the proper key noise.

Enclosure Design

To make the device portable, there was a need to centralize all the components within a single enclosure. The enclosure was designed using Autodesk Fusion 360. The key points to the design were to make the case compact and contain everything necessary to be a standalone device. The case contains holes for the 4 push buttons. It also contains openings for the potentiometer, LCD screen, power switch, micro USB plugin, audio switch, and audio cable plug in. The case also has a holder for the battery, keyboard keys, and wires to connect them. This enclosure includes engravings to help

brief what each button and switch is for. There are also engravings on the front of the enclosure detailing a list of the available instruments.

After this design was finished, the group decided to 3D print it. To make this possible, the model was exported to an stl file, and then imported into Cura. Some of the Cura setting were as follows: 0.3 mm layer height, 45 mm/s print speed, and an infill of 20%. This was exported to a gcode file and uploaded onto the 3D printer. The 3D printer used was the Creality CR-10. This print took about 23 hours to complete and used around 350 grams of 1.75 mm PLA plastic.

Conclusion

The final product meets every original specification while also implementing new features to better achieve the goal of portability and functionality. These extra features include: the LCD screen to display the characteristics of the outputted sound, the ability to choose multiple instruments, an internal speaker or external speaker, and power from micro USB cable or battery. The user is also able to play many different notes at once and has the ability to play every note on a conventional piano through octave change.

Future improvements for follow up projects on this would include changing the user interface, changing the capacitive proximity sensing to pressure sensing, and better wire management.

Changing the user interface could be completed by a smartphone application and implementation of a Bluetooth component on the microcontroller. This would allow the designer to remove all the physical components of the interface like the push buttons, potentiometer, and LCD screen. This will decrease the overall size of the design and make it even more portable.

Changing the proximity sensing to pressure sensing would allow greater versatility in how the user can create their music. Instead of only having one set sound every time a key is pressed, the sound will be able to increase or decrease in volume depending on how hard the user presses the keys.

The project could be improved in the aspect of portability with some wire management. The connectors from the box to the keys are very long to the point where you need substantial table space to properly use the device. By making these wires shorter, the space needed to use the device would be drastically cut down. The space needed for the box could also be cut down by placing the connectors for the wires to the keys on the sides of the case, decreasing the width of the enclosure.

References

[1]N. Pontius, "What is Printed Electronics? Learn about How Printed Electronics is Used, the Applications, Challenges, Benefits, and More", *Pannam Blog*, 2017.

[2]"Roadmap for Organic and Printed Electronics, 7th Edition", International Exhibition and Conference for the Printed Electronics Industry, 2017.

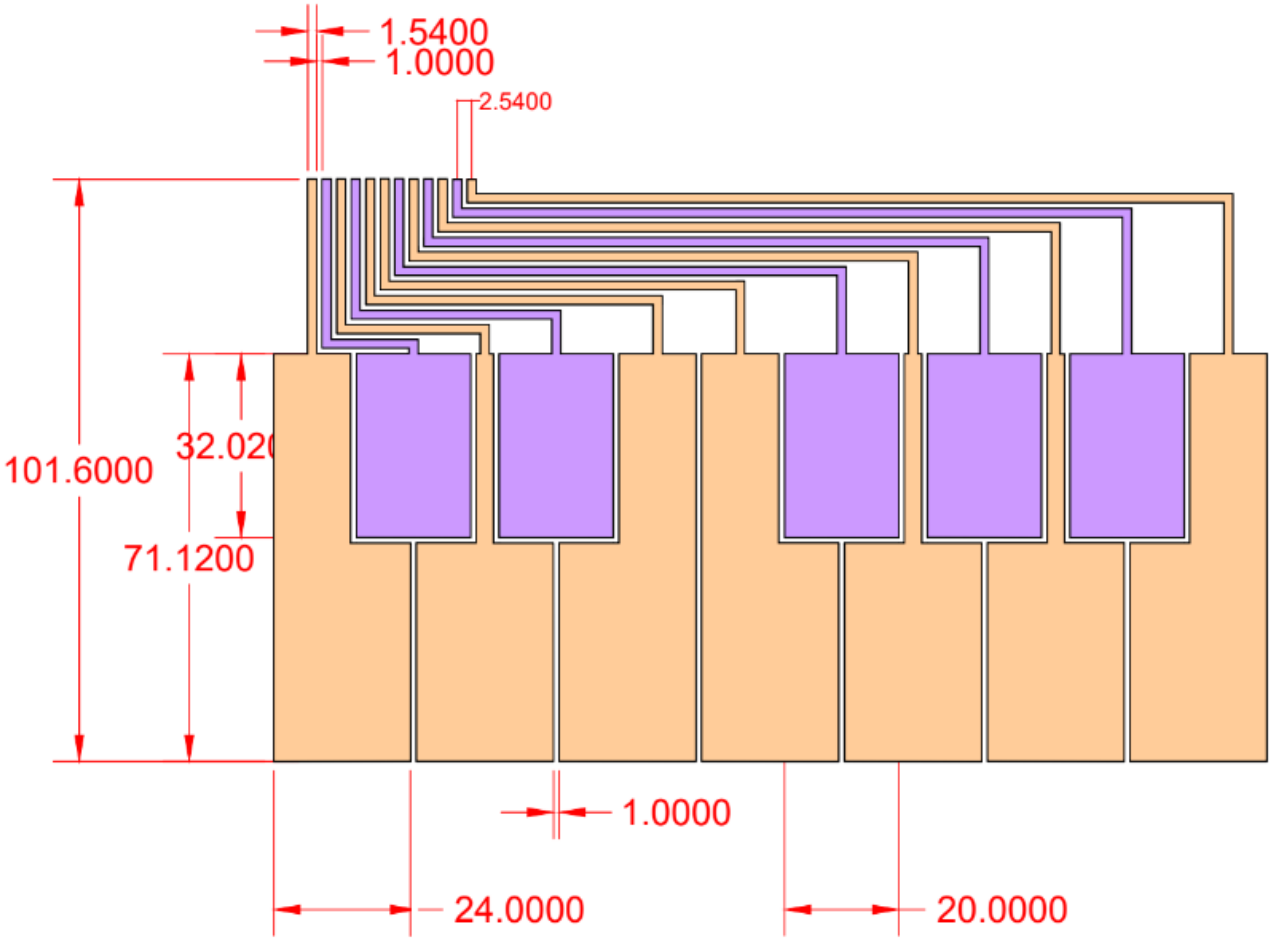
[3]B. Davison, "Techniques for Robust Touch Sensing Design", *Microchip Technology Inc. November 2012*.

[4]B. Board and A. sauce!, "Bare Conductive Touch Board - DEV-13298 - SparkFun Electronics", *Sparkfun.com*, 2017. [Online]. Available: https://www.sparkfun.com/products/13298.

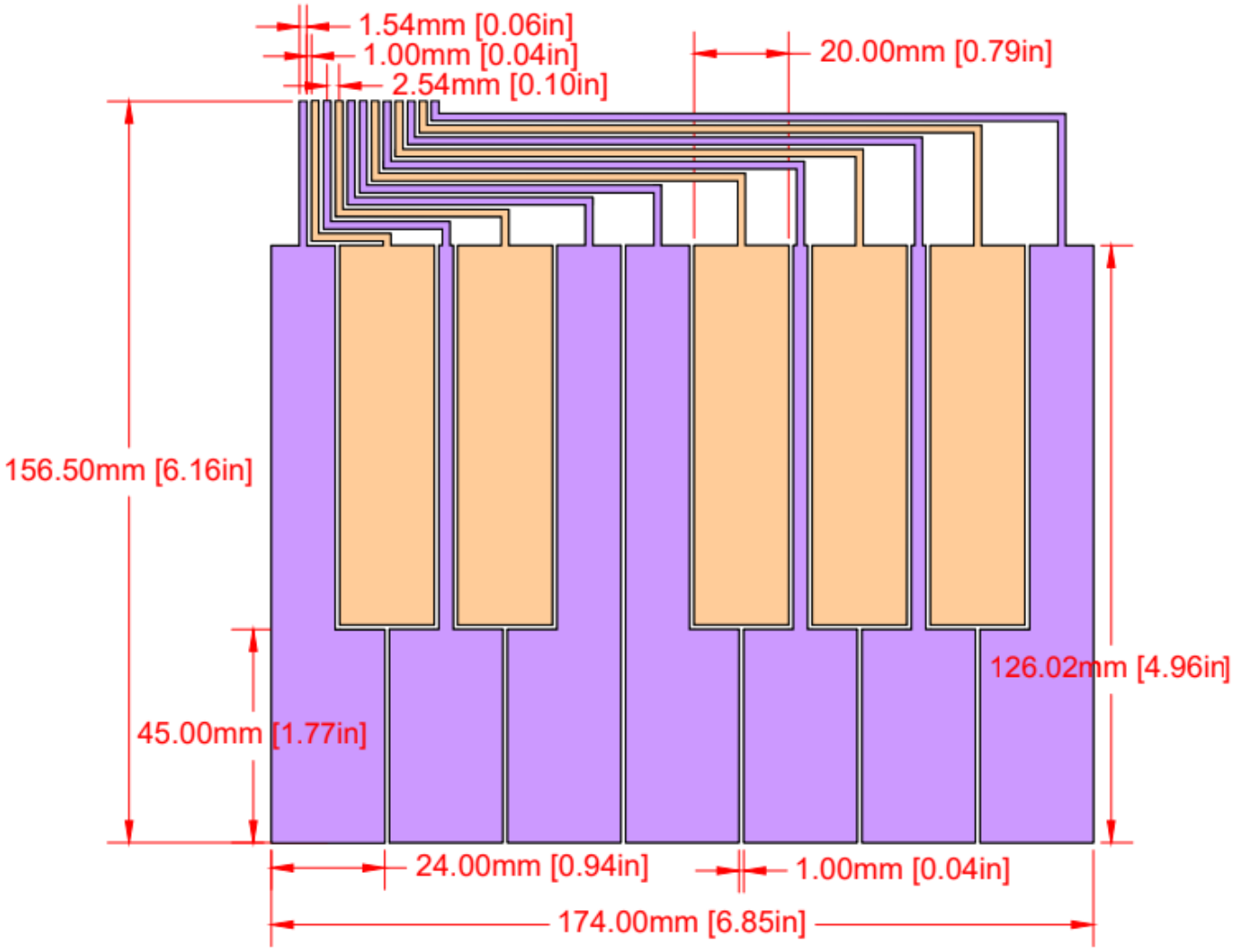
[5]Freescale, “Proximity Capacitive Touch Sensor Controller,” MPR121 Datasheet 2009, Revised September, 2010

Appendix

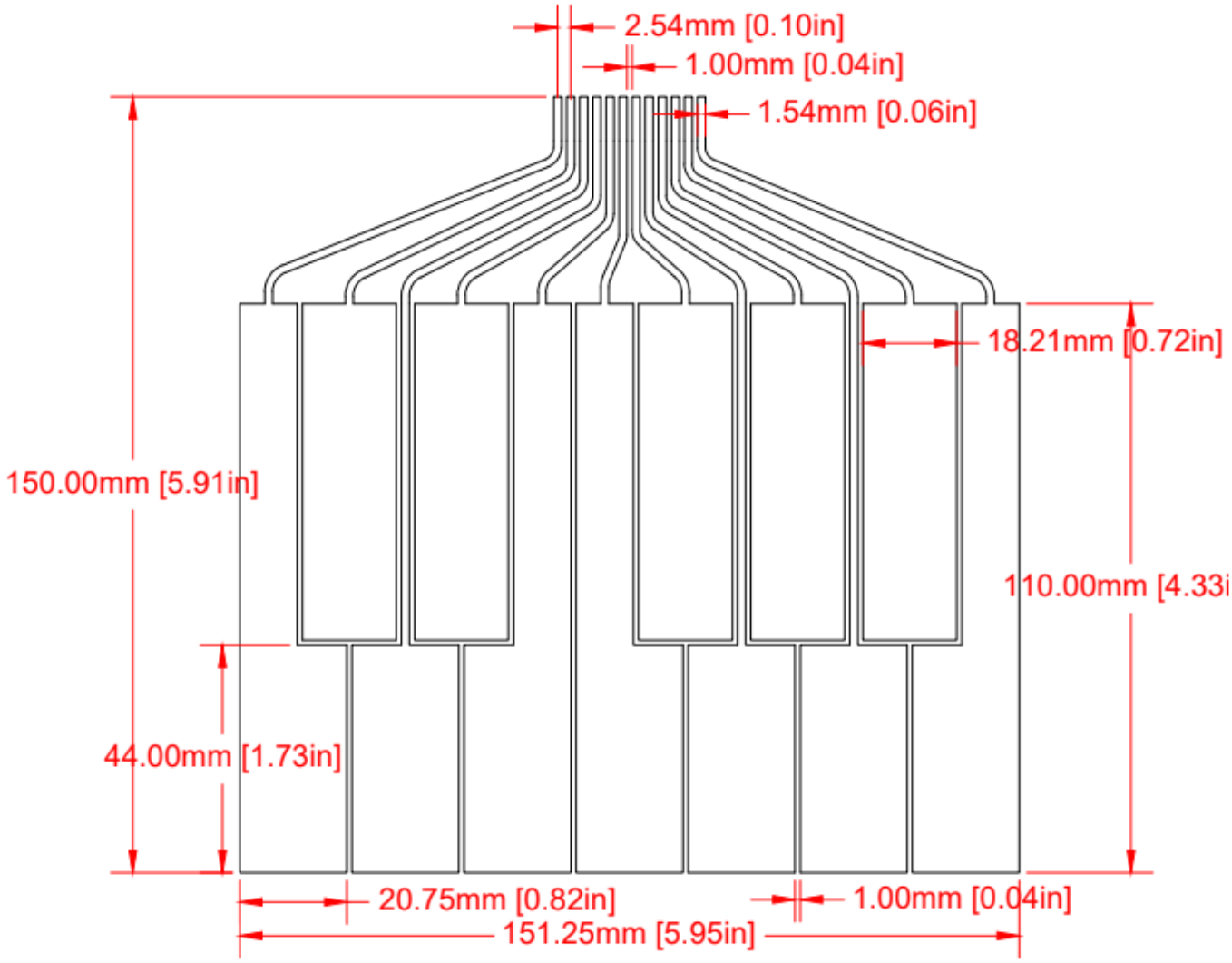
A First Design



B Second Design



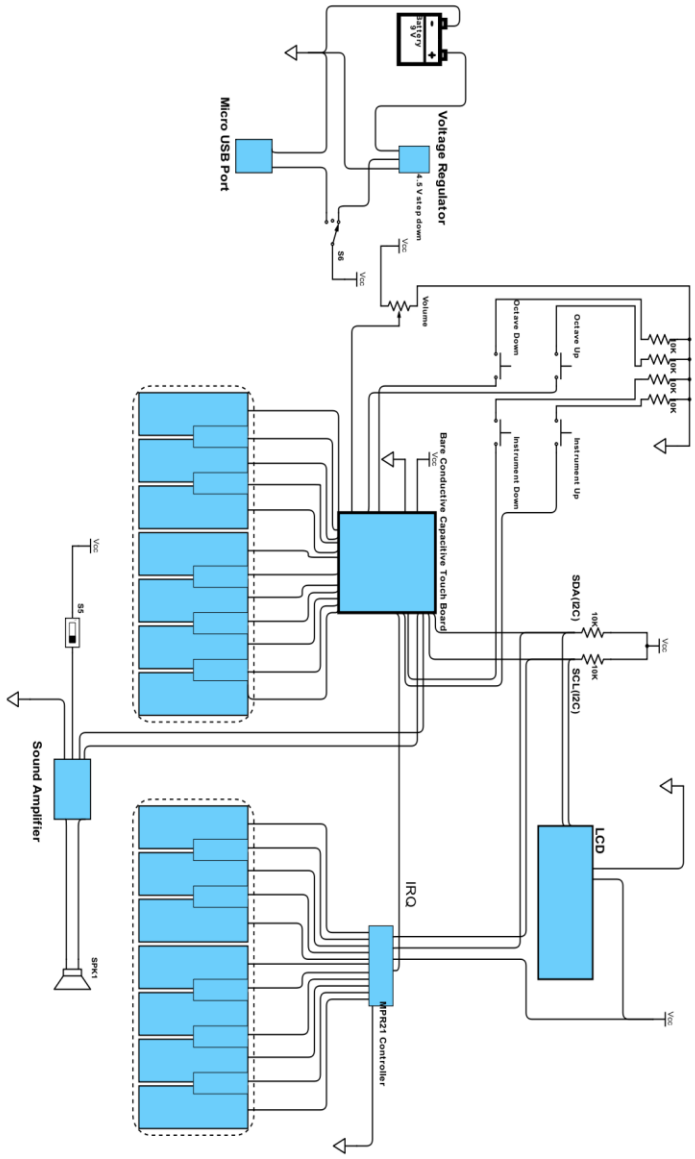
C Third Design



D Capacitance Values

	Theoretical		Experimental		Percent Error	
	CBase (pF)	CPressed (pF)	CBase (pF)	CPressed (pF)	CBase	CPressed
Key 1 (Silver)	19.67	35.437	13.2	40.5	49.02%	12.50%
Key 2 (Silver)	14.816	35.437	11.2	34.7	32.29%	2.12%
Key 3 (Silver)	12.481	35.437	12.6	36.3	0.94%	2.38%

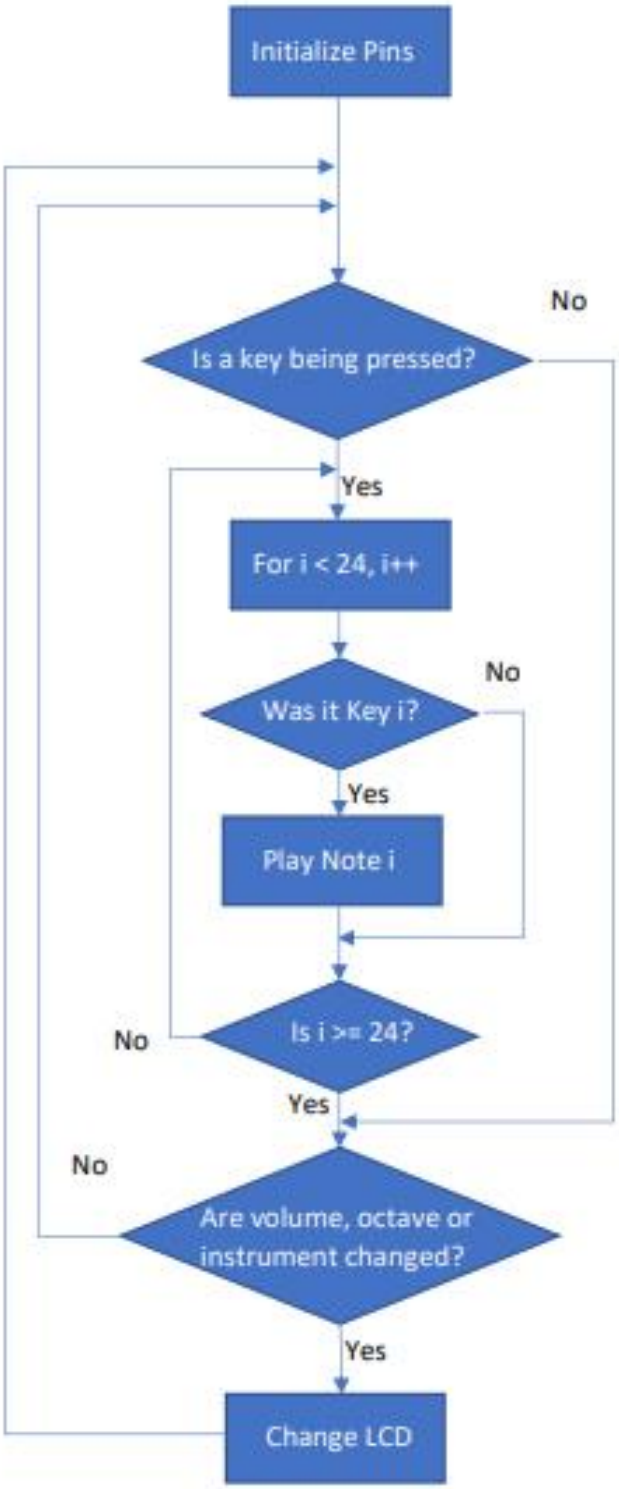
E Schematic


Micro USB Port
Voltage Regulator
4.5 V step down
Battery 3 V
S6
Vcc
Volume
Octave Down
Octave Up
10K
Instrument Down
Instrument Up
Base Conductive Capacitive Touch Board
SDA(I2C)
SCL(I2C)
S5
Sound Amplifier
SPK1
IRQ
MPR21 Controller
LCD

F Parts List

Part Name	Description	Quanity	Cost Each	Total Cost	Website
Bare Touch Capacitive Board	This is the master board.	1	$69.00	$ 69.00	Bareconductive.com
Aux Cable	Connect an external speaker or headphones to the board.	1	$4.50	$ 4.50	amazon.com
Wall plug	To have the board constantly plugged into the wall while on display	1	$3.00	$ 3.00	amazon.com
LCD Screen	To display Octave, Volume, and Insturment	1	$10.49	$ 10.49	amazon.com
Ribon Cables	To connect the keyboard to the controllers.	1	$6.98	$ 6.98	amazon.com
Push Buttons	To change octaves and insturment	1	$2.66	$ 2.66	amazon.com
Micro USB Breakout Board	To receive power	1	$1.50	$ 1.50	digikey.com
Micro USB Cable	Used to power and program the board	1	$2.62	$ 2.62	digikey.com
Internal Speaker	Speaker built inside the box	1	$4.83	$ 4.83	digikey.com
MPR121	The other touch sensor to make a total of 24 keys.	1	$7.95	$ 7.95	digikey.com
Audio Amplifier	Amplifer sound	1	$3.95	$ 3.95	digikey.com
Voltage regulator	For the battery input to give off 5 V	1	$0.56	$ 0.56	digikey.com
9V battery connector	To power the device thru a battery	1	$0.60	$ 0.60	digikey.com
3-way switch	To switch between wall power and battery power and off	1	$2.11	$ 2.11	digikey.com
10k Resistor	Pull down Resistors	6	$0.10	$ 0.60	digikey.com
10k pot	To change the volume	1	$1.02	$ 1.02	digikey.com
Connector	This is the connector which will go from the printed keyboard to the wires that will lead into the board	2	$1.79	$ 3.58	digikey.com
Switch for speakers	To switch from the internal speaker to an external speaker / headphones	1	$0.73	$ 0.73	digikey.com
Bread board (Prototyping)	A circuit board to solder everthing into	1	$1.55	$ 1.55	digikey.com
PLA Plastic	This is the plastic used to make the 3D printed box	350 g	$0.018 per gram	$6.30	amazon.com
			Total Price without shipping	**$ 134.52**	

G Software Flow Chart



H Code

```
// include the relevant libraries
#include <MPR121.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal_I2C.h>

//This is the instrument being used
int music [] = {1, 4, 19, 11, 13, 22, 26, 40, 56, 57, 58, 60, 71, 73, 104, 109, 115, 118, 119, 123};

//These are the octaves
const byte allNotes[7][12] = {{24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35}, {36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47},
  {48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59}, {60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71}, {72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83},
  {84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95}, {96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107}};

String instArr[] ={ "1 Acoustic Piano", "2 Electric Piano", "3 Church Organ", "4 Vibraphone" , "5 Xylophone", "6
Harmonica", "7 Electric Guitar" , "8 Violin", "9 Trumpet", "10 Trombone"
  , "11 Tuba", "12 French Horn", "13 Clarinet", "14 Flute", "15 Sitar", "16 Bag Pipe", "17 Woodblock", "18 Synth
Drum", "19 Reverse Cymbal", "20 Bird Tweet"};

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);// Sets up LCD
MPR121_t master = MPR121_t();// For the touch sensors connected to the bare board
MPR121_t slave = MPR121_t(); // Touch sensors connected to the MPR121 break out chip
SoftwareSerial mySerial(12, 10); // Soft TX on 10, we don't use RX in this code. Used for MIDI

// Touch Board Setup variables
#define firstPin 0
#define lastPin 11
#define masterInterrupt 4
#define slaveInterrupt 5

// VS1053 setup
byte note = 0; // The MIDI note value to be played
byte resetMIDI = 8; // Tied to VS1053 Reset line

int octave = 4;
int octaveUp = 1;
int octUpReading;
int octUpLastState = LOW;
unsigned long octUpLastTime = 0;
int octUpCurrentState;

int octaveDown = A1;
int octDownReading;
int octDownLastState = LOW;
unsigned long octDownLastTime = 0;
int octDownCurrentState;

int instrument = 0;
```

```
int instrumentUp = 12;
int instUpReading;
int instUpLastState = LOW;
unsigned long instUpLastTime = 0;
int instUpCurrentState;

int instrumentDown = 11;
int instDownReading;
int instDownLastState = LOW;
unsigned long instDownLastTime = 0;
int instDownCurrentState;

int volumeKnob = A3;
int volume127Conv = 80;
int volume11Conv = 0;
int volumeRawReading;
int volumeCheckDiff = 0;
int volumeDisplay;



void setup() {
  pinMode(octaveUp, INPUT);
  pinMode(octaveDown, INPUT);
  pinMode(instrumentUp, INPUT);
  pinMode(instrumentDown, INPUT);
  pinMode(volumeKnob, INPUT);

  /*Initialize display*/
  startLCD();
  delay(4000);//Delay for 4 seconds
  // initialise MIDI
  setupMidi();

  delay(1000);//Delay for 1 seconds
  master.setTouchThreshold(5);
  slave.setTouchThreshold(5);
  clearDisplay();
  lcdDisplayInterface();
}



void loop() {
  //old values to determine if there is a change
  int tempOct = octave;
  int tempVol11Conv = volume11Conv;
  int tempInst = instrument;

  //checks the external buttons for a change
  checkOctUp();
  checkOctDown();
  checkInstUp();
```

```
 checkInstDown();
 volumeControl();

 //update LCD if there is a change in Octave, Volume, or Instrument
 if (tempOct != octave || tempVol11Conv != volume11Conv || tempInst != instrument) {
  lcdDisplayInterface();
 }

 /*
  * Check to see if key is pressed or released on bare touch board
  */
 if (master.touchStatusChanged()) {
    master.updateAll();

    for (int i = firstPin; i <= lastPin; i++) {
      note = allNotes[octave][lastPin - i];
      if (master.isNewTouch(i)) {
        noteOn(0, note, 0x60);
      }
      else if (master.isNewRelease(i)) {
       noteOff(0, note, 0x60);
      }
    }
  }

 /*
  * Check to see if key is pressed or released on MPR121 breakout board
  */
 if (slave.touchStatusChanged()) {
    slave.updateAll();
    for (int r = firstPin; r <= lastPin; r++) {
       note = allNotes[octave - 1][r];
       if (slave.isNewTouch(r)) {
          noteOn(0, note, 0x60);
       }
       else if (slave.isNewRelease(r)) {
          noteOff(0, note, 0x60);
       }
     }
  }

 // end of loop
}


//Checks to see if the button for octave up is pressed.
//If so, then this changes the octave up one.
//Uses a debounce code.
void checkOctUp() {
  octUpReading = digitalRead(octaveUp);
  if (octUpReading != octUpLastState) {
    octUpLastTime = millis();
  }
```

```
  if ((millis() - octUpLastTime) > 50) {
    if (octUpReading != octUpCurrentState) {
      octUpCurrentState = octUpReading;
      if (octUpCurrentState == HIGH) {
        if (octave != 6) {
          for (int j = firstPin; j <= lastPin; j++) {
            note = allNotes[octave][lastPin - j];
            noteOff(0, note, 0x60);
            note = allNotes[octave - 1][lastPin - j];
            noteOff(0, note, 0x60);
          }
          octave++;}}}}
  octUpLastState = octUpReading;
}


//Checks to see if the button for octave down is pressed.
//If so, then this changes the octave down one.
//Uses a debounce code.
void checkOctDown() {
  octDownReading = digitalRead(octaveDown);
  if (octDownReading != octDownLastState) {
    octDownLastTime = millis();
  }
  if ((millis() - octDownLastTime) > 50) {
    if (octDownReading != octDownCurrentState) {
      octDownCurrentState = octDownReading;
      if (octDownCurrentState == HIGH) {
        if (octave != 1) {
          for (int i = firstPin; i <= lastPin; i++) {
            note = allNotes[octave][lastPin - i];
            noteOff(0, note, 0x60);
            note = allNotes[octave - 1][lastPin - i];
            noteOff(0, note, 0x60);
          }
          octave--;}}}}
  octDownLastState = octDownReading;
}


//Checks to see if the button for instrument up is pressed.
//If so, then this changes to the next instrument.
//Uses a debounce code.
void checkInstUp() {
  instUpReading = digitalRead(instrumentUp);
  if (instUpReading != instUpLastState) {
    instUpLastTime = millis();
  }
  if ((millis() - instUpLastTime) > 50) {
    if (instUpReading != instUpCurrentState) {
      instUpCurrentState = instUpReading;
      if (instUpCurrentState == HIGH) {
```

```
      if (instrument == 19)
        instrument = 0;
      else
        instrument ++;
      for (int j = firstPin; j <= lastPin; j++) {
         note = allNotes[octave][lastPin - j];
         noteOff(0, note, 0x60);
         note = allNotes[octave - 1][lastPin - j];
         noteOff(0, note, 0x60);
        }
      instrumentChange();
      talkMIDI(0xB0, 0, 0x00); // Default bank GM1
      talkMIDI(0xC0, music[instrument], 0);}}}
  instUpLastState = instUpReading;
}


//Checks to see if the button for instrument down is pressed.
//If so, then this changes to the next instrument.
//Uses a debounce code.
void checkInstDown() {
  instDownReading = digitalRead(instrumentDown);
  if (instDownReading != instDownLastState) {
    instDownLastTime = millis();
  }
  if ((millis() - instDownLastTime) > 50) {
    if (instDownReading != instDownCurrentState) {
      instDownCurrentState = instDownReading;
      if (instDownCurrentState == HIGH) {
        if (instrument == 0)
          instrument = 19;
        else
          instrument--;
          for (int j = firstPin; j <= lastPin; j++) {
            note = allNotes[octave][lastPin - j];
            noteOff(0, note, 0x60);
            note = allNotes[octave - 1][lastPin - j];
            noteOff(0, note, 0x60);
          }
        instrumentChange();
        talkMIDI(0xB0, 0, 0x00); // Default bank GM1
        talkMIDI(0xC0, music[instrument], 0);}}}
  instDownLastState = instDownReading;
}


//Read a potentiometer to ge a value 0-1023.
//Converts that value to a new scale of 0-127 for MIDI purposes,
//Then converts to a scale of 0-11 for LCD display purposes.
void volumeControl() {
  volumeRawReading = analogRead(volumeKnob);
  volume127Conv = map(volumeRawReading, 0, 1023, 0, 127);
  if (volume127Conv != volumeCheckDiff) {
```

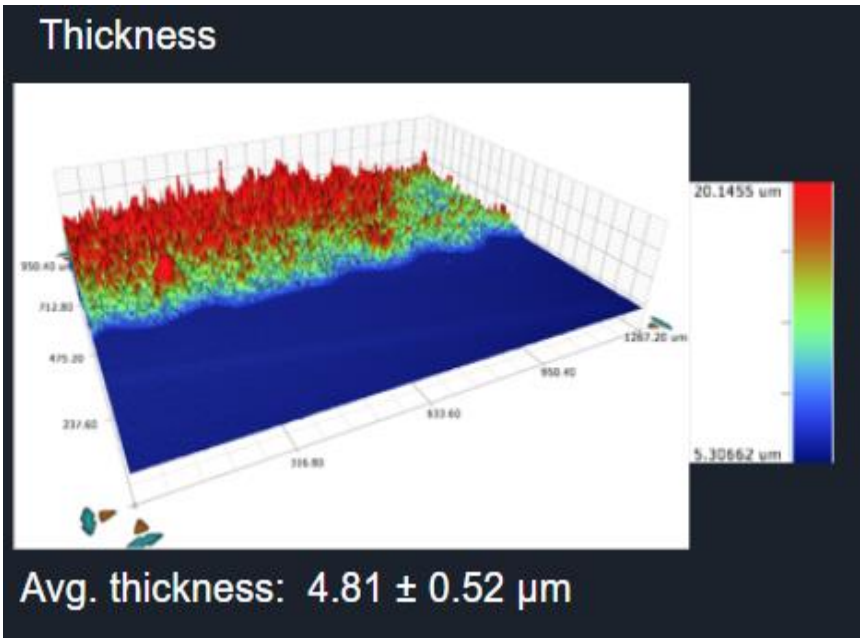
```
    volumeCheckDiff = volume127Conv;
    talkMIDI(0xB0, 0x07, volume127Conv);
    volume11Conv = map(volumeRawReading, 0, 1023, 0, 11);
  }
}


//This code will set the change the octave to the insturments default octave
void instrumentChange() {
  switch (instrument) {
    case 0: octave = 4;
      break;
    case 1: octave = 4;
      break;
    case 2: octave = 3;
      break;
    case 3: octave = 4;
      break;
    case 4: octave = 4;
      break;
    case 5: octave = 4;
      break;
    case 6: octave = 4;
      break;
    case 7: octave = 4;
      break;
    case 8: octave = 4;
      break;
    case 9: octave = 3;
      break;
    case 10: octave = 2;
      break;
    case 11: octave = 4;
      break;
    case 12: octave = 4;
      break;
    case 13: octave = 5;
      break;
    case 14: octave = 4;
      break;
    case 15: octave = 4;
      break;
    case 16: octave = 4;
      break;
    case 17: octave = 4;
      break;
    case 18: octave = 3;
      break;
    case 19: octave = 6;
      break;
    default:
      break;
  }
```

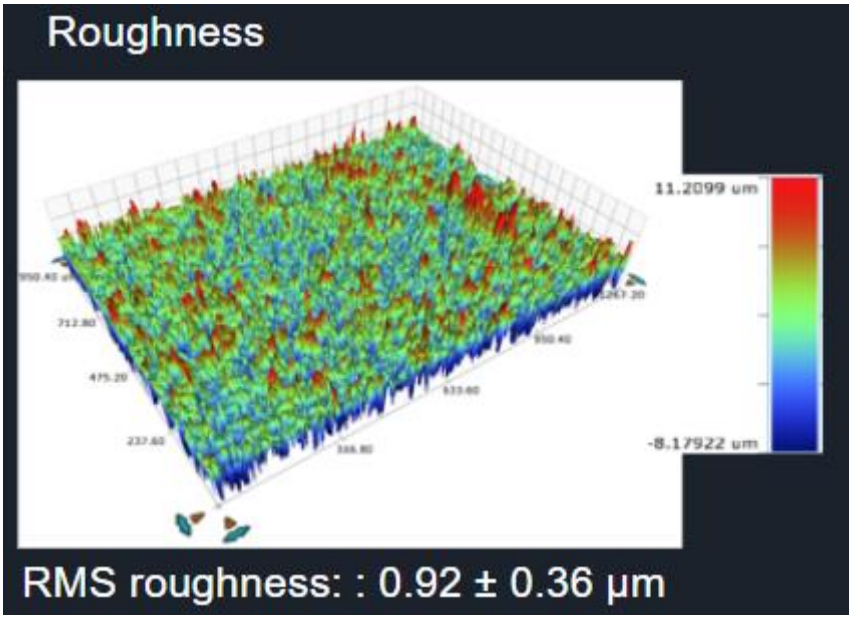
```
}


//Tells the MP3 decoder to play this sound
void noteOn(byte channel, byte note, byte attack_velocity) {
  talkMIDI( (0x90 | channel), note, attack_velocity);
}


//Tells the MP3 decoder to stop this sound.
void noteOff(byte channel, byte note, byte release_velocity) {
  talkMIDI( (0x80 | channel), note, release_velocity);
}



//Communicates with the MP3 decoder.
void talkMIDI(byte cmd, byte data1, byte data2) {
  mySerial.write(cmd);
  mySerial.write(data1);
  if ( (cmd & 0xF0) <= 0xB0)
    mySerial.write(data2);
}



void setupMidi() {
  // Setup soft serial for MIDI control
  mySerial.begin(31250);
  Wire.begin();

  if (!master.begin(0x5C)) {// 0x5C is the MPR121 I2C address on the Bare Touch Board
    while (1);
  }
  master.setInterruptPin(masterInterrupt); // pin 4 is the MPR121 interrupt on the Bare Touch Board
  master.updateTouchData();// initial data update

  if (!slave.begin(0x5A)) {  // 0x5A is the MPR121 I2C address on the MPR121 Breakout board
    while (1);
  }
  slave.setInterruptPin(slaveInterrupt);// pin 5 is the MPR121 interrupt for the MPR121 breakout board
  slave.updateTouchData();// initial data update

  // Reset the VS1053
  pinMode(resetMIDI, OUTPUT);
  digitalWrite(resetMIDI, LOW);
  delay(100);
  digitalWrite(resetMIDI, HIGH);
  delay(100);

  talkMIDI(0xB0, 0x07, volume127Conv);//Inital volume
  talkMIDI(0xB0, 0, 0x00); // GM1 Bank
  talkMIDI(0xC0, music [instrument], 0); // Set inital instrument.
```

```
}


//This is displayed during start up to show that the board is starting up
void startLCD() {
  lcd.begin(20, 4);
  lcd.setCursor(0, 0);
  lcd.print("Senior Design 2018");
  lcd.setCursor(0, 1);
  lcd.print("Setup in progress,");
  lcd.setCursor(0, 2);
  lcd.print ("Please don't touch..");
}


//Clear the LCD Screen
void clearDisplay() {
  lcd.setCursor(0, 0);
  lcd.print("                  ");
  lcd.setCursor(0, 1);
  lcd.print("                  ");
  lcd.setCursor(0, 2);
  lcd.print("                  ");
  lcd.setCursor(0, 3);
  lcd.print("                  ");
}


//LCD display while the program on in the main loop
void lcdDisplayInterface() {
  //LCD dipsplay for instrument change
  lcd.setCursor(0, 0);
  lcd.print("                  ");
  lcd.setCursor(0, 0);
  lcd.print(instArr[instrument]);

  //LCD dipsplay for octave change
  lcd.setCursor(0, 1);
  lcd.print("Octave: ");
  lcd.setCursor(8, 1);
  switch (octave) {
    case 1: lcd.print("XX00000");
      break;
    case 2: lcd.print("0XX0000");
      break;
    case 3: lcd.print("00XX000");
      break;
    case 4: lcd.print("000XX00");
      break;
    case 5: lcd.print("0000XX0");
      break;
    case 6: lcd.print("00000XX");
      break;
```

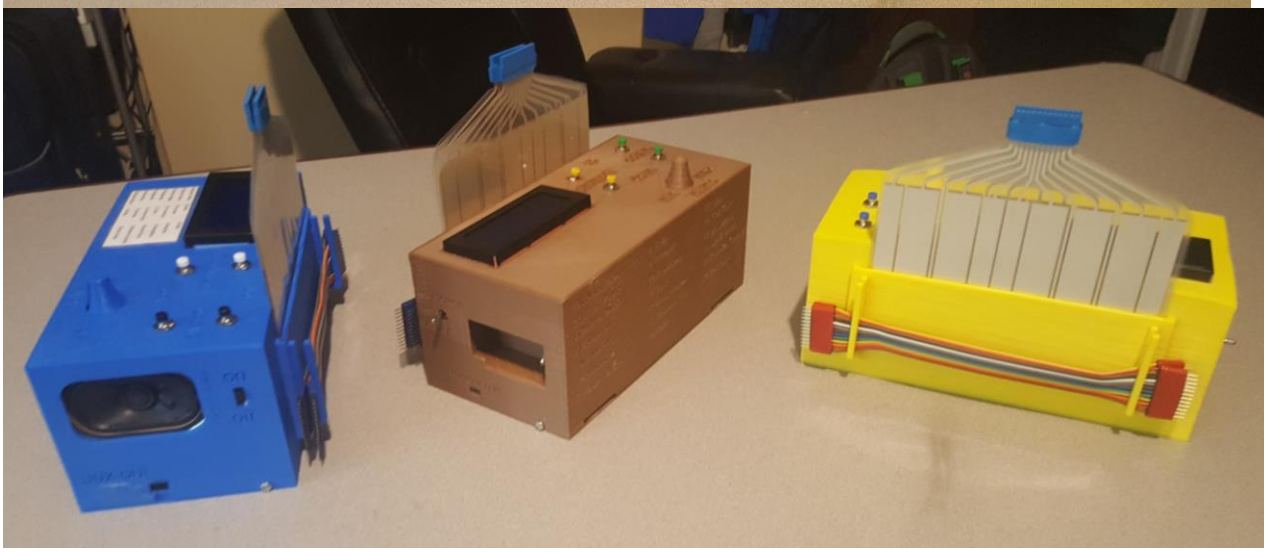
```
  }

  //LCD dipsplay for volume change
  lcd.setCursor(0, 2);
  lcd.print("Volume:");
  lcd.setCursor(8, 2);
  switch (volume11Conv) {
    case 0: lcd.print("|          |");
      break;
    case 1: lcd.print("|-         |");
      break;
    case 2: lcd.print("|--        |");
      break;
    case 3: lcd.print("|---       |");
      break;
    case 4: lcd.print("|----      |");
      break;
    case 5: lcd.print("|-----     |");
      break;
    case 6: lcd.print("|------    |");
      break;
    case 7: lcd.print("|-------   |");
      break;
    case 8: lcd.print("|--------  |");
      break;
    case 9: lcd.print("|--------- |");
      break;
    case 10: lcd.print("|----------|");
      break;
    default: lcd.print("|----------|");
      break;
  }
}
```

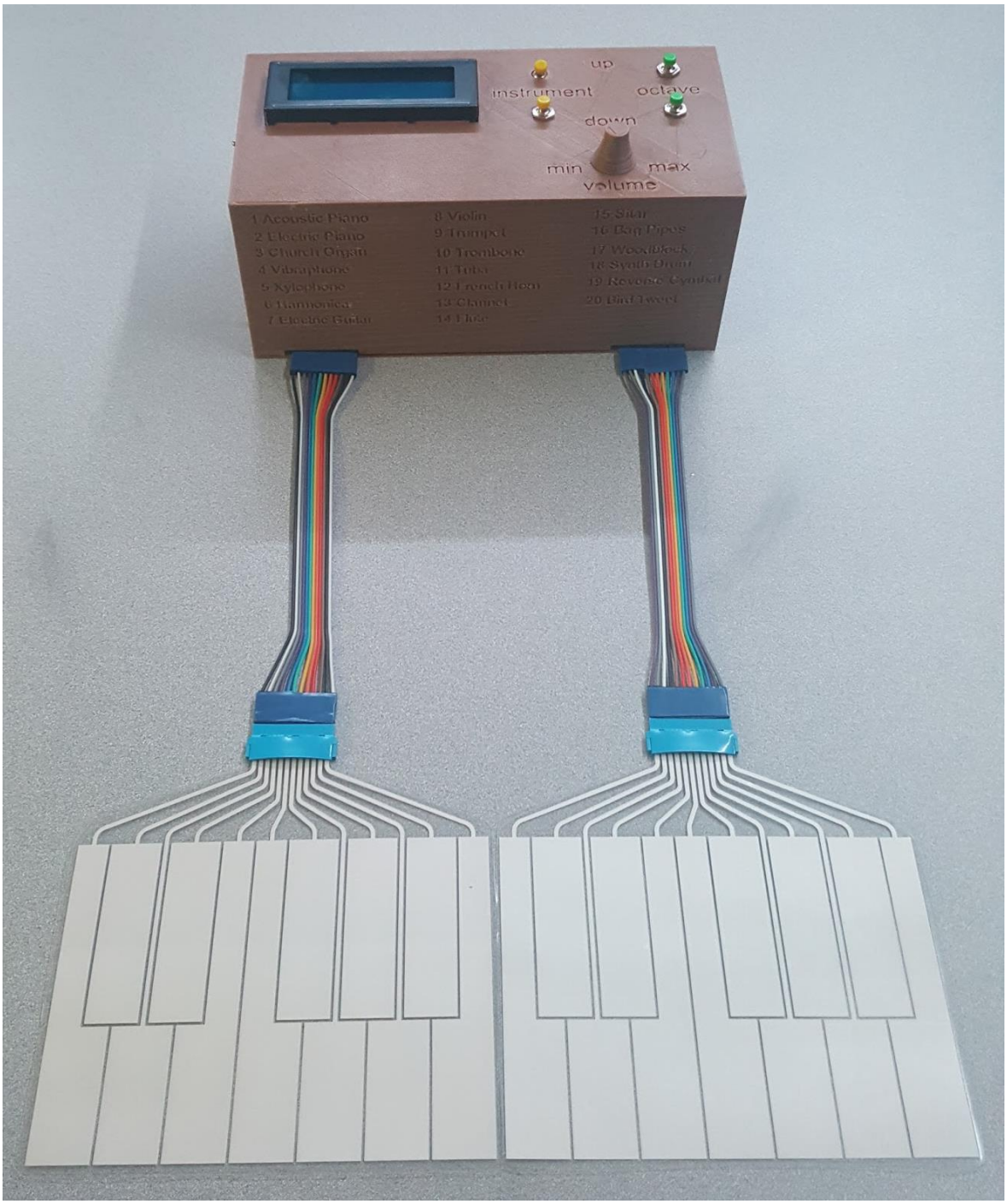
I Characterization of Sensors





J Final Products




up
instrument
octave
down
min
max
volume
1 Acoustic Piano
2 Electric Piano
3 Church Organ
4 Vibraphone
5 Xylophone
6 Harmonica
7 Electric Guitar
8 Violin
9 Trumpet
10 Trombone
11 Tuba
12 French Horn
13 Clarinet
14 Flute
15 Sitar
16 Bag Pipes
17 Woodblock
18 Synth Drum
19 Reverse Cymbal
20 Bird Tweet