ECE 196

# Servo Spectrum Analyzer

Team 4

Ramon (RJ) Dioneda
Kevin Piedad
Zhengyang (Jay) Ruan
Brad Stevenson

## Table of Contents

## Description

Spectrum analyzers are devices that take in audio signals and process the frequencies of sound coming in and display the amplitude of the different frequencies.  Following this idea, we created a spectrum analyzer that displays frequency bands within the range of human hearing using linear actuators to represent the frequency amplitudes. We wanted to create a device that allows users to get a visual representation of music using moving parts.

Using an Arduino, our device accepts incoming audio signals through the analog port and processes it using an FHT algorithm. The audio signal is pre processed with a circuit that gives all negative voltage values a DC offset. The algorithm takes in 256 samples of raw data at a time and processes those produce an array of 128 units of data as the other 128 units are redundant in an FHT. The array represents 128 frequency bands and their respective amplitudes. These amplitudes are mapped to servo angles. The servo will move to low angles for low amplitudes and move to a higher angles for high amplitudes. Gears mounted on the servos intertwine with gear racks to serve as a linear actuators that move up and down depending on the amplitude of the frequency detected. Each actuator represents a different band of frequencies, with the left-most servo representing the lowest band and the right-most servo representing the highest band.

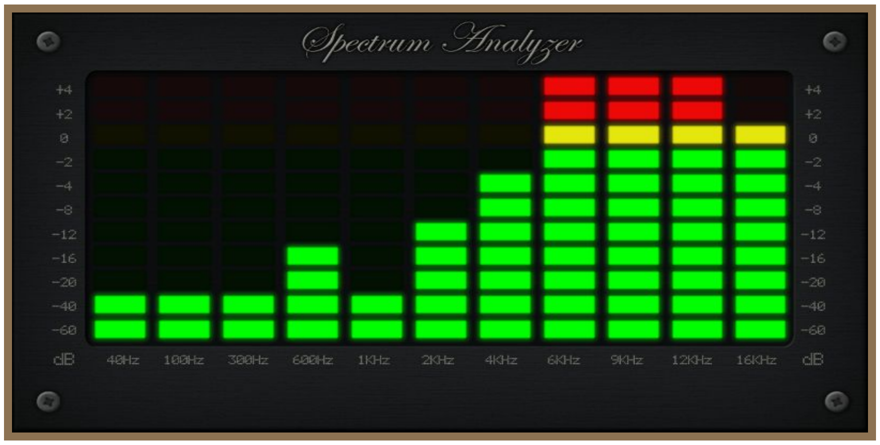


**Figure 1**

In Figure 1, the spectrum analyzer displays the amplitudes of 11 frequency bands in decibels. Our device is similar to this but uses 8 rods (gear racks) attached to gears to represent the amplitudes of 8 frequency bands.

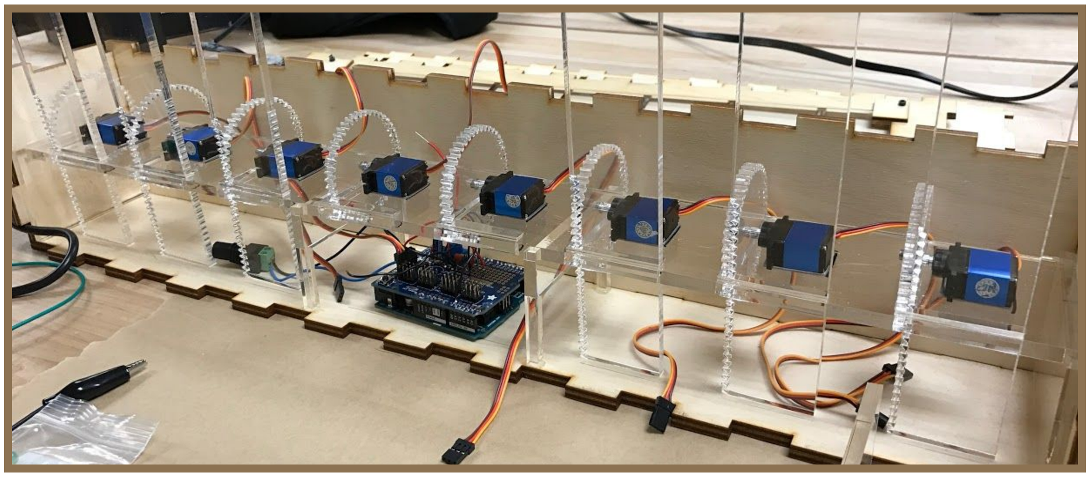


**Figure 2**

The housing design of the device is equally as important as the code that goes into it. The housing is designed such that a middle rack (shown in Figure 2 holding the servos) placed above the Arduino gives the servos room to spin gears large enough that the gear racks will move up and down. The middle rack contains L shaped holes that hold the servo gears as well as the racks. An upper panel is placed on top of the box with holes for the racks. The middle rack rests on top of four stands. The servos are carefully calibrated to not move the racks too far down that they hit the bottom of the box not too far up that they move the racks off the gears. Audio, USB, and power ports are cut into the back of the box.

## Learning Objectives

The primary learning objectives of our project include practices common in mechanical, electrical, and computer engineering. This project allows students to learn new skills as well as apply and build on skills they already have.

Getting into more detail, one of the first learning objectives involves Inkscape to design the housing as well as the gears. The correct sizing of the housing and gears are important since both will affect one another. Having a smaller, sleeker design will result in smaller gears and less movement, while opting for bigger gears and more movement will require a larger housing design. Anything designed in Inkscape also requires knowing how to use the laser cutter.

A large portion of this project is related to electrical engineering. More specifically, it involves breadboarding, soldering, and signal processing. Knowing how to design and build a circuit is key to anything involving electronics. Fitting the circuit onto a PCB requires soldering, a skill commonly used with circuit building. The main concept behind our project is signal processing, which is helpful in learning how sound signals are processed and modified.

Lastly, coding is required to make the project actually work. Any device containing a computer requires some coding in order to make the device work as desired. This project provides the opportunity to work with a microcontroller (Arduino) that will require coding (in C) so that the Arduino can make the servo motors function properly.

## Cost and Budget

| Material | Amount | Cost | Product URL |
|---|---|---|---|
| Arduino | 1 | 18.82 | https://www.amazon.com/Arduino-Uno-R3-Microcontroller-A000066/dp/B008GRTSV6 |
| Servos | 8 | 95.6 | https://www.adafruit.com/product/2307?gclid=CjwKEAjwvYPKBRCYr5GLgNCJ_jsSJABqwfw7smQf9I0vrCy8VajfnsD_WsOPD60p3tCs4sX-cu_tkhoCSGjw_wcB |
| Servo Shield | 1 | 17.5 | https://www.adafruit.com/product/1411?gclid=CjwKEAjwvYPKBRCYr5GLgNCJ_jsSJABqwfw7XO4KfjahmB6fBHeKE3YsX7UBU-sztMfZqg3f_65uLRoC39nw_wcB |
| LED | 8 | 0 | - |
| Resistors | 4 | 0 | - |
| Capacitor | 2 | 0 | - |
| 5V, 10A Power Supply | 1 | 21.99 | - |
| Aux Cables | 1 | 3 | - |
| Aux Cable Splitter | 1 | 3 | - |
| ¼” Wood | - | 0 | - |
| ¼” Acrylic | - | 0 | - |

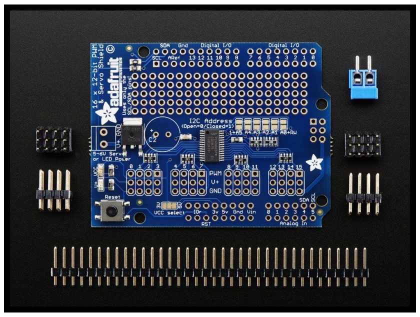
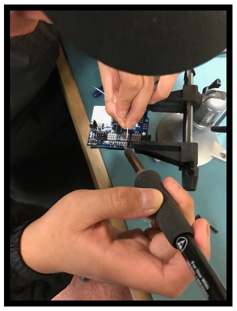
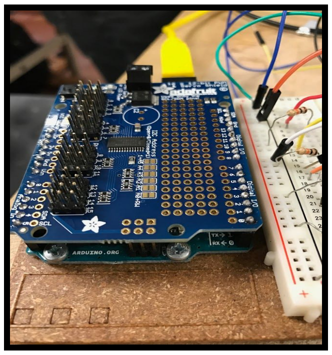
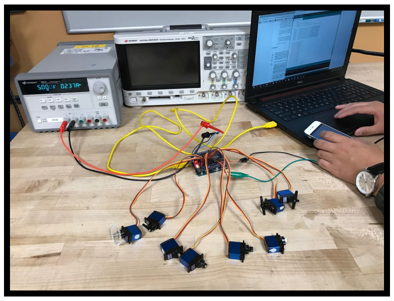
## Build Steps

### 1) Solder, attach, and install PWM-Servo shield:

The PWM-Servo shield comes with a set of header pins that need to be soldered on to the board in order to interface with the Arduino. The pins had to be soldered on (with the female pins facing downward) in order to properly mount to the Arduino UNO. There were two sets that needed to be soldered: the Arduino mount, and the PWM servo pins (PWM, V+,GND). The shield then was attached, with each header pin from the Arduino nested into the female pins from the shield.

Next, we soldered the power terminal provided with the PWM-shield kit in order to utilize the external power supply need for proper functionality (the servos are powered by a separate 5V power supply). We then connected a female jack for the 5V power supply to the shield, properly powering the device.

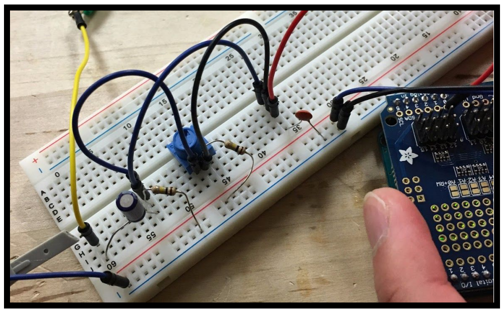
In order to test the PWM driver library provided by Adafruit for use with the shield, we attached all 8 servos to the board, and uploaded simple individual and group servo sweeps. We then calibrated the PWM lengths for the servo's minimum and maximum range, to later map to frequency analysis data.

**2) Create preliminary audio circuitry for proper frequency analysis on a breadboard:**

In order for the Arduino to effectively, we had to implement a DC offset in order to prevent any negative voltages from contaminating the FHT. The circuit had to be assembled, and tested on a breadboard. We used an oscilloscope and alligator clips to test and confirm that the circuit was behaving properly.

The circuit can be found under the 'Schematics' section.



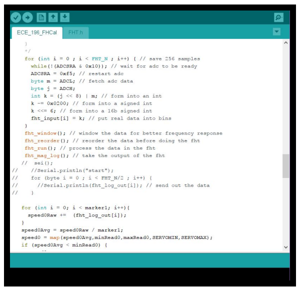
**3) Install circuit onto PWM shield:**

In order to be efficient with our wiring and housing space, it was best to solder the preliminary circuit directly onto the PWM shield, as nearly half of the space on the board is dedicated to a prototyping area. In order to install the circuit, it had to be prepared for the transfer by drafting a plan to put each component in its place, while still being in reach to solder to the rest of the circuit. We then used an oscilloscope to test that the circuit was functional after having been soldered to the PWM shield.

**4) Program and implement FHT (FFT substitute):**

An FHT (Fast Hartley Transform) is a transform similar to the Fast Fourier Transform, in the sense that it takes in a signal or waveform, and outputs the amplitudes of the signal's frequency components. This information is what drives the heart of the project, as it acts a prism to defract audio into its harmonic components. The provided library is a bit complicated, so we took some time understanding the functions and runtime options so that we can utilize it effectively.

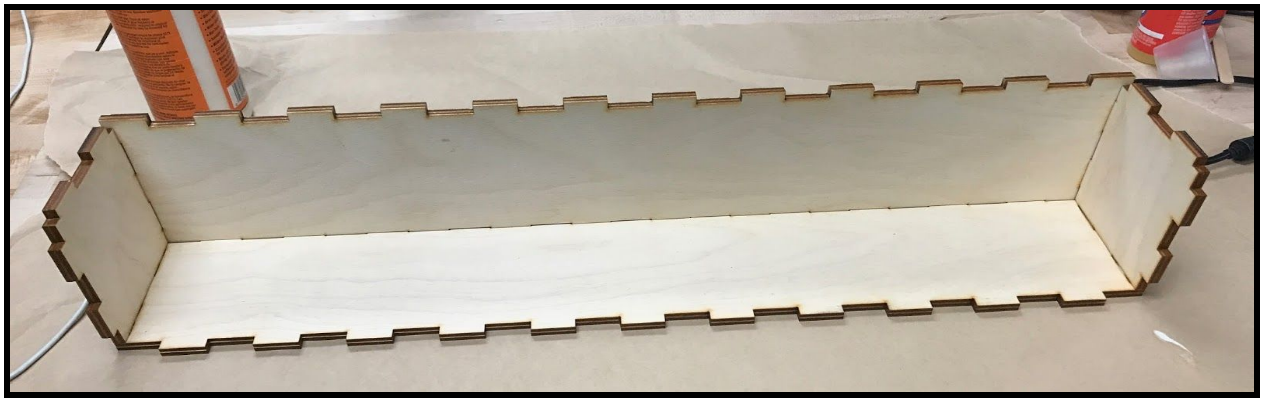
We eventually decided on using the LOG_OUT process, as it quantized frequencies into bins that were effective for meaningful servo data. We took the average of certain chunks of FHT data (frequency amplitudes are stored in an array of 128 unsigned ints) and averaged them to get specific average amplitudes of certain frequency blocks to then send to the servos for linear expression.

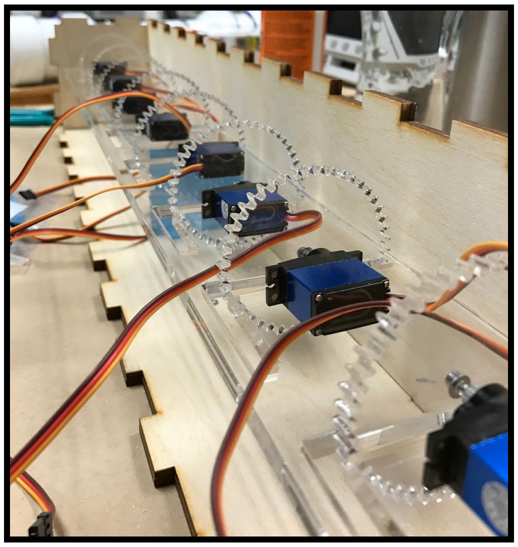


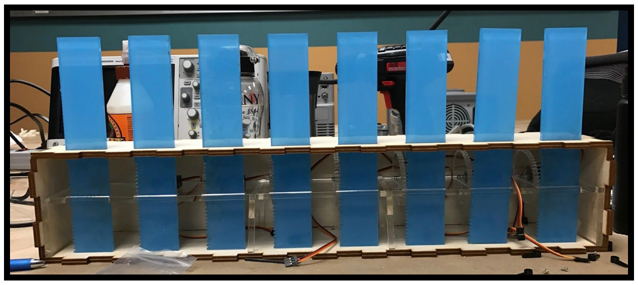
```
    }
    */
    for (int i = 0 ; i < FHT_N ; i++) { // save 256 samples
      while(!(ADCSRA & 0x10)); // wait for adc to be ready
      ADCSRA = 0xf5; // restart adc
      byte m = ADCL; // fetch adc data
      byte j = ADCH;
      int k = (j << 8) | m; // form into an int
      k -= 0x0200; // form into a signed int
      k <<= 6; // form into a 16b signed int
      fht_input[i] = k; // put real data into bins
    }
    fht_window(); // window the data for better frequency response
    fht_reorder(); // reorder the data before doing the fht
    fht_run(); // process the data in the fht
    fht_mag_log(); // take the output of the fht
  //  sei();
//    //Serial.println("start");
//    for (byte i = 0 ; i < FHT_N/2 ; i++) {
//      //Serial.println(fht_log_out[i]); // send out the data
//    }

  for (int i = 0; i < marker1; i++){
    speed0Raw +=  (fht_log_out[i]);
  }
  speed0Avg = speed0Raw / marker1;
  speed0 = map(speed0Avg,minRead0,maxRead0,SERVOMIN,SERVOMAX);
  if (speed0Avg < minRead0) {
```

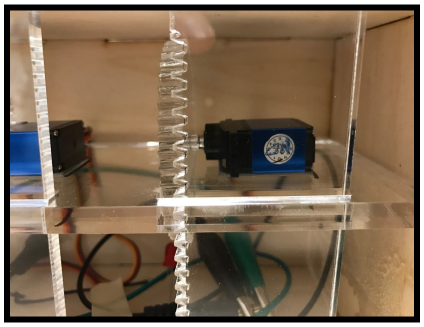
**5) Design housing, gears, and physical braces for motors and 'pistons' in Inkscape:**

Once the FHT library was correctly implemented, we began the design and construction of the physical housing, gears, braces and pistons. We used Inkscape to create 2D models of the housing faces, with slots on the top for the pistons to travel through, as well as a window cut for a view of the machine's inner workings. This was then cut with the laser cutter and ¼" Wood boards. We then designed the window, center brace, supporting braces, gears and pistons, to be later cut with the laser cutter and ¼"acrylic sheets. We then assembled the pieces together.









We then cut holes in the back for the two 3.5mm audio jacks (in/out), the Arduino power and the PWM shield power. After mounting the components into place on the central brace, we then began the final calibration of the servos.

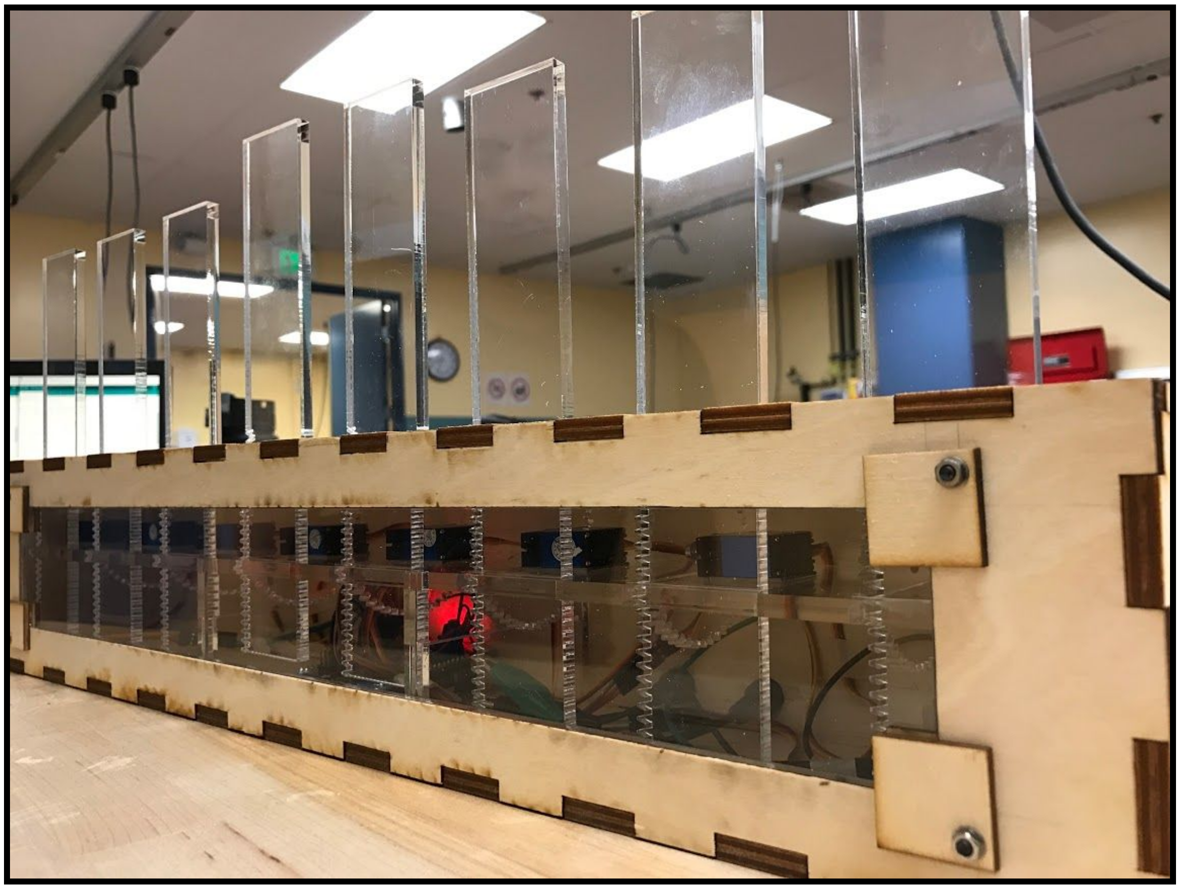
See 'Schematics' section for images of housing and gear design.

**6) Calibrate PWM for all 8 servos:**

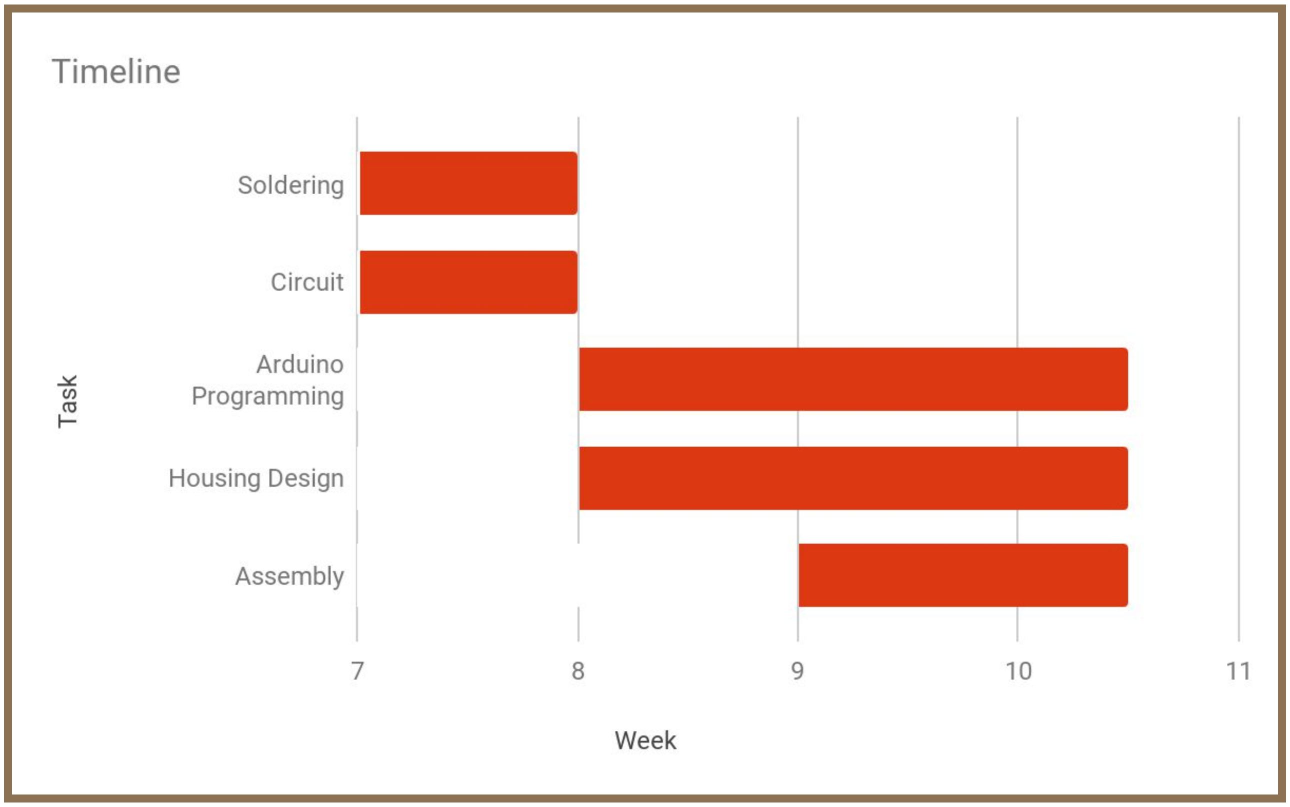
Once the pistons were fitted into place in the housing, we began the final calibrations for the pistons, servos and FHT analysis. To do this, we set the servos at their minimum height, loaded the pistons when they are in transit, and set upper limits on the servos as to not shoot the pistons out of the box. We then calibrated the servos to make meaningful changes in the slight changes in frequency amplitude data. We used the map() function to do this efficiently. Afterwards, we glued the sides of the box together to finish the construction of the device.

**7) Test and troubleshoot analyzer/finishing touches:**

The last step was to thoroughly test the analyzer by playing a multitude of songs, to tweak any of our code to make it run faster, or be more believable when listening to it. We also added a string of LEDs to go inside the housing, though this is cosmetic and is not required for the analyser to function.



## Timeline



**Total Time: 4 Weeks**

Week 7 (5/15-5/19):

- Solder PWM shield pins
- Build and install preliminary circuit for input audio

Week 8 (5/22-5/26):

- Calibrate servo functionality (range of movement, parallelization, data assignment)
- Begin laser-cutting for housing and gears
- Implement Adafruit’s PWM and Open Music Labs’ FHT Arduino libraries

Week 9 (5/29-6/2):

- Test PWM/FHT libraries with real-time audio
- Further servo calibration for frequency analysis
- Housing, gear and brace construction/assembly

Week 10 (6/5-6/7):

- Thoroughly test and troubleshoot servo functionality
- Assemble and install housing LEDs

# Challenges

### Arduino Programming

One of the biggest challenges we faced was working on the programming. The Arduino libraries we worked with offered some documentation, but for the most part we had to figure out the libraries for ourselves.

In the FHT library, we had to account for random noise causing the servos to move erratically, gathering meaningful data, and combining two libraries to work with each other. We solved the noise issue by gathering data from an analog port that wasn't being used and using the difference between our real data and that empty port. Meaningful data was hard to extract because a spectral analysis program showed us that some frequencies were present at all times even without sound/noise, some frequency bands started out with base values higher than others, and some frequencies never seemed to show amplitude differences. We remedied this with much trial and error, testing for good base values that we could map to servos and decreasing the range of possible amplitudes to maximize servo movement. When we tried combining the two libraries originally, they had clashed due to the fact that the shield required new API to make the servos function. Originally, we had used a simple servo function, servo.write(angle) given from Servo.h in the Arduino built in libraries to just move servos to a specified angle. With the PWM library from Adafruit, we could not use this function to control 8 servos. The PWM library required use of finding appropriate values for the servo's minimum and maximum pulse width to map to angles. The sample code for a full sweep also misled us because it showed us only able to use for loops to send servo pulse widths. We fixed the library issue with much trial, error, and research from others who have used the two libraries, and we eventually came to a compromise between the best of both.

## Housing Design

Designing the housing and gears for our device was rather difficult. Being that the design of our structure was an original idea, we didn't have any examples to reference. We had to think far ahead and make sure our measurements were precise to avoid any future complications. One of the hardest parts was figuring out the best possible layout for the servos in order to conserve space. In addition we needed to design a middle layer brace to keep the servos suspended and in place while also being able to support the vertical rack gears that would be moving up and down. The middle layer L-hole measurements had to be exact as the rack gears had to sit next to the gears perfectly in order for the racks to stay stuck to the teeth. We needed to be aware of how much the rack gears would move so that they would stay in place.

## Calibration

Calibrating the servo to move to very specific points was very difficult due to the small size of the housing and the nature of pushing the gear racks into the holes. The gear racks had to be pushed in while sweeping the servos because forcing the racks into the gears with unpowered servos caused them to reset to zero once they were powered, shooting the rack gear out of place. Once they were in place, finding a good index of the maximum pulse width was trial and error, and for some gears it was different because the Arduino was in the way.

## Learning Experiences

One of the key learning experiences that the team gained as a whole was working with the arduino to send meaningful data to the servos. Working with the Arduino was new to most of us and being that it was used to control our device, it was a huge learning experience. Even though others might have had some experience with the Arduino, using it to process signals and output meaningful data to the servos was new.

Another learning experience was designing the housing and gears for our device. None of us had ever worked with gears so it was a little difficult understanding the measurements on Inkscape and sizing them correctly so that the gear would properly move the rack gear. As for the housing, we had some prior experience designing boxes which was very helpful. However, our housing design required us to think ahead and make several measurements to ensure that our housing would be as sleek as possible. We also had to think of a way to support the servos and rack gears so that everything would function as planned.

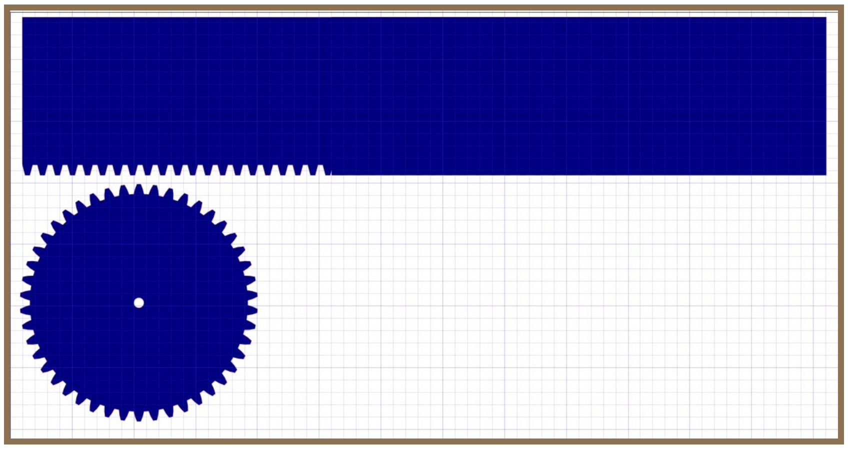
We want others to learn some basic circuit design, signal processing, Arduino programming, and housing design. The level of difficulty of our project is doable for someone with little experience, but also provides someone with experience an opportunity to build on skills they already have.

## Schematics

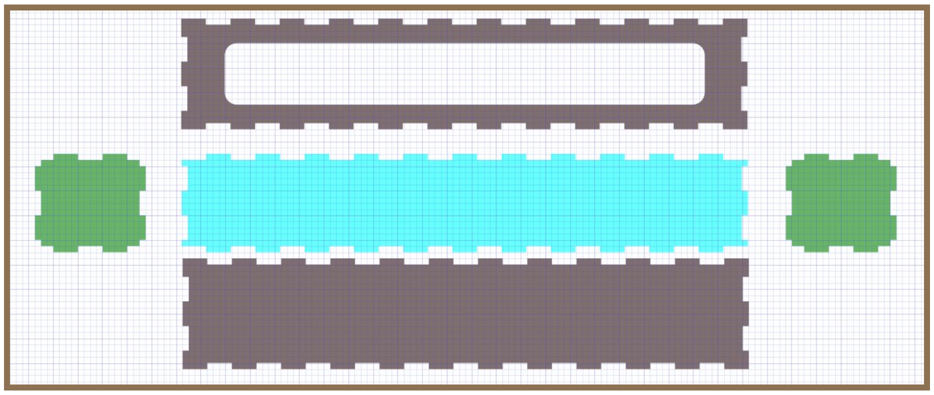
### Arduino Code - In folder

### Housing Schematics

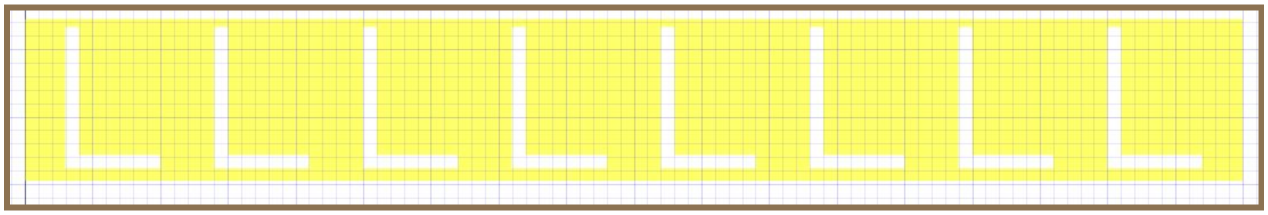
Gear/Gear Rack Schematic



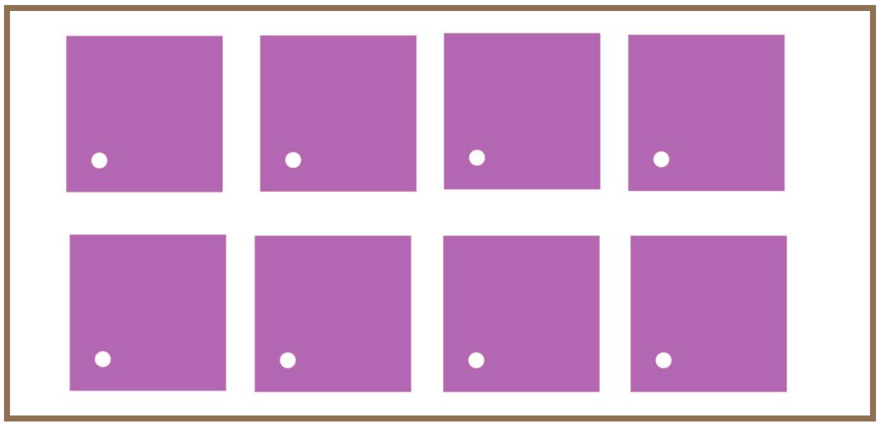
Box design, with cut out for acrylic panel



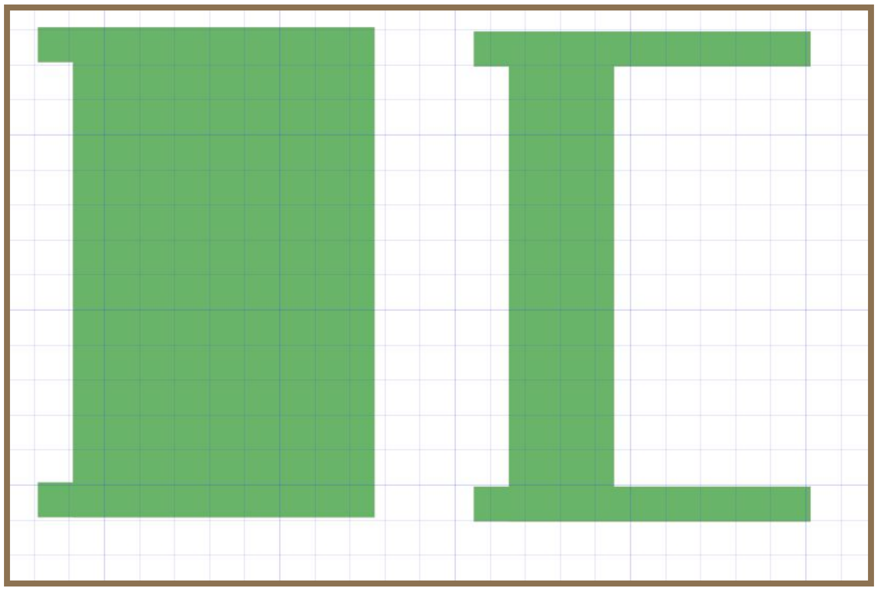
Middle panel to hold gears/gear racks



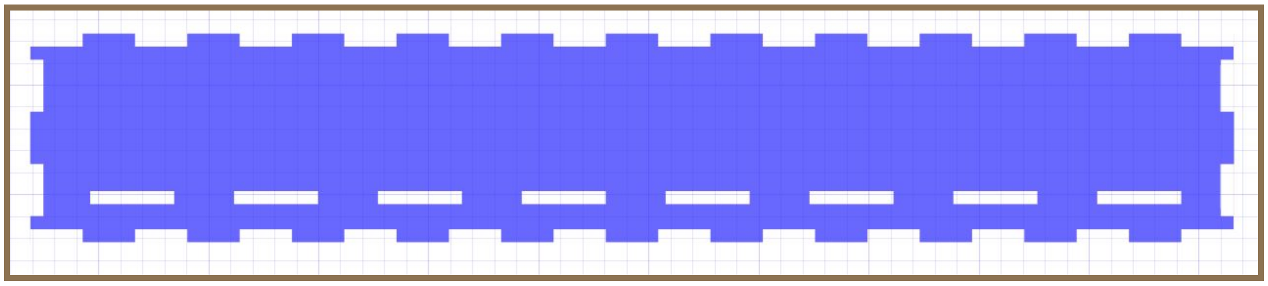
Window Holders



(Left) Stand for sides of the middle rack (Right) stands for holding middle portion

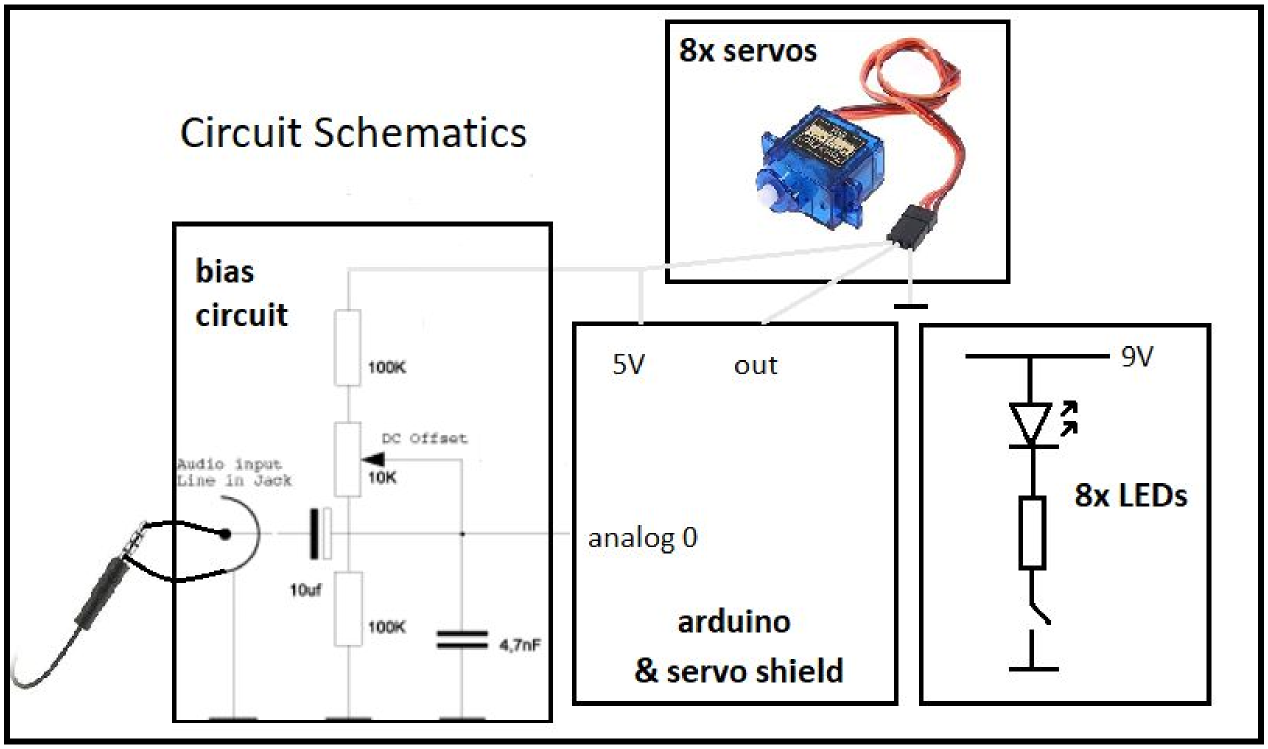


Top Panel With Holes for Gear Rack



Acrylic Panel

**Circuit Diagram**



## Resources

Open Music Labs: FHT Arduino Library:
http://wiki.openmusiclabs.com/wiki/ArduinoFHT

Adafruit PWM Servo Library:
https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library

Notes on Arduino Audio Processing:
http://interface.khm.de/index.php/lab/interfaces-advanced/arduino-realtime-audio-processing