12/9/18
Megan Heinhold, Joseph Kim, Carl Wolff (T-Pros)
meganjh3, jdk5, cwolff2

Final Report ECE 110 H

Introduction

Problem Description

When playing trumpet (or any musical instrument for that matter), a key part in achieving mastery of the instrument is understanding intonation and especially tuning. While there are many methods of achieving correct tuning externally (ex: a tuner), one must find the ability to tune by ear. Since we as musicians are by no means masters of tuning, we decided to build a device that would allow us to train our tuning by ear, to further our musical ability. To achieve this, the device would need to react to the note the user is playing, and sustain the correct intended pitch.

Design Concept

This device will "listen" to the user play a note, and identify the frequency (pitch) of said note. Then, the device will check what "valves" (buttons) are pressed. We will define ranges of frequencies in which there is only one correct note corresponding to each valve combination. Therefore, the device will be able to identify the correct, intended note, as long as the user has the right valve combination (not usually an issue, especially for more advance musicians) and is playing a pitch fairly close to the correct one. Since the input and output sounds will be "the same" but slightly off, a dissonant waveform is audibly generated which the user can hear, and then adjust their playing accordingly.

Analysis of Components

Characterization of each sensor

1. Pushbuttons

As part of our inputs, we used normally open pushbutton switches. These switches essentially break the circuit when they are not pressed, allowing no current to flow through. For this project, we needed switches that would work with digital pins, meaning they could be grounded or powered at all times. We addressed this issue with the circuitry below, and will discuss later in our design considerations.

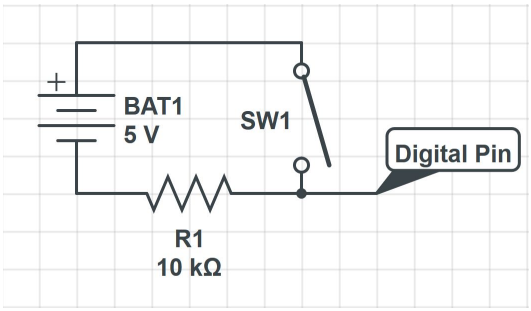


Figure 1. Circuit diagram of switches

2. SparkFun sound detector

For this project, we used the SparkFun Sound Detector SEN-12642. This audio sensing board has three outputs: an audio output as well as a binary (gate) and analog amplitude (envelope) output.

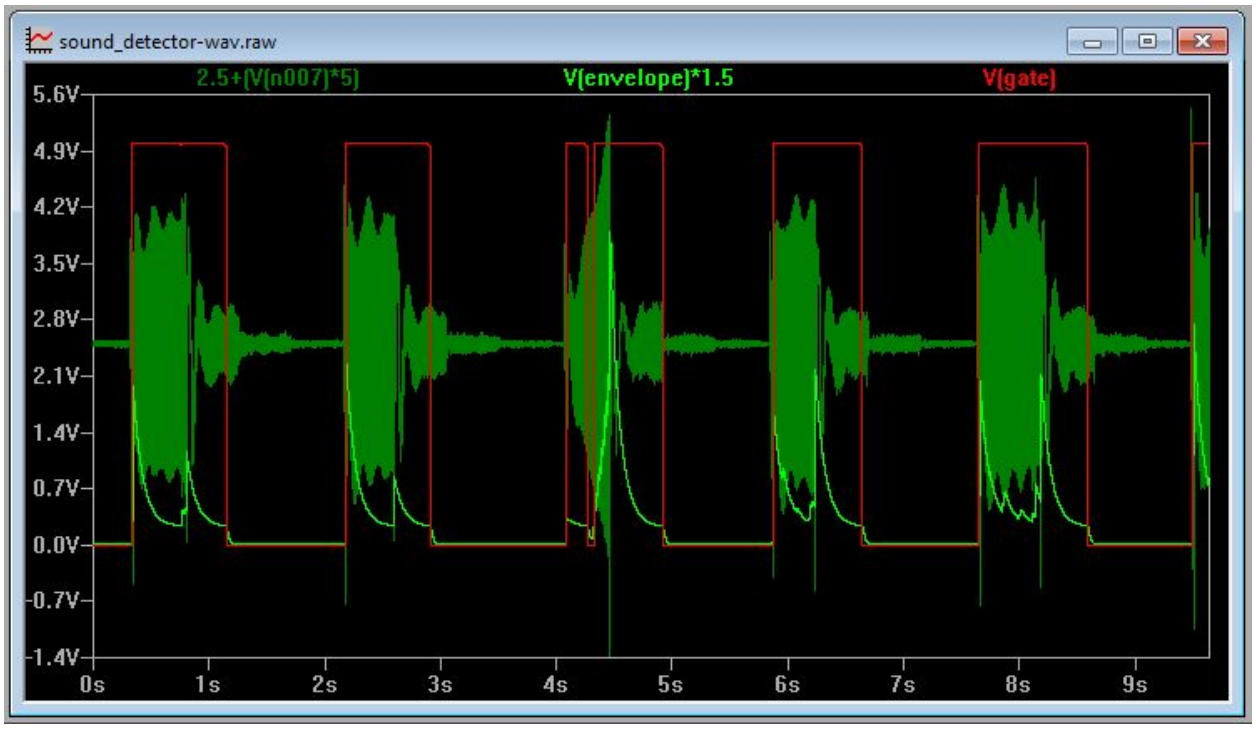


Figure 2. The output of the sound detector in response to sound pulses, retrieved from *https://learn.sparkfun.com/tutorials/sound-detector-hookup-guide/all*

Initially, we tested each of these outputs with the oscilloscope to determine what data we could use. When we first characterized the audio output, we saw a rather messy signal on the oscilloscope. The microphone outputs through the audio pin a waveform resembling the waveform of the input sound. This means it will output a sum of the harmonics, resulting in a relatively dirty sinusoidal signal. Although the signal within each period is unclean, the frequency of the signal as a whole will be very close to the frequency of the dominant sound in the environment. The square wave we used as a reference translated very well in terms of frequency through the microphone. We did in fact observe the signal reacting to both increases in

intensity (volume) of incoming sound and change in frequency. The amplitude and frequency of the signal both changed on the oscilloscope accordingly. Since the frequency was accurate, and most applicable to our project, we decided to use this output.

Design Considerations

One significant part of this project was creating a product that was somewhat user friendly, that somewhat felt like a trumpet and acted similar to it. After all, what good is a trumpet simulation that feels like an ECE project? For this, we placed our buttons a realistic distance away from the circuitry of our project, and made sure that the speaker was close to the user so they could hear it. In addition, we positioned the microphone and output in opposite direction so as to minimize feedback loops. This was not too big of an issue, especially since we discovered that the microphone only picks up fairly loud sound within a few centimeters.

We modified the pushbutton switches to act as pseudo SPDT switches in that it causes the Arduino digital pin to swap between a grounded and a powered circuit (HIGH and LOW signals). We achieved this function by connecting the ground of the pushbutton to both the digital pin and a resistor connected to ground, as shown below. This was necessary because the digital pins on the Arduino must be grounded or powered at all times to ensure accurate readings, and avoid floating voltages.

We decided to only use the AUDIO output from the SparkFun board because, as discussed before, we needed the frequency value of the incoming sound. The rest of the noise on top of the fundamental frequency made it difficult for the Arduino to find a steady, accurate pitch, and the final product still has some issues with maintaining a steady pitch. When deciding where to place the microphone on our project, we knew from testing that the microphone did not pick up sound accurately from more than a few centimeters away. Therefore we decided to place the sound detector chip as close as possible to where sound would be on this device, which is the mouthpiece of the instrument.

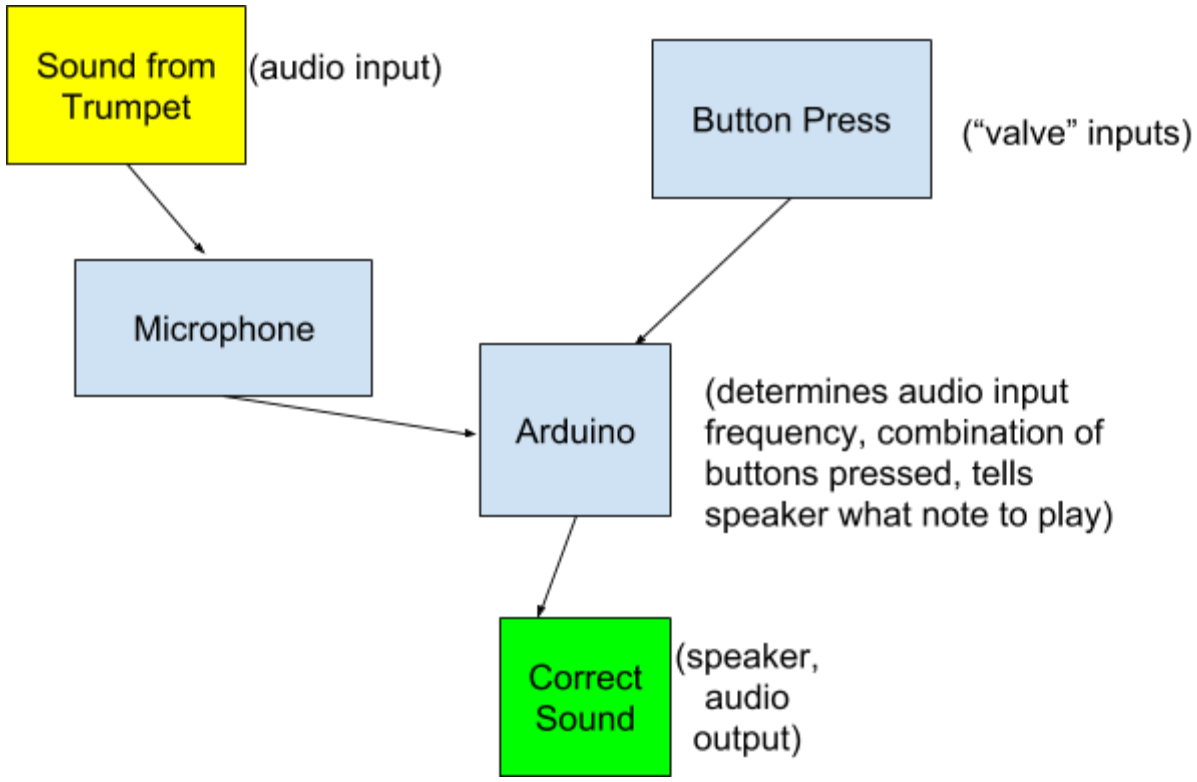
We used the Arduino heavily in this project to help us analyze the inputs from the microphone. The analog pins were used as the input for the sound detector, and with it we could discern the frequency of the pitch that was being played. The digital pins were used to read the state of the pushbuttons. Most of this analysis was done in the code we created for this project. The starting point was to find existing code that could do the calculations required to derive the frequency from the voltage signal output by the audio pin. We found an open source code at https://www.instructables.com/id/Arduino-Frequency-Detection/ and from there further developed our code. We added variables for our digital inputs, the pushbuttons. Then, we broke the main code into two main operations: checking frequency and checking button states. The Arduino checked the incoming frequency first, and then identified which of our identified ranges it fell into. These ranges were low C to F, F# to B flat, and B to C (covering one full octave). The reason the ranges get smaller as one goes up the scale is due to the harmonics of the instrument, in that valve combinations start to repeat more quickly as notes get higher. After checking which of these ranges the played frequency fell into, the Arduino checked the buttons. Since each valve

combination corresponds to exactly one note in each range, the output note was determined here and sent to the speaker. It should be noted, then, what the device does when the inputs do not match a note. First, if the frequency fell outside of our ranges (usually large spikes in frequency readings due to the instability of the microphone), the speaker stops playing so as to not confuse the user with wrong notes. In addition, if the frequency falls into a range but there is no note corresponding to the valves pressed, the speaker also stops playing. This would signal to the user that they are using the incorrect valves for that note.

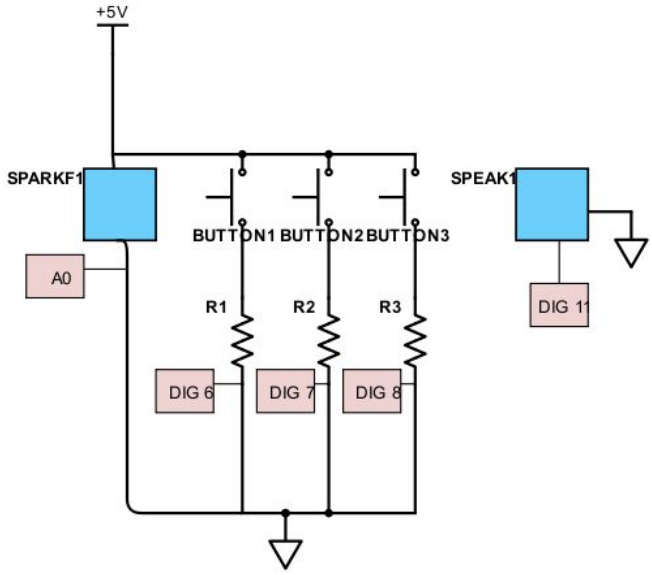
A small speaker was used as the final output of our device. Based on the reading of the sound detector and the combination of button/valves being activated, the Arduino would output the corresponding sound through the speaker, using the tone function.

Design Description

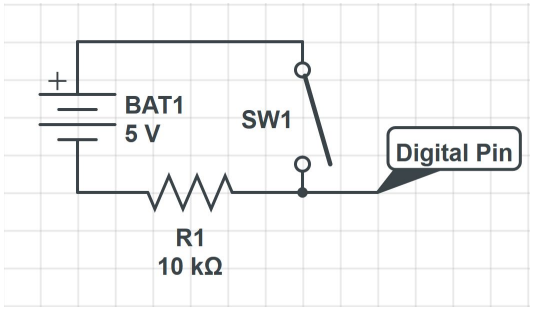
Block Diagram

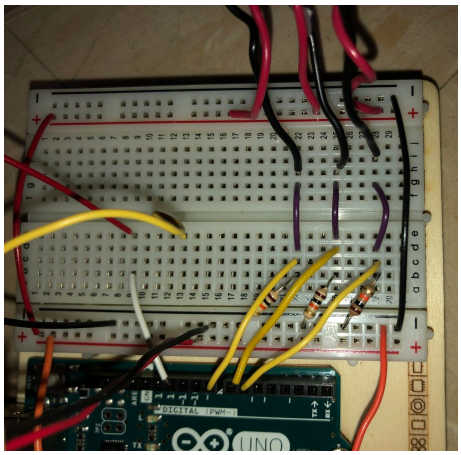


Circuit Schematics



Pushbuttons:





Above is the circuit schematic for one of our pushbuttons. The same circuitry was repeated for all three and connected to digital pins 6, 7, and 8, respectively. When the pushbutton is pressed, the digital pin of the arduino is connected to power (current takes the path with least resistance) and the pin reads a HIGH value. When the button is not pressed, the pin is connected to ground through the resistor and the pin reads LOW.

Physical/Mechanical Construction:

Our physical construction consisted of two main components: the breadboard with the Arduino and a small acrylic tube which held the buttons. The tube served as a mock instrument, giving the user something to hold while interacting with the switches. Using a soldering iron, small holes were burned through the acrylic, into which we attached the buttons. The wires that connected to the buttons were fed through the tube to be connected to the breadboard with the rest of the circuitry.

Conclusion

We took away many valuable lessons from this project. For one, none of us had coded with Arduino to this extent before, and learning what the board was really capable of doing will be helpful for future projects. One obstacle we did not foresee due to our inexperience was with the buttons: we bought SPST pushbuttons, not realizing that digital signals had to be powered or grounded at all times (avoiding floating voltages). However, we were able to fix that issue by applying things we had learned in our lab course. Also, it was pretty interesting that some of the most recent material we covered in ECE 110 (signal processing) related to our project (frequency from samples). We also got a very helpful review in how to solder components correctly.

All in all, we were successful in creating a device that could read an input frequency and button combinations and analyze them together to create a useful output in developing pitch accuracy and musicianship. Further development could include a visual display (perhaps through LEDs or a light strip) that would display whether the input pitch was relatively flat, sharp, or correct. The ranges in our code could be expanded to include more octaves. More work could also be done on the coding aspect to make the input frequency reading more accurate and the output buzzer noise faster in responding to changes in the pitch or switch combination. We could

also experiment with different types of sound detectors in order to get a more accurate, stable reading in frequency.