Dual Glove Air Bass Guitar

By

Ying Chen

Pranathi Gummadi

Niranjan Jayanth

Final Report for ECE 445, Senior Design, Fall 2017

TA: Michael Genovese

13 December 2017

Project No. 7

Abstract

The Dual Glove Air Bass Guitar is a wearable device designed to be a novelty imitation instrument, able to perform the basic functions of a bass guitar. Musicians are often chained by the physical limitations of their instruments, which affect both the bulkiness of the experience as well as the mobility of a given instrument. The air bass will consist of two gloves fitted with sensors sending analog input into a control unit manipulating an audio output. The hardware-software workflow of the project proved to connect and work consistently; however, the limited accuracy and precision of our machine learning setup, consisting of a data acquisition training module and a support vector machine, neither had the volume nor quality of data to accurately predict the gestures involved in the play of a bass guitar.

Contents

1 Introduction

1.1 Objective

As a musician in the 21st century there are many technological tools available to assist the learning process. However, most of those tools are software based and are unsuitable for training muscle memory. This inability to convey, arguably, one of the most important parts of the musical experience impedes the musicians ability to practice relatedly on such devices. Therefore, individuals must rely on the traditional method of carrying fragile, cumbersome instruments to and from various locations. The act of repeatedly transporting a heavy instruments can potentially damage the instrument itself.

This project's objective is to remedy this problem by developing a portable electronic device that replicates the physical characteristics of the instrument without the physical medium. For this solution, the inspiration of was drawn from the concept of air guitar, a performance art in which an individual pretends to play an imaginary instrument with accuracy. In this specific case, we will be implementing a wearable device capable of generating audio output as a user manipulates notes using a glove to mimic real play.

Success in any musical endeavor is a mental mastery over an instrument or voice. The first mental challenge many musicians face is the playing of music without notes. The evolution of that mastery is the mental mastery of an instrument without the physical instrument itself. In one study meant to observe the effects of both mental and physical practice with regards to pianists, many advantages of mental practice were posited and agreed upon. The switching between mental and physical practice as a general practice strategy is suggested to be more effective than simple physical practice [1]. Since this product is a merging of mental practice (psychomotor understanding of the dimensions of a given instrument) and physical practice (auditory feedback upon practice), it offers a harmonious blend of the two practice disciplines. The market for such a virtual instrument thus far has existed solely in the VR realm, with products such as GloveOne. However, these products require an existing VR device, such as an OculusRift or an HTC Vive, which can set a musician back upwards of $500. The cumbersome and expensive nature of such a solution makes it far less appealing, concerns the air bass seeks to avoid with a portable hardware solution.

The other driving need for the air bass is transportability. Despite recent regulations helping traveling musicians, airlines are not required to store instruments in baggage closets, treating them just as other carry-on luggage, when in fact they are far more fragile than the average carry-on suitcase [2]. Further, it is potentially dangerous to store them in the cargo hold of a flight. The extra troubles that traveling musicians who feel a need to practice are thus rather onerous, and can be alleviated from having a portable version of their instruments. This also allows for quite enjoyable musical experiences that require limited effort to set up at any given time; this novelty is what the air bass seeks to achieve. Much of musicians muscle memory lies in their fingers, so the novelty of having a product that optimizes those abilities would be quite appealing.

1.2 High Level Requirements

1. The air bass must be able to sustain through 8 hours of battery-operated playing.
2. The air bass must have at least 90% accuracy of pluck attempts as played notes.
3. The air bass must have at least 90% pitch accuracy on left-hand finger placement.

2 Design

The air bass implements 20 different hand positions and enforces single note play. Musically this represents the range of one and a half octaves specifically between E1 and B2. At the core of the hardware is a microcontroller unit (MCU) which is responsible for data management, distribution, and processing. In terms of outputs, the MCU services the output module, containing a digital-to-analog converter (DAC) and an AUX jack. Overall power to all subunits will be delivered by the power module, which involved 2 voltage regulators. The flex sensors will be the primary sensory input into both the MCU, used for execution, as well as the training module, used for data acquisition into the machine learning aspect of the design.

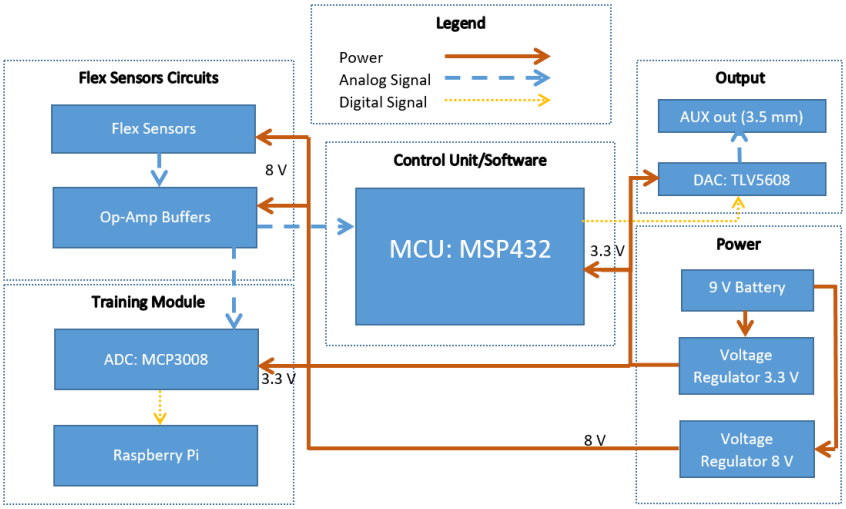


Figure 1: System Level Block Diagram

2.1 Physical Design

The physical design entails two gloves with sewn on flex sensors, a PCB in a holder strapped onto the left arm, and an attached perf-board with an op-amp circuit in a holder. It can be seen in Figure 2.

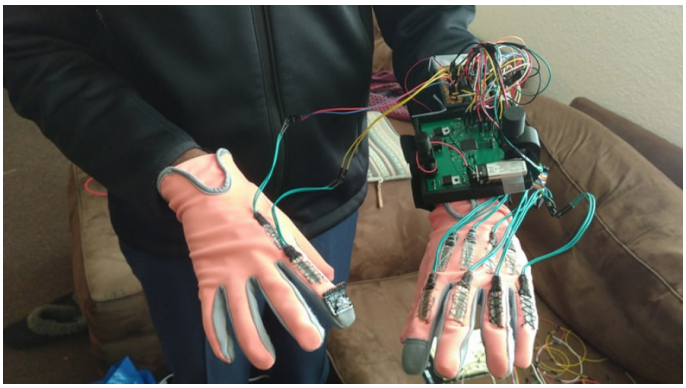


Figure 2: Final Physical Build

Figure 3: Brewer Science InFlect™ Flex Sensor

2.2 Hardware Block Design

2.2.1 Flex Sensors

The flex sensors module serves to provide analog input into the MCU, measuring the bend angles of the left and right hands The angles determine both the pluck state of the right hand as well as the finger state of the left hand. The flex sensors used are the carbon-based BrewerScience InFlect™ Flex Sensors [3], and can be seen in Figure 3. In total, there are 10 flex sensors implemented: 8 on the left hand and 2 on the right hand. On the left hand, the index, middle, ring, and pinky fingers have 2 flex sensors each which are attached to the middle knuckle and the base knuckle. On the right hand, only the index finger has flex sensors. The rationale behind this is that the left hand sees movement from those 4 fingers at those locations to create the specific note being played, and the right hand is enforced to pluck with only the index finger. Table 1 shows the relevant output given an input of left hand flex sensors, given O is open and the other letters represent the 4 strings. Since the sensors on the base knuckle are less significant, they are bundled together as one, and the individual fingers represent the middle knuckle sensors.

Table 1: Output Given Flex Sensor Input

Index	Middle	Ring	Pinky	Base Knuckles	Note
O	O	O	O	E	E1
E	O	O	O	x	F1
E	E	O	O	x	F#1
E	E	E	O	x	G1
E	E	E	E	x	G#1
O	O	O	O	A	A1
A	O	O	O	x	A#1
A	A	O	O	x	B1
A	A	A	O	x	C2
A	A	A	A	x	C#2
O	O	O	O	D	D2
D	O	O	O	x	D#2
D	D	O	O	x	E2
D	D	D	O	x	F2

Continued on next page

Table 1 – continued from previous page

Index	Middle	Ring	Pinky	Base Knuckles	Note
D	D	D	D	x	F#2
O	O	O	O	G	G2
G	O	O	O	x	G#2
G	G	O	O	x	A1
G	G	G	O	x	A#2
G	G	G	G	x	B2

Each flex sensor can be modeled as a variable resistor attached to a voltage divider circuit with $V_{cc} = 8V$ and with reference resistor R_{ref} chosen such that the change in V_{flex} is maximized. The output V_{flex} is then fed into an analog-to-digital converter (ADC) channel in the MCU. The schematic-level diagram of the flex sensors can be seen in Figure 4. Note: this implementation changes slightly with the addition of the op-amp buffer circuit (2.2.2).

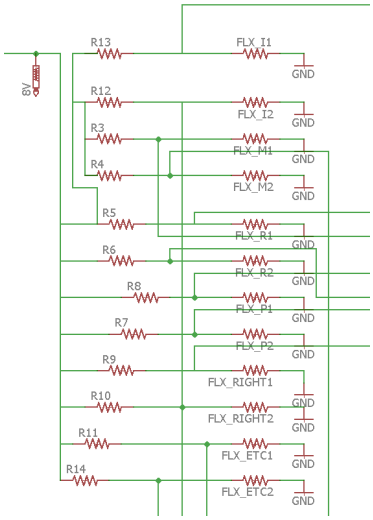


Figure 4: All Flex Sensors in Schematic

The choice of the reference resistor can be determined through modeling two voltage dividers. At maximum bend, the flex sensor can be modeled as one resistance, and at minimum bend, it can be modeled as another. As can be seen in Figure 5 from the InFlect^{TM} Flex Sensor Datasheet [3], the resistance of the flex sensor with respect to the bend angle is a linear relationship.

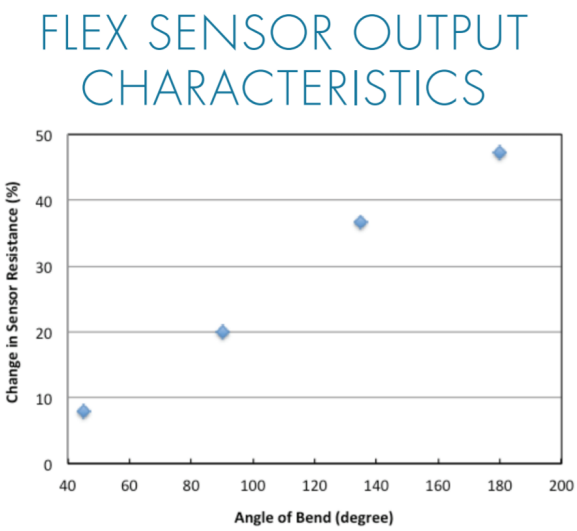


Figure 5: InFlect^{TM} Flex Sensor Resistance vs Bend Angle

Thus, the following equations can be used to determine the optimal reference resistor.

$$\frac{V_{cc} - V_{flex}}{R_{ref}} = \frac{V_{flex}}{R_{low}} \tag{1}$$

$$\frac{V_{cc} - (V_{flex} + x)}{R_{ref}} = \frac{V_{flex} + x}{R_{high}} \tag{2}$$

$$V_{flex} = \frac{V_{cc} R_{low}}{R_{ref} + R_{low}} \tag{3}$$

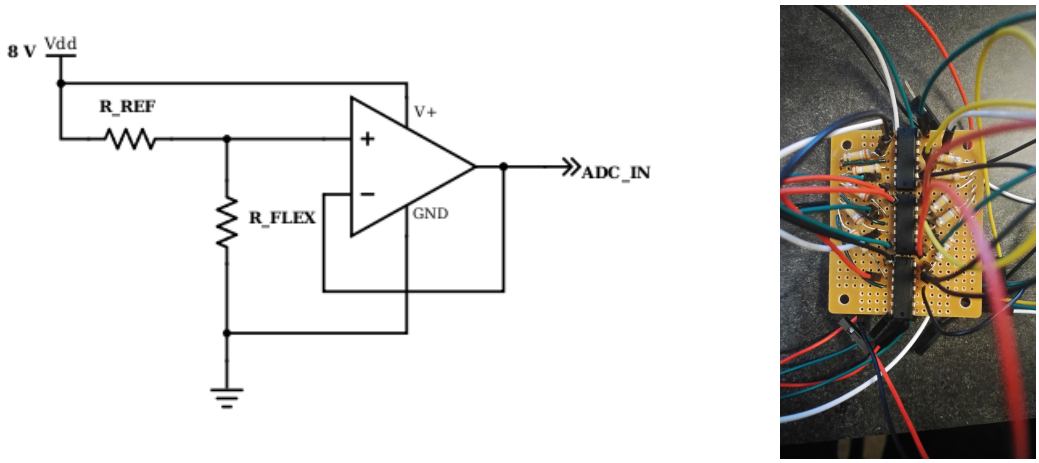
$$x = \frac{V_{cc}(R_{high} - R_{low})R_{ref}}{(R_{ref} + R_{low})(R_{ref} + R_{high})} \tag{4}$$

For example, using Using $V_{cc} = 8V$, $R_{low} = 250k\Omega$, and $R_{high} = 310k\Omega$, as well as the condition $V_{flex} + x < 3.25V$ (born from the MCU being unable to handle more than 3.3 V input into the ADC channel), R_{ref} can be determined to be $464.6k\Omega$. This is done such that x, or the change in the flex sensor voltage, is maximized.

2.2.2 Op-Amp Buffer Circuit

Looking back at Figure 5, the InFlect^{TM} Flex Sensors operate within an order of 10^2 $k\Omega$. Both the ADC in the training module as well as that used by the MCU itself requires input impedances of less than 1.5 kΩ. The way implemented by this design to remedy that problem is to use an op-amp buffer. This circuit connects the negative terminal of an op-amp to its output, allowing the output to match the input almost exactly, but have the output impedance match the negative-to-output impedance, which is only a few Ω. The details of this implementation can be seen further in the TL074ACN datasheet [4]. The circuit schematic for the op-amps can be seen in Figure 6a.

The perf-board solution implemented with 390 kΩ resistors can be seen in Figure 6b.



(a) Op-Amp Schematic

(b) Op-Amp Circuit Perf-Board Implementation

Figure 6: Operational Amplifier Circuits

2.2.3 Power

The design is powered by a 9V battery. This type of battery offered both the max voltage necessary for our voltage dividers (at least 8 V) as well as being common, easy to replace, and easy to attach to the PCB. Further, the choice of a battery over a power supply comes from the need for the overall product to be mobile.

Two voltage regulators are needed for this design. Both voltage regulators used were the LM317T model [5]. For digital devices such as the MSP432 (the MCU) as well as the DAC, a 3.3 V rail is needed to provide V_{cc}. The schematic for this regulator can be seen below in Figure 7.

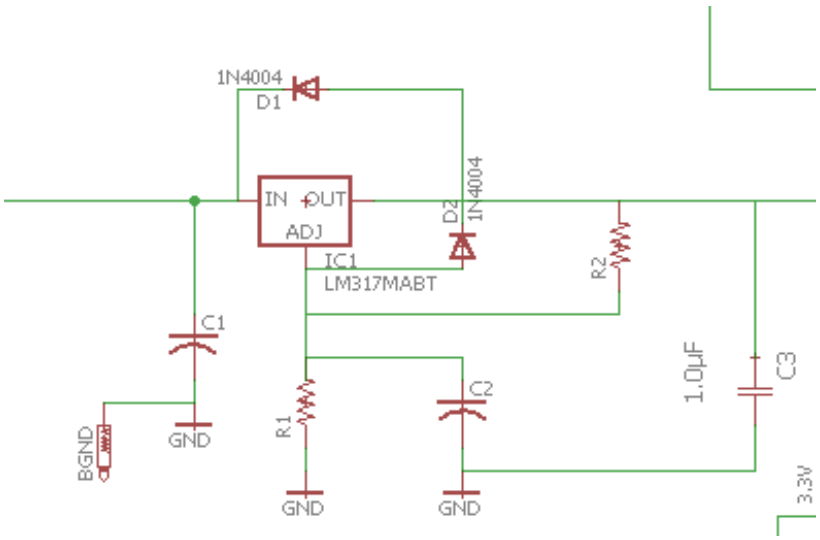


Figure 7: 3.3 V Rail Voltage Regulator

Using the guidelines in the LM317 datasheet [5], R_1 was chosen to be 1.5 kΩ and R_2 was chosen to be 1 kΩ. Further, the capacitors C_1, C_2, and C_3 were chosen to be 100 nF, 10 μF, and 1 μF, respectively. C_3 must also be a tantalum capacitor, to improve the transient response.

The second voltage regulator operates similarly to the first, but to offer the greatest possible range to the ADC coming in from the flex sensor voltage divider, it operates at an output voltage of 8 V. The schematic can be seen in Figure 8.

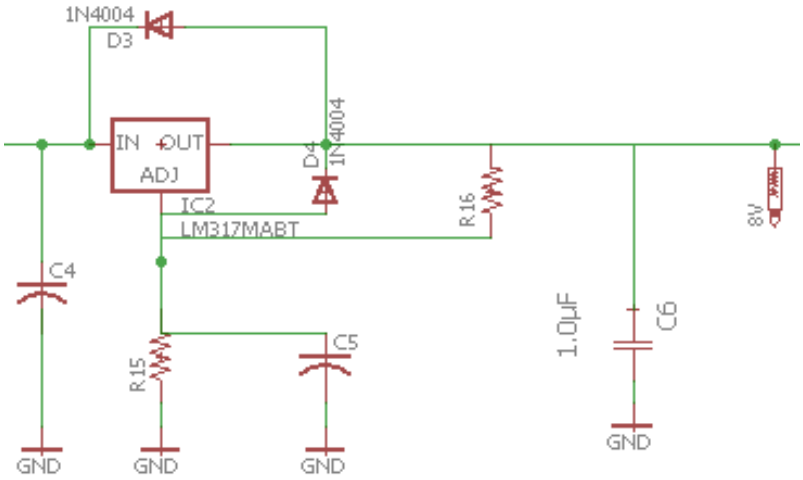


Figure 8: 8 V Rail Voltage Regulator

Again using the guidelines in the LM317 datasheet [5], R_{15} was chosen to be 1 kΩ and R_{16} was chosen to be 4.7 kΩ. Similarly, the capacitors C_4, C_5, and C_6 were chosen to be 100 nF, 10 μF, and 1 μF, respectively. C_6 must again be a tantalum capacitor, to improve the transient response.

2.2.4 Output

The output consists of a DAC and a 3.5mm stereo jack. This minimizes power usage by eliminating the need of having to power on-board speakers. The device communicates via a SPI interface. The DAC outputs a smoothed signal with a low pass filter.

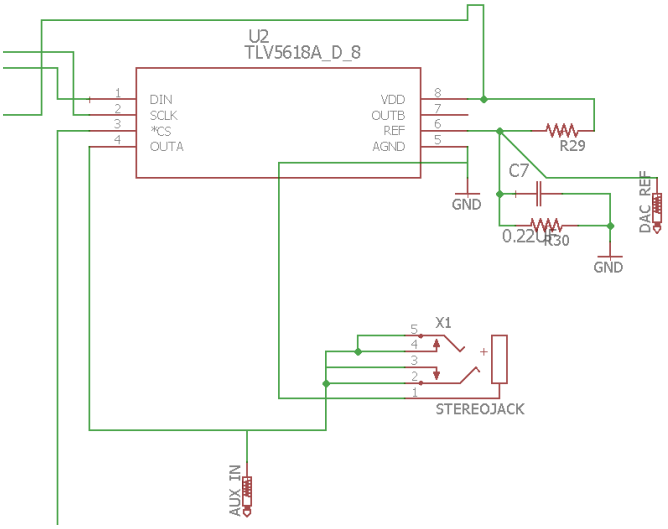


Figure 9: Digital Analog Converter

2.2.5 MCU

For our MCU, we used the MSP432 which is 100 pins and only available in a surface mount package. TI's MSP432 was selected due to our requirement of needing 10+ ADC channels and enough data storage to hold the machine learning model. The chip has 24 ADC channels as well as 256KB Flash memory and 64KB RAM memory [6].

Programming of the MSP432 is done through serial wire debug (SWD). The method for doing so is explained in the MSP432 LaunchPad User Guide, Section 2.3.4 [7]. Using another LaunchPad as a programmer, one can connect signals GND, SWDIO, and SWCLK to successfully load and debug code from a computer connected through USB to a Launchpad and an MSP432 chip on a custom PCB. The SWD pins are exposed on the PCB, as seen in the layout (Figure 18).

2.2.6 Training Module

The training module is composed of a Raspberry Pi which runs a python script that saves ADC values in text files with their labels. However, the Raspberry Pi does not have standalone ADC capability; thus, we used the MCP3008 8-channel ADC chip. This chip operates via a SPI interface. Since the MCP3008 only offers 8 channels, the training for the right hand and the left hand were done separately. The pin diagrams and application notes for the MCP3008 can be found in its datasheet [8].

2.3 Software

2.3.1 Energia

The project structure contains seven files: MCU_code.ino, model.h, model_left.c, model_right.c, pitches.h, svm-limited.h, and svm-limited.c

We first initialize the ADC pins; in our case we have 10 ADC pins, each corresponding to their individual flex sensor. The array adcPins contains the ADC identification and we iterate through all ten sensors to partition into a left and right data array. These arrays store the voltages seen during each iteration of sampling. The order of values is I0, I1, M0, M1, R0, R1, P0, P1, S0, S1, where I,M,R and P refer to index, middle, ring, and pinky fingers on the left hand, and S refers to the index finger on the right hand. The zeroth term refers to the middle knuckle and the first term is the base knuckle flex. We call each predict function with respect to left and right hand data to get the left hand note and the right hand pluck state.

With the information obtained through the LibSVM classifications we map them to their corresponding note. Timing for tempo can be controlled by editing parameters in the delay methods. Code snippets can be seen in Appendix C.

2.3.2 Machine Learning

In order to classify the 20 notes and 2 on/off states that could be potentially played by our gloves we utilized Support Vector Machines (SVMs). The SVM algorithm was provided by an already existing library called scikit-learn. To do this we passed in training data that we had collected ourselves. We collected a total of 200 samples from the left hand, which helps determine which note is being played and 50 samples from the right hand, which helps determine if a note is being played at all or not. This training data was passed into two separate SVMs one for the left hand and one for the right hand. The outcome was a classification model

that was used to predict the real time data from the gloves. The flow of the model to the MCU is depicted thoroughly in Figure 10.

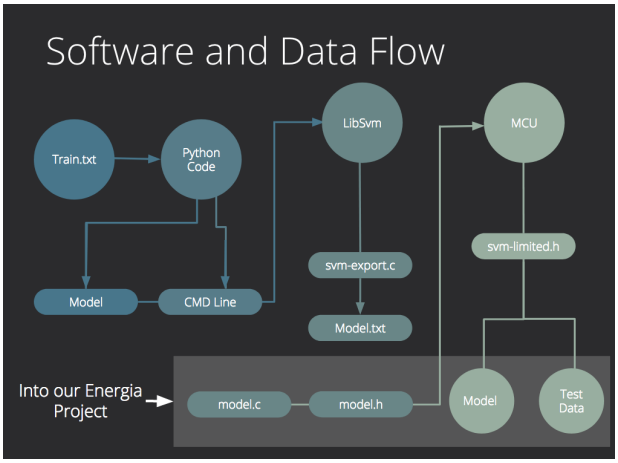


Figure 10: Software and Flow of machine learning code

When analyzing the results of the model we received a 99.97% accuracy rate for both the right hand and left hand. This accuracy is reflected in the confusion matrix, refer to Figure 11 generated to analyze what notes may have been confused with each other. However once we started classifying the real time notes from the gloves when connected to the MPS432, we were not successful. This error can be attributed to the fact that the range of data we received from the InFlect^{TM} Flex Sensors was not large enough to make accurate classifications between the 20 classes.

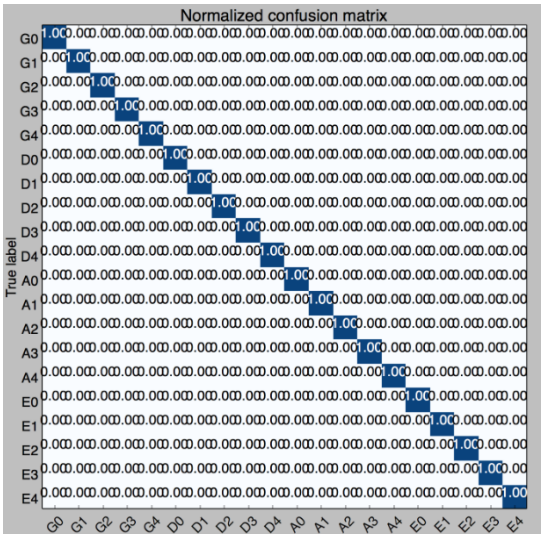


Figure 11: Confusion Matrix

3 Design Verification

3.1 Flex Sensors

Figure 12 shows the requirement verification of the flex sensors module. It sees a 0.1 V difference between key angles, a <50 mV wavering in voltage at a given angle, and, as seen in the rise and fall to steady-state before the "D" angle (at 8 s), a less than 100 ms time to steady-state. For detailed requirements/verification, refer to Table 5.

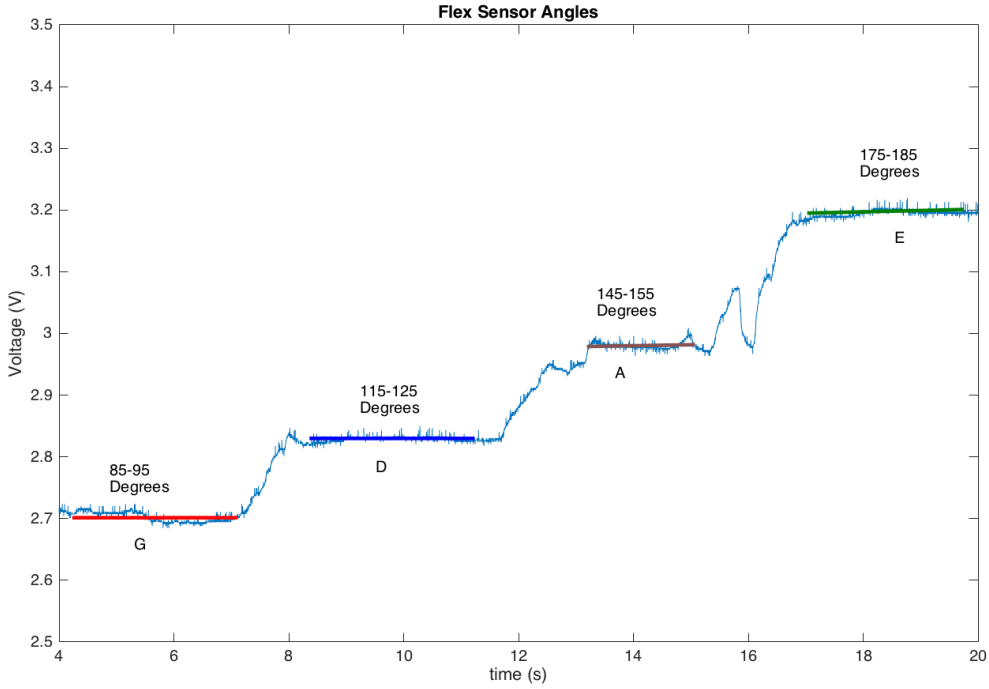


Figure 12: Design Verification of Flex Sensors

3.2 Battery

There are two requirements for the battery; one covers the entire circuit (Figure 13 and Figure 14), and the other only relates the the power consumption of the MCU (Figure). For detailed requirements/verification, refer to Table 5.

Figures 13 and 14 show the PCB without any flex sensors attached and with one flex sensor/op-amp combination attached, respectively. From these figures, the following calculations can be made to verify the first battery requirement (5).

$$I_{total} = 29.0 + 10 * (29.6 - 29) = 35mA \tag{5}$$

$$I_{max} = 40mA, Capacity_{battery} = 450mAH \tag{6}$$

$$Time_{estim}(hours) = 11.25 \tag{7}$$

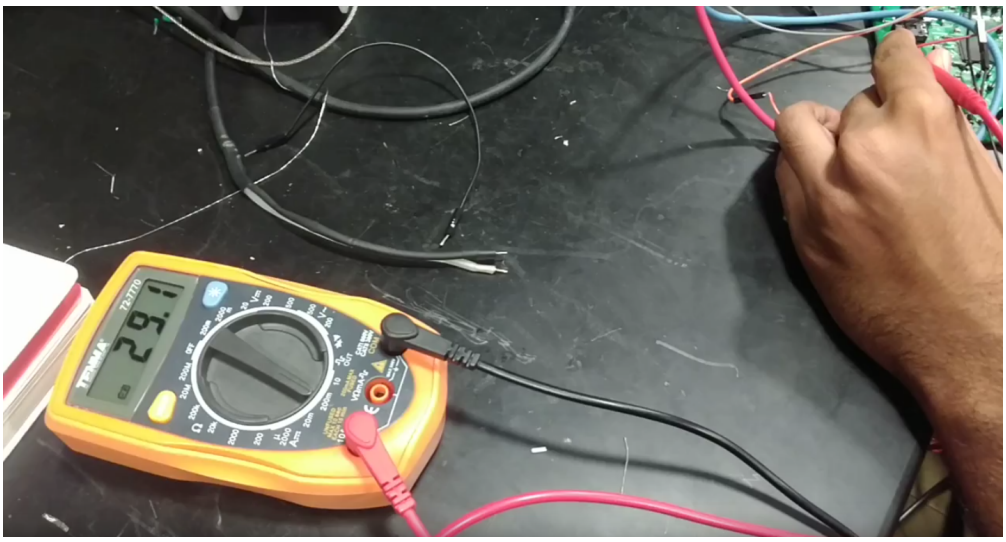


Figure 13: Design Verification of Battery Life (only PCB)

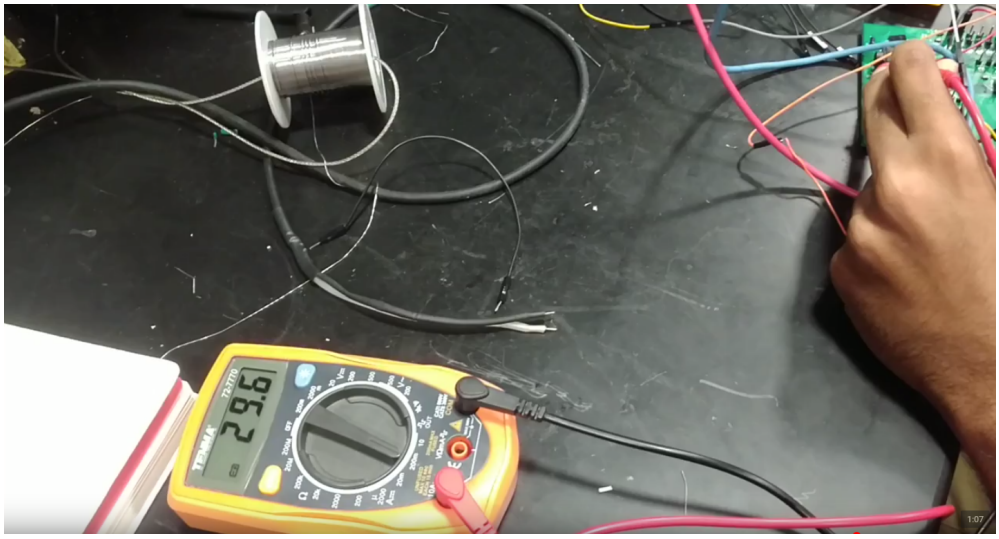


Figure 14: Design Verification of Battery Life (PCB with one flex sensor)

3.3 Voltage Regulator

Figure 15 shows the requirement verification of the voltage regulator module. The regulator, intended at 3.22 V, remains steady for over 15 seconds. For detailed requirements/verification, refer to Table 5.

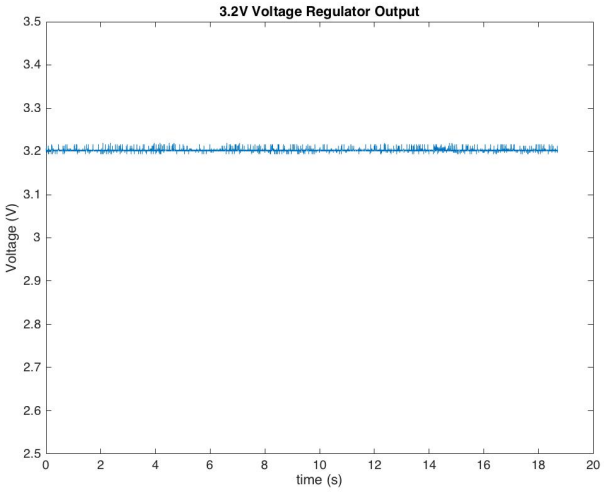


Figure 15: Design Verification of Voltage Regulator

3.4 Op-Amp Buffer Circuit

Figure 16 shows the requirement verification of the op-amp module. The circuit output, intended at 2.97 V, remains steady. For detailed requirements/verification, refer to Table 5.

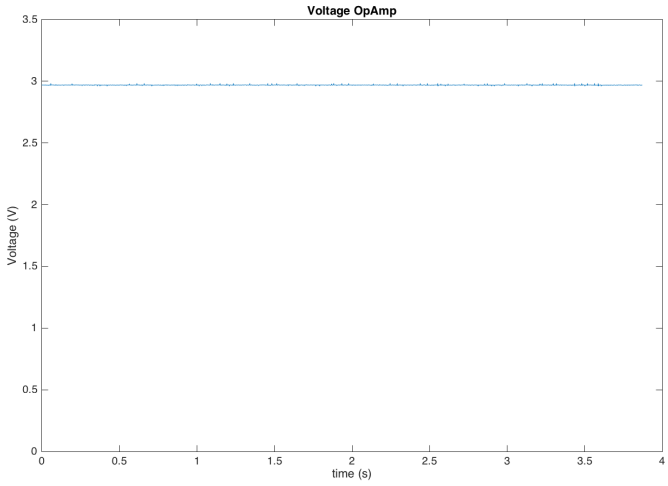


Figure 16: Design Verification of Op-Amp

4 Cost

4.1 Parts

Following is a starter table for parts costs. Add cell contents as well as rows and, if necessary, columns. Update the table number according to your sequence. Note that columns 1 and 2 are set up for centered text (words) and columns 3∼5 (numbers) are set up for right-alignment so that decimal points align.

Table 2: Parts Costs

Part	**Manufacturer**	**Retail Cost ($)**	**Quantity ($)**	**Total ($)**
Microcontroller MSP432	TI	28.63	1	28.63
Micro Tubing	Uxcell	6.12	1	6.12
4-in Heat Shrink	Uxcell	9.49	1	9.49
Photoresistor	eBoot	5.99	1	5.99
5mm LED	CO-RODE	6.32	1	6.32
Accelerometer Breakout Board	Adafruit	7.47	2	14.94
Audio Jack	Adafruit	6.54	2	13.08
Battery Clip	ECE Supply Center	0.25	1	0.25
9V Battery	Energizer	1.34	1	1.34
Battery PCB Mount Holder	ECE Supply Center	1.44	1	1.44
Gloves	Menards	5.44	2	10.88
Passive Elements	Merlintools	17.99	1	17.99
MCP3008 ADC	Adafruit	6.94	2	13.88
3D Printing Filament	Hatchbox	22.99	1	22.99
Epoxy Glue	Bob Smith and Industries	8.76	1	8.76
Electrical Tape	Scotch	6.21	1	6.21
PET Plastic Sheet	Small Parts	8.40	1	8.40
LED Strip	LEDJUMP	15.49	1	15.49
PCB	PCB Way	7.80	5	39.00
Brewer Science InFlect Flex sensor	Brewer Science	Samples	30	Samples
Parts Total				217.32

4.2 Labor

Table 3: Labor Cost

Team Member	Hourly Rate	Total Hours ($)	Expense Multiplier ($)	Total Cost ($)
Ying Chen	33.07	130	2.5	10,747.75
Niranjan Jayanth	33.07	130	2.5	10,747.75
Pranathi Gummadi	33.07	130	2.5	10,747.75
Labor Total	99.21	390	2.5	31,003.13

Table 4: Total Costs

Grand Total ($)
31,220.45

5 Conclusion

5.1 Accomplishments

In conclusion, we were able to achieve a prototype of our air guitar by using the MSP432 launchpad instead of the PCB. We were able to produce audible sound using the 3.5mm AUX jack. Motion in the gloves and the voltage representations were accurate compared against equipment probed values. In addition, the machine learning algorithm was able to make classifications for both the right and left hand separately.

5.2 Uncertainties

The overall uncertainties with our project lies in the durability of the build and the reliabilities of our support vector machine. Our classifications were strongly skewed toward a single note, G2. Likewise, the right hand classification displayed a strong preference for "on". To fix this issue, we must retake training data, accounting for the variation in different hand positions for the same note.

Another roadblock we had in training was trying to offset the continuous physical strain on the device. As we continued to collect data the values would often become unstable due to the loose wires and connections.

5.3 Future work

In the future, we would like to acquire more training data and create new machine learning models for the SVM. Another feature that should be implemented is NFC, RFID or Bluetooth capability; this will allow the user to actually enjoy the freedom of an "air guitar". By omitting the bulky wire joints we will also eliminate all primary sources of physical strain on the gloves. This will contribute to more accurate training, testing, and predictions. Currently, our sound output is a PWM wave which means it is hard on the ears. To provide better quality audio output instead of using a PWM we could map the classes to .wav files.

Another feature for future work could be the addition of all the bass guitar notes, since currently the design only supports first position. Looking even further, a future application of this project could be the expansion of these gloves to other instruments, such as an acoustic guitar with multi-note functionality, or classical stringed instruments such as violin and cello. Since the bones of the product lie in the use of various sensors for musical gesture recognition, the possibilities are limitless.

5.4 Ethical Considerations

The most relevant safety concern with wearable devices is the physical contact between electrical equipment and the users body. Therefore, insolation and build must be of the utmost quality to prevent potential harm. A possible physical safety concern could be the potentially heavy weight of the glove causing repeated stress and strain on fingers and wrists. Continued use could result in medical conditions such as carpal tunnel or wrist tendonitis. We will execute the project while in agreement with IEEE's code of ethics obligation number 9, to avoid injuring others, their property, reputation, or employment by false or malicious action [9]. We understand these physical concerns and have taken as many precautions as possible with regards to them.

References

[1] D. D. Coffman, "Effects of mental practice, physical practice, and knowledge of results on piano performance," *Journal of Research in Music Education*, vol. 38, no. 3, p. 187, 1990.

[2] "U.s. department of transportation issues final rule regarding air travel with musical instruments," Dec 2014. [Online]. Available: http://www.transportation.gov/briefing-room/us-department-transportation-issues-final-rule-regarding-air-travel-musical

[3] B. Science, "Inflect flex sensor," 2016.

[4] T. Instruments, "Low-noise jfet-input operational amplifiers," 2005.

[5] STMicroelectronics, "Lm217, lm317," 2014.

[6] T. Instruments, "Msp432p401r, msp432p401m simplelink mixed-signal microcontrollers," Sep 2017.

[7] T. Instruments, "Msp432p401r simplelink microcontroller launchpad development kit (msp-exp432p401r)," Oct 2017.

[8] Microchip, "Mcp3004/3008," 2008.

[9] "Ieee ieee code of ethics." [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

Appendix A Requirement and Verification Table

Table 5: System Requirements and Verifications

Requirement	Verification	Requirement Verified?
1. The flex sensor must produce voltages distinguished by 0.1 V when bent at 90°., 120°, 150° and 180°. 2. The flex sensor voltages at a given angle must not differ more than 50 mV. 3. The flex sensor voltages must approach their correct value range for each angle within 100 ms.	1. Build a test circuit using a flex sensor and a 390 kΩ reference resistor. 2. Use MCU to supply 8 V to a test circuit to measure the voltage through the flex sensor at key angles. 3. Extract data into MATLAB. 4. Check for 0.1 V distinguishment on MATLAB generated Voltage graph between key angles. 5. Check for a maximum 50 mV variation within a key angle range on MATLAB generated Voltage graph. 6. Check for the time between the key angle steady states on MATLAB graphs.	1. Yes 2. Yes 3. Yes
1. The battery will be able to provide 50 mA at rated voltage for at least 8 hours.	1. Measure current draw from entire running circuit. 2. Compare current draw to battery capacity at given current. 3. Confirm that $\frac{Capacity}{CurrentDraw} > 8$	1. Yes
1. The step-down power of the regulator will not exceed 0.2V below or above intended voltages at 50 mA.	1. Connect output of the regulator to a data acquisition module, and run acquisition for at least 15 seconds. 2. Extract data into MATLAB. 3. Check for steady output at a given intended voltage.	1. Yes
1. The controller must read voltages at least 10-bit ADC resolution. 2. The controller must draw less than 4 mA.	1. Debug ADC channels to see a 10-bit resolution. 2. Probe current consumption of PCB between 3.3 V and GND.	1. Yes 2. Yes

Continued on next page

Table 5 – continued from previous page

Requirement	Verification	Requirement Verified?
1. The op-amp buffer circuit will output a value within 5% of intended value, linearly scaled between 1.5V and 2V. 2. The buffer will have an output resistance of less than 1 kΩ.	1. Connect output of the op-amp to a data acquisition module. 2. Extract data into MATLAB. 3. Check for steady output at a given intended voltage. 4. Probe resistance of op-amp between negative terminal and output.	1. Yes 2. Yes
1. The AUX output will be able to produce audible sound at each note that would be played.	1. Connect audio input to test code that plays a melody going up the scale of all 20 notes. 2. Observe audible sound at each note.	1. Yes

Appendix B PCB Schematics and Layout

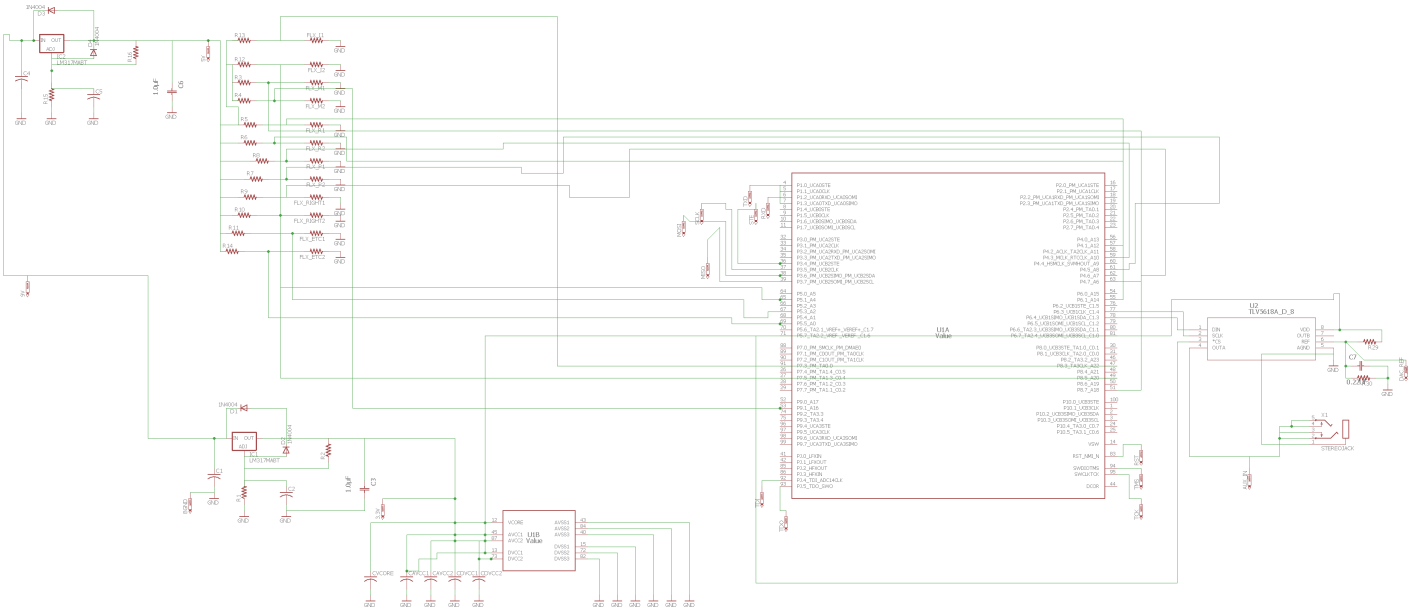


Figure 17: PCB Schematic

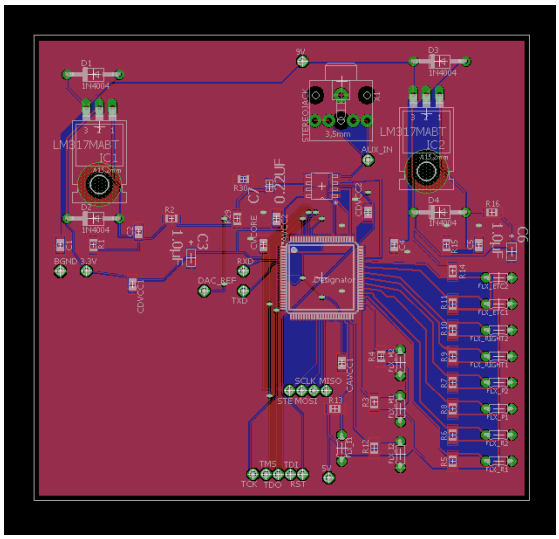


Figure 18: PCB Layout

Appendix C Code Snippets

```
extern "C"{
  #include "model.h"   //get the name from Pran
  #include "svm_limited.h"

};

#include <SPI.h>
#include <msp432.h>

void setup() {
    //set up the flex reading
    Serial.begin(115200);


}


```

Figure 19: MCU code Part 1 - Setup

```
void loop() {
   //read and store flex sensors
   for (int i = 0; i < pinCount; i++) {
     int sensorValue = analogRead(adcPins[i]);
     double v = sensorValue * (5.0 / 1023.0);
     flexVoltage[i] =  v;
   }
   for(int i = 0; i < 8; i++){
     left[i] = flexVoltage[i];
     Serial.print(i);Serial.print("Voltage:"); Serial.println(left[i], 6);

   }

   right[0] = flexVoltage[8];
   Serial.print("0");Serial.print("Right Voltage:"); Serial.println(right[0], 6);

   right[1] = flexVoltage[9];
   Serial.print("1");Serial.print("Right Voltage:"); Serial.println(right[1], 6);


  double * pointerleft = left;
  double * pointerRight = right;

  int note = svm_predict(&svm_left, pointerleft);  //get this function from Pran
  int onoff = svm_predict(&svm_right, pointerRight);  //get this function from Pran
  Serial.println(onoff);
```

Figure 20: MCU code Part 2 - Reading from ADC and Classification

```
/************************************************
 * Public Constants
 ************************************************/

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
```

(a) Header Files of Pitch Mapping

```
62   if(onoff == 1){
63     switch(note) {
64      case 1:
65        p = NOTE_G2;
66        Serial.print("Class:");
67        Serial.print("G0    ");
68        Serial.print("Note:");
69        Serial.println("G2");
70        break;
71      case 2:
72        p = NOTE_GS2;
73        Serial.print("Class:");
74        Serial.print("G1   ");
75        Serial.print("Note:");
76        Serial.println("GS2");
77        break;
78      case 3:
79        p = NOTE_A2;
80        Serial.print("Class:");
81        Serial.print("G2   ");
82        Serial.print("Note:");
83        Serial.println("A2");
84        break;
85      case 4:
86        p = NOTE_AS2;
87        Serial.print("Class:");
```

(b) MCU code Part 3 - Class to Note matching

Figure 21: Pitch Mapping Code

```
204       default:
205         p = 0;
206         break;
207       }
208     delay(1);        // delay in between reads for stability
209     int noteDuration = 1000/4;
210     tone(A4,p,noteDuration); //TODO: CHANGE PIN # TO SOMETHING ELSE
211
212     // to distinguish the notes, set a minimum time between them.
213     // the note's duration + 30% seems to work well:
214     int pauseBetweenNotes = noteDuration * 1.30;
215     delay(pauseBetweenNotes);
216     // stop the tone playing:
217     noTone(A4);
218     delay(pauseBetweenNotes*1.75);
219
220   }
221   else if(onoff==2){
222       Serial.println("open");
223       p = 0;
224       delay(250*1.3);
```

Figure 22: MCU code Part 4 - Sound Production