# Diode Overture: Music Reactive Headpiece

David Bis

Adviser: Dr. Ellen McKinney

Honors Project – Spring 2019

Iowa State University

This project was funded in part by Dr. Cecil Stewart with the Stewart Research Grant.

**Abstract**

An important facet of live musical performance is its visual appeal. Musicians such as Daft Punk and Buckethead perform in iconic outfits that characterize the music they play. The objective of this project is to design and construct a headpiece capable of interpreting live musical data and generating a corresponding light routine based on the input from an instrument. Wearable technology is becoming increasingly important as means of daily utility and expression, and this project serves to explore and document the creation of such products.

Through the creative design process, an initial plan was developed to focus the features of the final product. From here, the headpiece and the electronics were developed independently until the moment of integration. The headpiece was constructed with a foundation of EVA foam and utilized heat forming and various coating techniques. The electronics were operated by an Arduino Nano microcontroller that managed the signal processing from both guitar and microphone input.

The findings and lessons learned throughout the project were collected into this document detailing the design and construction process. The document serves to preserve the information found and to enable others in exploring wearable technology implementations on their own.

# Table of Contents

# Diode Overture: Music Reactive Headpiece

The objective of this project is to design a headpiece capable of translating a signal from a musical instrument to a corresponding light routine. Stage presence is an important facet of live music performance. The combination of audio and visuals acts as a synergistic combo that can help augment the musician's intent and expression. Musicians such as Daft Punk and Buckethead perform as an alter ego character that have now become synonymous to their music. As a stage musician myself, I wanted to make a garment I could wear on stage that would serve a similar purpose. Since I play the guitar, I figured a headpiece would be the best piece of apparel to make since it would be out of the way of the instrument. I also wanted to develop a way for the music I was playing to translate to the headpiece. Since these actuators would need to be seen from afar, I figured using LED's would be the best way to show off the music visually. As for input, I wanted to be able to use my guitar to control the lights. I also wanted to have a microphone so that I could detect singing. As for the design of the helmet, I wanted to stick to the more mechanical side. Another inspiration from Daft Punk, I also wanted to make the aesthetic robotic yet still iconic. In the end, I wanted to make an alter ego for myself.

## Headpiece Construction

When I was planning on the design of the headpiece, I had a few requirements I had defined early on that I wanted to follow:

1. It should be light to wear.
2. It should not impair my vision.
3. It should house the electronics rather than have a separate component (like a waist box)

These functional requirements drove my design. This section will discuss the process and the decisions involved in the fabrication of the headpiece.

**Design.**

The first step in designing the headpiece was sketching out ideas. I wanted to have the headpiece appear as something out of science fiction media, such as Star Wars, the Destiny video game, and the Marvel movies. I collected my thoughts and drew them out on some paper. Figure 1 shows some of the highlights from my sketches, which acted as a driving force and a visual guide for the headpiece.

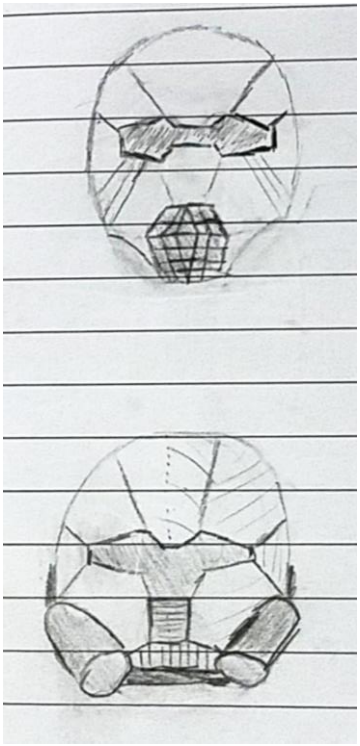

*Figure 1:* Initial headpiece idea sketches

**Material Selection.**

The first identified requirement was that the headpiece should be light to wear. Through my research, I discovered that many cosplay armors are made from ethylene-vinyl acetate (EVA) foam, which is a lightweight material that can be easily shaped and decorated to appear as other materials. Due to its popularity in cosplay, there are an abundance of resources online discussing

the best practices and other tips and tricks for working with it, making it an ideal candidate for the project.

An alternative considered was to use a 3D printer to fabricate the headpiece. This option would have likely given me more room to experiment with the appearance and better control for finer detailing. However, these would be due to the 3D modelling software, something I am rather inexperienced with. Due to the time constraints, I figured I would be better off sculpting with the EVA foam rather than learn advanced 3D modelling techniques.

The EVA foam identified was found in the form of flat sheets. In fact, the sheets I used were designed for exercise floor mats for home gyms. An unexpected benefit of the foam sheets was that one side had a diamond steel plate texture on it. This was great because it suited my mechanical design plan perfectly. The sheets I got were ½" thick, but that thickness can vary based on your purposes. I also got some 2mm thick sheets and 5mm thick sheets that I used for additional details to give the headpiece some more depth.

**Flat Pattern.**

Although it is possible to make a headpiece from a singular piece of foam, it would be challenging to properly shape. Instead, breaking the headpiece into several smaller components allows for an easier job for shaping the product. To do this, a flat pattern is made, a prominent technique used throughout all of apparel design. A flat pattern will be important for being able to map the flat EVA sheets to its proper 3D form.

There are few ways to go about designing a flat pattern. One popular technique is computer-aided design software. Another would be to start with a 3D foundation such as a mannequin and draw out the component lines on the form, remove the markings, and flatten out the result (Hiers, 2018). Due to the simplicity of the latter method, I decided to go that route.

The foundation of choice was a mannequin head. To preserve my component lines, I coated the headpiece with a layer of aluminum foil and then sealed the foil in place with a layer of duct tape so that the material held tightly to the mannequin to keep the shape of the head once it gets flattened. Using a marker, I drew a dividing line along the center of the face all the way to the back of the neck that would act as the line of symmetry for the components. From here, I only drew the component lines on the right side of the face because I could simply mirror each component to get an equivalent one for the left side to ensure symmetry later in the construction. I ended up with three components: one along the top of the head, one for the side of the head, and one in front of the face. This eventually worked out nicely as it fit the natural contours of the head, so each piece could remain relatively flat, which made it great to work with when trying to adhere them together. Figure 2 shows the flat pattern creation process.

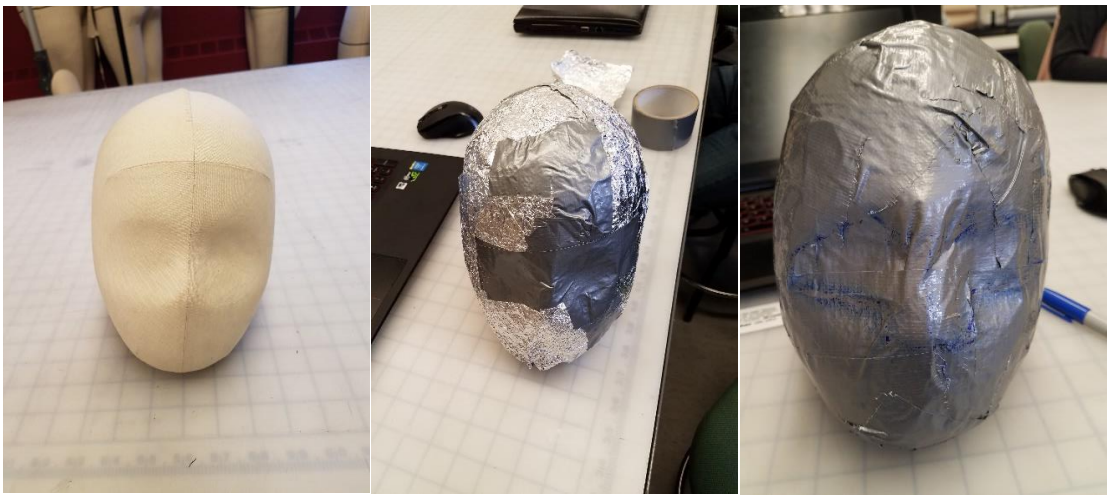

*Figure 2:* Flat pattern creation process

Once these lines were drawn, I cut them out and traced them onto some cardstock paper. This step is entirely optional, I just preferred being able to use this later when tracing the flat patterns onto the foam sheets because they provided a firmer edge to trace against as opposed to the foil and tape. Again, I only made a copy of the right-side components because they can just

be flipped over and work as an equivalent left side copy. Based on my design, I would require six unique components for the headpiece foundation. On each component, I drew small tick marks to help make sure everything was lining up to ensure symmetry. Figure 3 shows the flat patterns transferred to the cardstock paper.

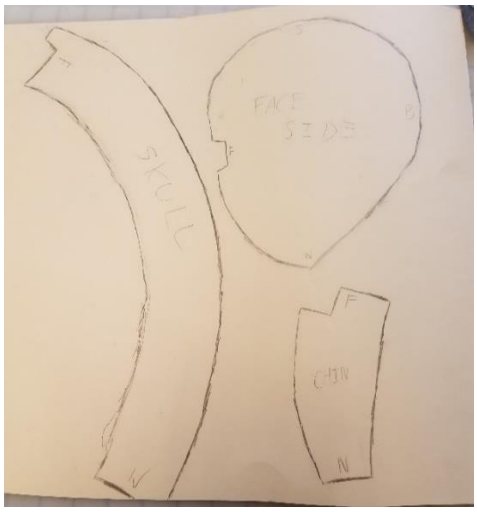


*Figure 3:* Cardstock paper traced flat patterns before transferring the tick marks

Before moving forward, I did run into a complication once these flat patterns were used to assemble the headpiece: it was too small. There are two possibilities as to why this may have happened: either the mannequin head was smaller than my own head, or once the foam was formed, the flat pattern mapped to the outermost layer of the foam rather than the innermost. Since the foam was ½" thick, it likely reduced the headspace significantly moving inward ½" from the original intended design. I remedied this problem by scaling up the flat patterns up by a ¼" in all dimensions, which made the headpiece fit much better. The original headpiece ended up as a testbed for the various coatings for the foam. For the intents of this document, I will only discuss the process on my larger flat pattern revisions unless stated otherwise.

In addition to the foundation, I made flat patterns for all the detail work later using a similar process as above. However, I used the headpiece foundation as the base, and opted out of

using the foil-and-duct-tape method for just fitting some paper directly on it instead. This made it easier for me to move and adjust them while giving me a good visual idea of how the final product would appear.

**Foundation Construction.**

The flat patterns were traced onto the flat side of the foam sheets. The foam sheets I bought came in a stack of four, and I was able to get away with using one sheet for each side of the head. I cut them out using both a heavy-duty pair of scissors and a utility knife. It is crucial that the edges of each component are perpendicular to the faces because it allows for maximum contact to be made when working on adhering them together. Once cut, I traced the tick marks onto the foam sheets shown in Figure 4.

*Figure 4:* Flat patterns transferred to foam sheets

Once they were cut out, I used a hair dryer to heat form the foam into its ideal shape. Other alternatives include using a heat gun or even a stove top to do this work. The heat forming is necessary to shape the pieces closer to their installed state, and so that the adhesive will not have to resist the force of the foam wanting to flatten out again. It is also good to give the form

more depth for ergonomics (Smith, 2014). For example, the two pieces on the sides of the head were formed to bow outward to provide room for the wearer's ears.

To adhere them together, contact cement was used. Some containers of contact cement come with a brush connected on the lid, but mine didn't, so I used a small foam brush instead. For safety measures, the best practice is to use some nitrile gloves when working with these kinds of adhesives, since they can be detrimental to your health if contact is made. Contact cement works by applying the glue to each face you want connected, waiting approximately ten minutes (though this can vary based on the temperature and humidity of the working environment), and then slowly connect the two faces together. The cement should bond instantly, so it's important to be careful about how contact is applied. However, if you begin the connection just before they are ready, you can still make small adjustments to the bond. I opted into doing this for each connection due to its error forgiveness. Many of these bonds weren't as seamless as I wanted, so I used some superglue for some of the minor touch-ups. This worked incredibly well for this purpose. Figure 5 shows the completed foundation. This particular design has plenty of space in the back of the head for the electronics, thus satisfying that third requirement.

*Figure 5:* Assembled headpiece foundation

For the detail components, I used some residual ½” foam for the antennas, and then some thinner foam sheets for the rest. I connected the antennas using contact cement and used superglue on the thinner sheets. Figure 6 shows the headpiece with the detail components added on.

*Figure 6:* Additional details added to headpiece

**Coatings.**

Now that the headpiece was assembled, the next step was to begin coating the foam to look more like metal. A solution here would be to use metallic spray paint, but there’s one problem: the foam will just absorb the paint. The solution to this problem is Plasti Dip, which is an aerosol-based rubber coating. A couple of layers of this that will seal the foam so that you can paint it (Rolon, 2016). Using this (and the spray paint), it is recommended to apply it in a well-ventilated room and to wear some kind of respirator, since inhaling the fumes can be detrimental to your health. An alternative to this would to be use Mod Podge, which will have a similar

sealing effect. For my project, I used Plasti Dip. There are three colors you can buy Plasti Dip in: white, black, and red. All three will act the same, but since the foam I was working with was black, I used the white variety so that I could easily see my coverage during application.

A good rule of thumb to use when applying this Plasti Dip is to spray it about 8” (20cm) away from the surface (Quindt, 2018). Anything outside this range and you start getting an undesired texture formed from the Plasti Dip. If it’s too close it will start to form air bubbles and being too far will not dry smoothly. You’ll need to apply multiple layers to ensure a complete seal. The results of the Plasti Dip application are shown in Figure 7. I was able to get by using four layers, but this can vary anywhere from three to six layers.

*Figure 7:* Headpiece coated in Plasti Dip

Once the Plasti Dip is dried, the paint could be applied. I used silver metallic spray paint for the foundation. Again, as with the Plasti Dip, it’s recommended to do this in a well-ventilated room and to wear a respirator when applying it. I ended up using three coats of paint, but this can vary based on your needs. To add some variety in the color, I used a brush to paint the detail

components with a gunmetal grey metallic paint. Although it is possible to paint the entire headpiece using this method, the spray paint was great because I could mostly ignore the diamond texture on the foam when applying it, something I could not do if I had painting it with a brush.

One unexpected benefit of using this kind of paint is that it produced a great brushed metal texture to the foam. Due to the side panels being so low on the headpiece, the Plasti Dip ended up forming a bubbly texture as gravity pulled some of the excess down. By applying multiple layers of this paint, I was able to smooth out the unwanted bubbles and instead transform it into the brushed metal texture.

**Additional Details.**

Before I integrated the electronics in the headpiece, there were a few additional details I wanted to add to it. One focus area was the antennas. To add some variety into them, I coated them with two layers of wires. The innermost layer is parallel to the antenna, whereas the outmost layer was a bus of wires coiled around it. From a distance, this gives the illusion that the antennas are just a large bundle of wires. Figure 8 shows the completed antenna details, along with lower face details.

*Figure 8:* Antenna details

The second focus area is the lower face. One of the goals with the details here was that I wanted to give it an option to breathe, so that when I was wearing the headpiece, I wouldn't be trapping myself with the stale air. My solution was to use the columns used in wedding cakes and to line them up after cutting them in half. The ends of these columns were attached with superglue to the foam, but the centers were cut out to act like a vent gate, allowing for my breath to leave the headpiece rather than get circulated internally.

Another important feature I added in was the head mount for the wearer. The headpiece is admittedly larger than most comparable helmets. This is good from a design standpoint because it gives plenty of room for the electronics to fit. However, when the wearer moves their head around, the headpiece often won't follow along with the direction of movement. To resolve this, an adjustable headband from a bicycle helmet was installed to lock the wearer's head movement to the headpiece. If fitted properly, it helps ensure that the wearer's vision is not impaired by the headpiece shifting around in relation to their eyes.

## Electronics

The other major component of the project is the electronics. To aid in the experimentation, development, testing, and debugging, the electronics were developed independently from the headpiece until it was necessary to integrate them together. There are three active components in the electronics: the microphone, the ¼" jack input, and the lights. Each of these components are driven by an Arduino Nano microcontroller. This section will discuss the design, implementation, and decisions regarding the electronics.

### Microcontroller.

The microcontroller is effectively the brain of the system. All of the input will be processed with the microcontroller and all respective commands will be output through it. Due to

their documentation and support, Arduino microcontrollers (or often referred to as Arduinos) are a great option for this kind of work. There are several different models of Arduinos, each with their own pin layouts and specifications. Due to their compact size and peripheral capabilities, the Arduino Nano makes for a great candidate for the project. This Arduino has 8 analog input pins and 22 digital output pins. I'll only need to use 2 analog input pins and 1 digital output pins, so satisfies that requirement. Figure 9 shows an Arduino Nano for reference. Likewise, they are easy to power. They have a Mini-B USB port on them, so they can be powered with a power bank. For this matter, I used a power bank with 3000mAh of capacity. Some power banks have a feature where if they are drawing less than 50mA, they turn off. Thankfully, the LED's will pull much more than that, but it is an issue to consider when selecting a power bank for other projects. For debugging purposes, I used an Arduino Mega and a breadboard to prototype the circuits.

*Figure 9:* Arduino Nano. From *Arduino Nano*, by Arduino .cc, https://www.flickr.com/photos/arduino_cc/8484361225

**Microphone Input.**

The microphone identified to use is the SparkFun Electret microphone. This was chosen because it has an amplifier built into it, so its output is ready to be processed out of the box. There are three pins on the component: ground, power, and signal. The ground and power are connected accordingly to the Arduino, whereas the signal is connected to one of the analog input pins. From here, we can do some processing of the signal through some code using the Arduino IDE. I found a great library that supports computing a Fast-Hartley Transform (FHT) on the signal (Open Music Labs, 2016). Like the Fast-Fourier Transform (FFT), the FHT computes the frequency spectrum of a signal. Just like how some car stereos have the display for different columns for the amplitude of different frequencies in a signal, the FHT stores the same kind of data in an array, where each element represents the magnitude of frequency within a specific range. This FHT library provided a high-resolution decomposition of the frequencies, being able to output up to 128 frequency segments rather than 8 that the FFT libraries could output[1].

[1] This has been found to be untrue, there are FFT libraries that are comparable to this FHT library in respect to output size. However, the FHT operation is slightly faster than the FFT operation (Piccinin, 1988), so it is still preferred to use that library.

**Guitar Input.**

The guitar input is taken in using a ¼" jack where a guitar (or bass) can be plugged in directly. The jack has two pins on it: ground and signal. However, the signal will give a response between -2.5V and 2.5[2], while the Arduino operates between 0V and 5V. If the guitar input was plugged directly to an analog pin, half of signal is lost because it would drop below zero and be unreadable by the Arduino. To resolve this, a DC offset circuit is made to shift the voltage up by 2.5V so that it, too, operates between 0V and 5V (nikoala3, n.d.). The circuit was placed on a protoboard to keep it small and concise. Figure 10 displays the breadboard used for testing a circuit with a DC offset and amplifier circuit.

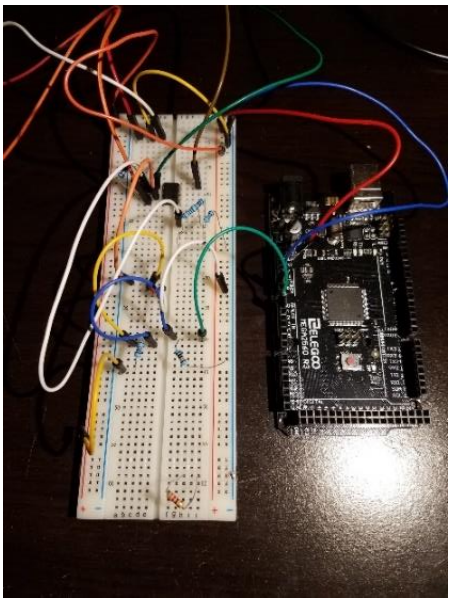

*Figure 10:* Experimental Amplifier and DC Offset Circuit

An amplifier circuit was considered for the circuit so that it could be processed the same way as the microphone using the FHT. I decided not to use this because I found a few alternatives for the signal processing that were simpler and wouldn't require the amplifier, making the system simpler and more manageable. For example, I noticed a trend in how the guitar signal responds based on the notes that are played. Lower notes tend to result in a higher

amplitude response, whereas higher notes tend to result in a lower amplitude response[3]. However, this is not fool proof because I could simply play the higher notes louder and I could fool the algorithm into thinking it's a lower note, and vice versa if I played quieter on a lower note. It's workable if I were to play at the same dynamic level across all strings, but that is divorced from being able to play with expression, which is an obvious setback of this method. Regardless, this is a pretty naïve solution to note detection.

I finally settled on a method known as autocorrelation, which works by approximating the length of a sinusoidal input's period as a function of a signal and a delayed version of that signal (akellyirl, n.d.). The benefits of this one is that it can provide a frequency approximation of the notes, but at the cost of some lag, which is a result of the algorithm needing to take large samples of the signal between processes. An implementation of this method is shown in the appendix using code from Arduino forum user MrMark (MrMark, 2018).

However, the FHT library and the autocorrelation algorithm implementations have produced some compatibility issues. Independently, the two operate as expected, but together the code appears to lock up. A solution for this problem is planned for future work.

[2] This is only true if the signal has been amplified with the proper gain. A direct input from the guitar is a signal of only a few millivolts centered around zero volts.

[3] It's not the notes that result with the amplitude response, but instead the strings. The lower-pitched strings on my guitar resulted in the higher amplitudes, yet a specific note can be played on multiple strings. However, the guitar is normally played vertically across the strings rather than horizontally across one string, so the trend is still practical for pitch approximation, but it's still a naïve solution.

**LED Strips.**

The LED strips used were strips of WS2812b LED's. These each have four different pins: 5V power, ground, data in, and data out. Due to their operation, it is required that the LED strip is connected to the Arduino on a single digital output pin. Although it may appear that the LED's are being lined up in series, only the data pins are lined up that way. The power and ground pins are still assembled in series.

Between each LED there are some small leads for each pin that can be cut and wired together, so I cut several smaller strips to and wired them up appropriately. Thankfully, many LED strips have an adhesive on the bottom, so I just used that to apply each LED strip to the headpiece.

To support the code for the LED Strip, the Adafruit NeoPixel library was used. The library offers an assortment of methods to simplify the programming process. Through the library, the LED strip is treated as an array, where each array element is an LED. This abstraction makes it convenient for doing simple animations with the lights. An implementation of this library is shown in the appendix.

One unfortunate side effect of the integration of these components is signal noise. When the lights are off, the microphone and the guitar signal read just fine. However, as soon as the lights turn on, the signals get mutated enough to the point where the frequency detection algorithms do not work anymore. Instead, only the attacks and releases of notes are detectable. Solving this problem is planned as future work for the project and will be discussed later.

**Implementation into Headpiece.**

Once the electronics were made and tested, they were implemented into the headpiece. Since there was enough room in the back of the headpiece, the Arduino, protoboard, and ¼" jack

was fitted on the back inside the headpiece. A long bus of wires was stretched to the front for the microphone so that it can more directly take input from my voice. A power drill was used for the holes for the LED strip wires to move in and out of the headpiece discreetly. A completed circuit diagram of the electronics is shown in Figure 11, produced with Fritzing software and using additional schematics from Fritzing forum users Peter Van Epp for the guitar audio jack and rlah for the WS2812 LED strip.

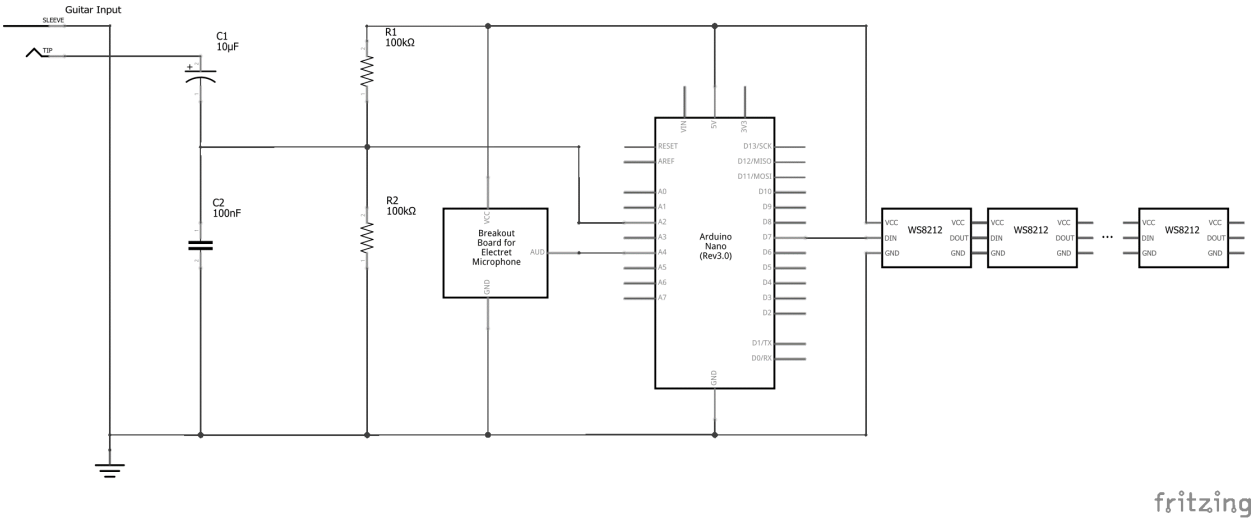


*Figure 11:* Electronics Circuit Diagram

## Results and Conclusion

The final headpiece is shown in Figure 12. The electronics are capable of successfully detecting note can send appropriate commands to the LED strips for performance. However, the signal noise from the lights prevents appropriate signal processing and causes erroneous behaviors when the lights are turned on. There is work currently planned towards finding a solution to solve this problem. A bill of materials is shown in the appendix along with code incorporating all the algorithms discussed.

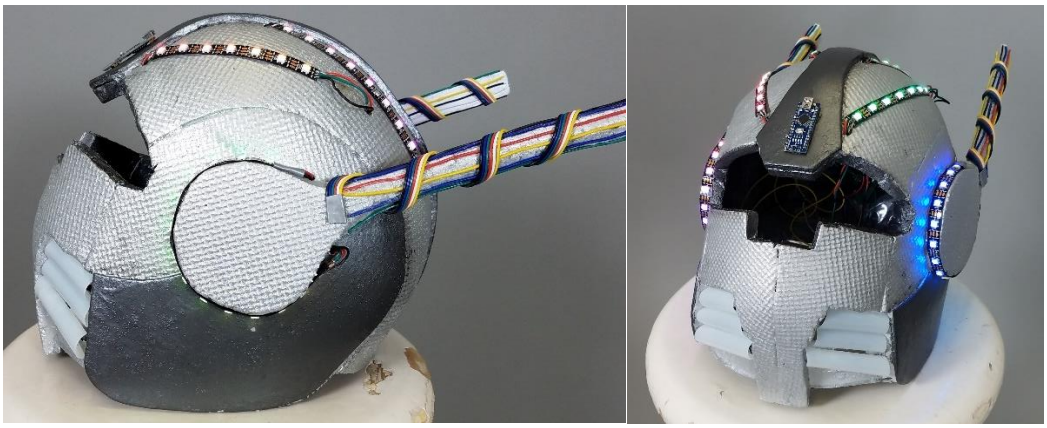

*Figure 12:* Completed headpiece

**Lessons.**

The most important lesson I learned from this project was to plan for failures to happen in the schedule. Having my first headpiece come out as too small was an unexpected result that set me back a week in time from my originally planned schedule. Especially for creative design work, I find that it is important to allocate significant time to experiment and demo ideas to understand what ideas and methods will work and what won't.

In terms of technical skills, I learned a few signal processing methods for frequency detection. There are a handful of different methods, each with their own advantages and disadvantages. Likewise, I also improved my understanding of circuits. As a software engineering major, I don't get to work with circuits often, so this provided me with a challenge outside my normal curriculum.

The challenges involving the design process have also given me a great opportunity to learn. In my engineering classes, the definition of project requirements is crucial for the success of a project. Following a regimented design process for this project has allowed me to see the parallels between traditional engineering and creative design. The act of defining sufficient

requirements to focus my efforts has further concreted my understanding of effective project management, no matter the nature of the project.

**Future Plans.**

The highest priority change I want to implement is to improve the simultaneous signal processing and the light animations. Due to the lights producing significant signal noise, the signal processing algorithms just don't work as intended. There are two potential solutions I'd like to try to see what effects they have. The first is to send the signal data through a low-pass filter to eliminate the high frequency noise while still preserving the target signal. Another solution would be to use two microcontrollers in the system: one for signal processing and one for light control. The two can communicate using the Receive (Rx) and Transmit (Tx) pins. Isolating these two systems can help mitigate the amplification of noise through the system. Likewise, finding a solution for enabling sequential input from the microphone and guitar inputs is also a priority.

**Conclusion.**

I certainly learned a lot by working on this project. All the way from the drawing board to the circuit board, this full-stack project has provided me a challenge in both my technical skills as a software developer and my creative design skills. Most importantly, the project gave me an opportunity to fuse those challenges together, providing me a new perspective on issues and the various solutions that follow.

## References

Adafruit. (n.d.). *Adafruit_NeoPixel*. Retrieved from GitHub: https://github.com/adafruit/Adafruit_NeoPixel

akellyirl. (n.d.). *RELIABLE FREQUENCY DETECTION USING DSP TECHNIQUES*. Retrieved from Tech Stories: http://www.akellyirl.com/reliable-frequency-detection-using-dsp-techniques/

Burgess, P. (2019, December 7). *Adafruit NeoPixel Überguide - Basic Connections*. Retrieved from Adafruit: https://learn.adafruit.com/adafruit-neopixel-uberguide/basic-connections

Hiers, Q. (2018, January 2). *Making a Pattern for a Foam Helmet*. Retrieved from But Why Tho?: https://butwhythopodcast.com/2018/01/02/making-a-pattern-for-a-foam-helmet/

MrMark. (2018, April 12). *Arduino pitch detection in realtime (using autocorrelation and peak detection)*. Retrieved from Arduino Forum: https://forum.arduino.cc/index.php?topic=540969.msg3687113#msg3687113

nikoala3. (n.d.). *Arduino Guitar Tuner*. Retrieved from Instructables: https://www.instructables.com/id/Arduino-Guitar-Tuner/

Open Music Labs. (2016, November 27). *Arduino FHT Library*. Retrieved from Open Music Labs: http://wiki.openmusiclabs.com/wiki/ArduinoFHT

Piccinin. (1988, February). *The Fast Hartley Transform as an alternative to the Fast Fourier Transform* [Memorandum]. Salisbury, South Australia: Department of Defence, Defence Science and Technology Organisation, Surveillance Research Laboratory. Retrieved from https://apps.dtic.mil/dtic/tr/fulltext/u2/a212493.pdf

Quindt, S. (2018, June 14). Retrieved from How to apply PlastiDip - Cosplay Tutorial: https://www.youtube.com/watch?v=WVcfaIGedtk

rlah. (2019, January 15). *WS2812 RGB LED strip & matrix*. Retrieved from Fritzing Forum: https://forum.fritzing.org/t/ws2812-rgb-led-strip-matrix/6339

Rolon, E. (2016, September 5). Retrieved from EVA Foam Armor Plasti Dip Tutorial: https://www.youtube.com/watch?v=irZkMX2hfEI

Smith, T. (2014, March 13). *How To Make a Foam Helmet,Tutorial Part 1*. Retrieved from https://www.youtube.com/watch?v=ODSNPYdvJRo

Van Epp, P. (2017, July 25). *Conductor ¼ jack mono graphics?* Retrieved from Fritzing Forum: https://forum.fritzing.org/t/conductor-1-4-jack-mono-graphics/4231

# Appendix

## Bill of Materials.

The following table shows the material requirements for the headpiece. Each item is broken into two categories, headpiece or electronics, which dictates which primary component they are implemented with.

| Item ID | Description | Qty. | Category |
|---|---|---|---|
| 1 | ½” EVA foam sheets (4 pk) | 1 | Headpiece |
| 2 | 5mm EVA foam sheet | 1 | Headpiece |
| 3 | 2mm EVA foam sheet | 2 | Headpiece |
| 4 | Wedding Cake Columns | 2 | Headpiece |
| 5 | Contact Cement (1 pint) | 1 | Headpiece |
| 6 | Plasti Dip | 1 | Headpiece |
| 7 | Metallic Silver Spray Paint | 1 | Headpiece |
| 8 | Gunmetal Grey Paint | 1 | Headpiece |
| 9 | SparkFun Electret Microphone | 1 | Electronics |
| 10 | ¼” Audio Input Jack | 1 | Electronics |
| 11 | Arduino Nano | 2 | Electronics |
| 12 | 3000 mAh Power Bank | 1 | Electronics |
| 13 | 22 AWG Threaded Wire (per spool) | 6 | Electronics |
| 14 | WS8212b LED Strip (per meter) | 2 | Electronics |
| 15 | 100kΩ Resistor | 2 | Electronics |
| 16 | 100F Ceramic Capacitor | 1 | Electronics |
| 17 | 10 µF Electrolytic Capacitor | 1 | Electronics |
| 18 | Protoboard | 1 | Electronics |

*Table 1:* Project Bill of Materials.

**Code Implementation.**

The following code is the implementation running on the Arduino Nano. It uses code derived from the Adafruit NeoPixel libraries for the LED strip (Adafruit, n.d.) and an autocorrelation algorithm by Arduino Forum user MrMark (MrMark, 2018).[4]

```
#include <Adafruit_NeoPixel.h>

#define PIN 7
#define INPUT_SIZE 256

int numLights = 60;

byte guitarInputAC[INPUT_SIZE];

int count;

// Sample Frequency in kHz
const float sample_freq = 8919;

int len = sizeof(guitarInputAC);
int i, k;
long sum, sum_old;
int thresh = 0;
double freq_per = 0;
byte pd_state = 0;

int lightCounter = 0;
const int LIGHT_COUNTER_MAX = 15;

Adafruit_NeoPixel strip = Adafruit_NeoPixel(numLights, PIN, NEO_GRB + NEO_KHZ800);
uint32_t off = strip.Color(0,0,0);

int leftEar_start = 0;
int leftEar_end = 13;

int head_start = 14;
int head_end = 45;

int rightEar_start = 46;
int rightEar_end = 60;

/**
 * Set-Up Lights
 */
void setup() {
  analogRead(A2);
  Serial.begin(115200);
  count = 0;

  strip.begin();
  strip.setBrightness(20); //adjust brightness here
  strip.show(); // Initialize all pixels to 'off'


```

```
50.     for(int i = 0; i < INPUT_SIZE; i++) {
51.       guitarInputAC[i] = 0;
52.     }
53.   }
54.
55.   void loop() {
56.     // Collect analog signal for autocorrelation
57.     for (count = 0; count < INPUT_SIZE; count++) {
58.       guitarInputAC[count] = analogRead(A2) >> 2;
59.     }
60.
61.     // Begin autocorrelation algorithm from MrMark
62.     // https://forum.arduino.cc/index.php?topic=540969.msg3687113#msg3687113
63.     // Calculate mean to remove DC offset
64.     long meanSum = 0;
65.     for (count = 0; count < INPUT_SIZE; count++) {
66.       meanSum += guitarInputAC[count];
67.     }
68.     char mean = meanSum / INPUT_SIZE;
69.
70.     // Autocorrelation stuff
71.     sum = 0;
72.     pd_state = 0;
73.     int period = 0;
74.     for (i = 0; i < len; i++)
75.     {
76.       // Autocorrelation
77.       sum_old = sum;
78.       sum = 0;
79.       for (k = 0; k < len - i; k++) sum += (guitarInputAC[k] - mean) * (guitarInputAC[k +
      i] - mean) ;
80.
81.       // Peak Detect State Machine
82.       if (pd_state == 2 && (sum - sum_old) <= 0)
83.       {
84.         period = i;
85.         pd_state = 3;
86.       }
87.       if (pd_state == 1 && (sum > thresh) && (sum - sum_old) > 0) pd_state = 2;
88.       if (!i) {
89.         thresh = sum * 0.5;
90.         pd_state = 1;
91.       }
92.     }
93.
94.       // Frequency identified in Hz
95.       if (thresh > 100) {
96.         freq_per = sample_freq / period;
97.       }
98.
99.      // Only update if a frequency was detected
100.     // There's a chance there is no frequency detected, which results in freq_per == inf

101.      if(freq_per < 1100) {
102.         int newColor = map(freq_per, 80,1100, 0, 255);
103.         rainbow(newColor);
104.      } else {
105.         colorFill(off);
106.      }
107. }
108.
```

```
109.
110.
111.
112. void colorFill(uint32_t c){
113.   for(uint16_t i=0; i<strip.numPixels(); i++) {
114.     strip.setPixelColor(i, c);
115.   }
116.    strip.show();
117. }
118.
119. void rainbow(uint16_t j) {
120.   uint16_t i;
121.   for(i=0; i<strip.numPixels(); i++) {
122.     strip.setPixelColor(i, Wheel((j) & 255));
123.   }
124.   strip.show();
125. }
126.
127. /***********************************/
128. /**        UTILITY METHODS        **/
129. /***********************************/
130.
131. // Input a value 0 to 255 to get a color value.
132. // The colours are a transition r - g - b - back to r.
133. uint32_t Wheel(byte WheelPos) {
134.   if(WheelPos < 85) {
135.    return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
136.   } else if(WheelPos < 170) {
137.    WheelPos -= 85;
138.    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
139.   } else {
140.    WheelPos -= 170;
141.    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
142.   }
143. }
```

[4] This code has been modified from the original in order to resolve an error and to keep it concise.

## Addendum

### Post-Graduation Update

I have been working more on the project and have reached an important milestone in the project worthy of an update. Although I have since graduated, I find it important to share the relevant information. This addendum will serve to share progress on this project after my graduation.

An issue I was working on solving was the concurrent input of the microphone and guitar input. It is important to note that the Arduino Nano is single-threaded, so it's not possible to take input from the two sources at the exact same time. Instead, it is best to sample the two sequentially, and then process them afterwards. When the other signal is being sampled, and while processing is occurring, the signal is being lost, however, the microcontroller is fast enough that the loss is functionally insignificant.

As for the code, the OpenMusicLabs FHT library provided some example code to demonstrate a proper implementation of its features. The Arduino native libraries provide a handful of helpful functions to easily configure the microcontroller or use the hardware features, such as *analogRead(uint8_t pin)*, which reads the analog response from the ADC on a given pin. When the *analogRead(uint8_t pin)* function is called, the code triggers a conversion from the analog signal to a digital byte format. The result is stored in two registers, ADCL and ADCH, where are then concatenated to create an integer. The catch is, the example code provided doesn't use this and instead uses their own implementation of the function. When their implementation is used in conjunction with an analogRead() implementation, the code will actually get stuck because to two methods handle the ADCSRA and ADSC differently, causing the code to reach an infinite loop waiting for the register to reset since the escape condition will never be true. To get around this, I refactored the FHT library example to get data using the

analogRead() method, allowing for both guitar and microphone inputs to be collected sequentially.

Another one of the biggest problems I had been facing with this project was dealing with the signal noise from the LED's. Whenever the LED's were on, the microphone and guitar signals became unreadable. I knew the problem was stemming from how the system was wired, so I wanted to reconsider how my circuit schematic was designed. I developed a plan to isolate the LED's from the inputs as a much as possible, which would hopefully minimize the noise added to the system. In doing this, I added in a separate power source in form of a second power bank specifically for the LED strip. The 5V line for the LED strip was redirected to the new power bank, and the power bank ground was connected to the common ground for the rest of the system. In addition, I added a 330 Ohm resistor in series with the LED strip's data line. Some LED strip configurations warrant for having a capacitor placed in between the positive and negative terminals of the strip's power source, but I opted out since I don't have a particularly substantial amount of LED's, and my trial with using a 1000μF electrolytic capacitor didn't perform as well when installed. Regardless, these are both intended to help manage the current in the system and protect the LED's from being damaged (Burgess, 2019). A revised circuit diagram is shown below in Figure 13, produced with Fritzing software and using additional schematics from Fritzing forum users Peter Van Epp for the guitar audio jack and rlah for the WS2812 LED strip.

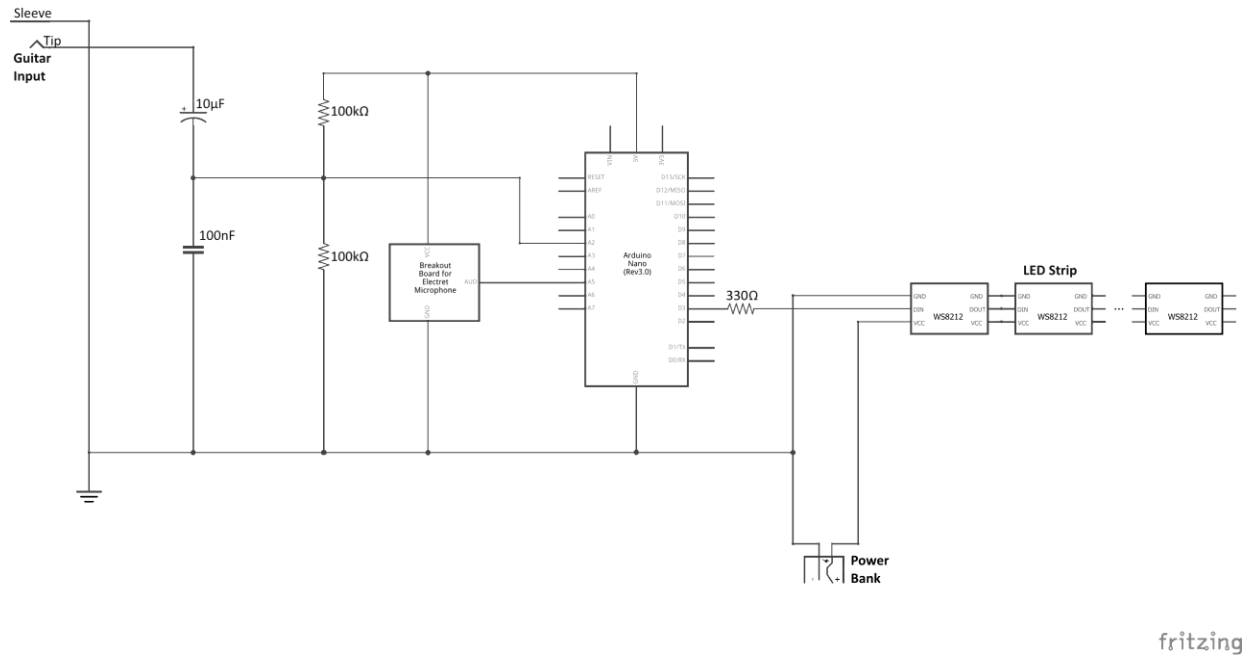


*Figure 13*: The Improved Circuit Diagram

This design solved the problem with signal noise pollution! Now, the system can manage the audio inputs with the outputs. It is also important to note that because the current drain was divided between two power banks, the Arduino system was at risk of not pulling enough current to keep the power bank on. Thankfully, I had a power bank available that did not have the 50mA minimum draw limit that I could use to power the Arduino.

The improvements made in the iteration have helped solve the LED noise pollution problem, marking this as a significant milestone in the project. The headpiece is functioning in a manner responsive to the two musical inputs with minimal interference from external factors. Although there are improvements that could be had with a more powerful microcontroller, the current microcontroller gives consistent and clean readings and performs surprisingly well. I wanted to measure how fast it was able to update, so I used the Arduino function *millis()* to keep track of the time the *loop()* function was called. The difference of the times reported from *millis()* each loop was recorded as the execution time for one update, which resulted in an

average of 115ms. A song at 120 beats per minute has a sixteenth note taking 125ms, so this update time is plenty fast for a wide range of performances.

While I was improving the circuit, I wanted to be able to detach the circuit from the headpiece so that it would be easier to work with. There are only three wires necessary for the LED strip, so everything else could be kept in one piece. I decided to cut those wires and implement a system of pin sockets at the ends of each wire. The LED strip ends were all male sockets, and the detachable circuit ends were all female sockets, so connecting the two parts was just as easy as plugging the right sockets together. I made sure to color code my wires, so connecting them isn't a challenge. In order to keep everything neat and tidy, I built my improved circuit on a new protoboard so that I wouldn't lose my original circuit. The new protoboard had terminal rails like those seen on a breadboard, so it made assembling the new circuit much easier because I could rely on the terminals to manage connections rather than needing to solder my own connections between the components.

Another improvement I wanted to implement was the circuit's casing. Now that the primary circuit was detachable, I was able to house the electronics in a plastic casing that could easily be attached and removed from the back interior of the headpiece. The primary requirement I had for this casing was that it would have enough openings for the two power bank ports, the guitar input, the microphone wires, and the three LED strip wires. I settled on using a case for a Raspberry Pi 4 because of its compact size and the fact that it had a curved surface very similar to that of the headpiece interior. The curved surface allowed me to use Velcro to mount it inside since I could guarantee full contact between the two Velcro parts throughout the curve. I did have to cut some additional holes in the case so that my ports would be easily accessible once the circuit was inside, but the plastic case was thin enough to break away with some scissors and a

needle nose plier. Afterwards, I sanded down the cuts to make sure the surfaces were smooth and wouldn't present any risk to the wires. The circuit fits snugly inside the case, so even while moving around, the circuit shouldn't be in any harm of damage. The separation of the circuit allows for the algorithm logic to be implemented into other devices. If I were to design another garment with the same addressable LED's, I could reuse that same circuit. An image of the circuit box is shown in Figure 14, along with an image of the headpiece interior in Figure 15.

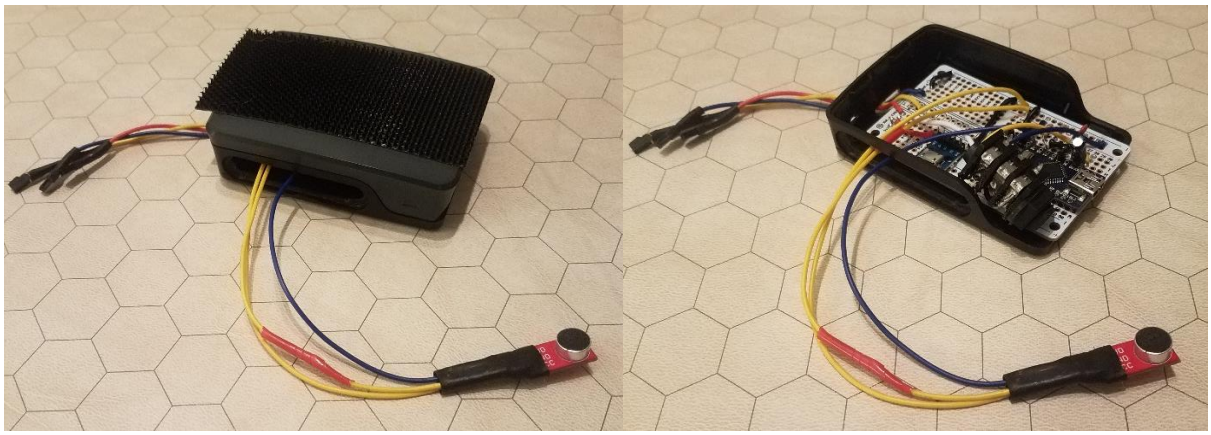

*Figure 14*: Circuit Box

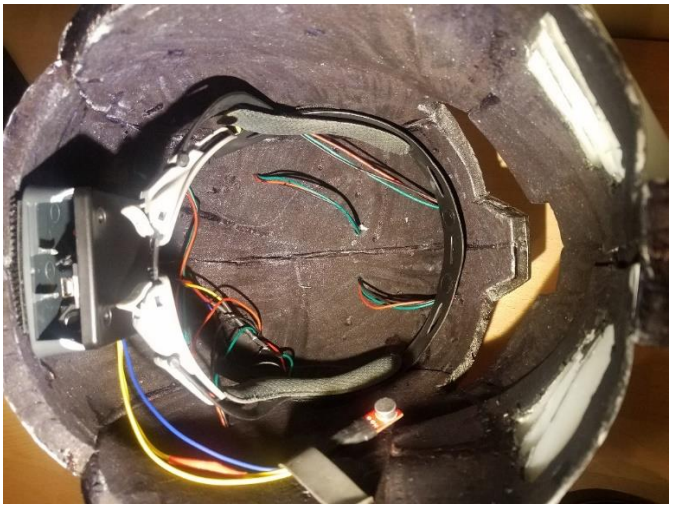

*Figure 15:* Headpiece Interior

There are a few comments I wanted to make regarding the other decisions that I had not mentioned before that in retrospect I think are important to point out. The first point is about the sample rate in the autocorrelation algorithm. The algorithm implementation by MrMark has a constant variable referred to as SAMPLE_FREQ set to 8919, which is in reference to the frequency of which the Arduino reads an input in kHz. This is an important constant for the autocorrelation algorithm because without it, there is no source of temporal truth when the algorithm is running.

Another point worth mentioning regards the FHT output. Each bin corresponds to a specific range of frequencies. However, each bin peaks at around once every 140Hz. A standard guitar can range from E2 (82 Hz) all the way up to E6 (1319 Hz), which would fit in approximately 9 bins. Keep in mind, there are 60 notes between E2 and E6, making it a poor candidate for detecting guitar notes. Although it would open the opportunity for discerning a polyphonic signal, the precision of process would only be able to detect if multiple notes were played over a large range, such as a chord. However, it wouldn't be precise enough to detect what chord was being played. Although the autocorrelation algorithm is designed for monophonic input, it is precise enough to discern between two semitones across the fretboard, making it much better suited for that type of input. Regardless, processing the microphone with the FHT can detect the frequency variety of the environment. Consider this being used in an ensemble performance, such as a rock band consisting of drums, bass, and a singer. If the microphone is listening in on the stage sound, the headpiece could react to the entire band. My original plan was to use the microphone for vocals, however, being able to react to the environmental noise can make it more expressive.

It is important to discuss how this helmet is intended to be used for live performance. A naïve solution would be to use a ¼" jack splitter right out of the guitar with one end going to the headpiece and the other to the amp. However, the signal to weakened resulting in a change of the guitar's tone. A solution to this is to use a circuit that can split the signal without significantly weakening it. Thankfully, this kind of circuit is quite popular with guitar pedals. For this purpose, guitar pedals are best viewed as circuits that are designed to alter the guitar signal in a specific way, such as adding a reverb or chorus effect to the signal. One pedal type is known as the ABY pedal, which can split a signal to two different outputs without significantly altering them from the input. These are generally designed so that one guitar can be output to two amps, but for my project, one of the outputs could be directed to the headpiece. Another solution is found on a variety of different pedals, such as my MXR Analog Chorus pedal. That solution is that the pedal has two outputs: one altered "wet" signal, and one original "dry" signal. I can run my guitar directly into my chorus pedal and use the dry signal to go to the headpiece and use the wet signal to go into the amp. Since I already had that pedal, I've identified to use that for performance purposes.

Ultimately, these improvements allowed for the project to reach its original requirements. The circuit improvements eliminated the noise issue and allowed for the input signals to be read and processed appropriately. Likewise, being able to separate the circuit allows for it to be easily worked with and can now be implemented into future projects. Although the Arduino Nano is a bit limited in performance, given that it is a single-threaded machine, it is still able to react to the playing at a satisfactory rate for most musical performances. All in all, these improvements have helped satisfy the requirement of visually representing the musical inputs as well as enabling the electronics to be implemented into new devices.

**Bill of Materials.**

The following table shows the material requirements for the headpiece. Each item is broken into two categories, headpiece or electronics, which dictates which primary component they are implemented with.

| Item ID | Description | Qty. | Category |
|---|---|---|---|
| 1 | ½” EVA foam sheets (4 pk) | 1 | Headpiece |
| 2 | 5mm EVA foam sheet | 1 | Headpiece |
| 3 | 2mm EVA foam sheet | 2 | Headpiece |
| 4 | Wedding Cake Columns | 2 | Headpiece |
| 5 | Contact Cement (1 pint) | 1 | Headpiece |
| 6 | Plasti Dip | 1 | Headpiece |
| 7 | Metallic Silver Spray Paint | 1 | Headpiece |
| 8 | Gunmetal Grey Paint | 1 | Headpiece |
| 9 | SparkFun Electret Microphone | 1 | Electronics |
| 10 | ¼” Audio Input Jack | 1 | Electronics |
| 11 | Arduino Nano | 2 | Electronics |
| 12 | 3000 mAh Power Bank | 2 | Electronics |
| 13 | 22 AWG Threaded Wire (per spool) | 6 | Electronics |
| 14 | WS8212b LED Strip (per meter) | 2 | Electronics |
| 15 | 100kΩ Resistor | 2 | Electronics |
| 16 | 330Ω Resistor | 1 | Electronics |
| 17 | 100F Ceramic Capacitor | 1 | Electronics |
| 18 | 10 µF Electrolytic Capacitor | 1 | Electronics |
| 19 | 1000 µF Electrolytic Capacitor | 1 | Electronics |
| 20 | Protoboard | 1 | Electronics |
| 21 | USB to PCB Breakout Board | 1 | Electronics |
| 22 | Wire sockets | 6 | Electronics |
| 23 | Raspberry Pi 4 Case | 1 | Electronics |

*Table 2:* Project Bill of Materials.

**Code Implementation.**

The following code is the implementation running on the Arduino Nano. It uses the Adafruit NeoPixel libraries for the LED strip (Adafruit, n.d.), adapted code from the Open Music Labs Fast Hartley Transform (Open Music Labs, 2016) for microphone input, and an autocorrelation algorithm by Arduino Forum user MrMark (MrMark, 2018). The color of the lights is controlled by the guitar pitch, with higher notes resulting in colder, bluer tones, and lower notes resulting in darker, redder tones. The brightness is determined by loudness of pitches around 262 Hz, or middle C.

```
#define FHT_N 64 // For microphone input
#define LOG_OUT 1
#define PIN 3
#define INPUT_SIZE 256 // For guitar input

#include <Adafruit_NeoPixel.h>
#include <FHT.h>

const float SAMPLE_FREQ = 8919; // Sample Frequency in kHz
const int MICROPHONE_SAMPLE_SIZE = 5;
const int GUITAR_THRESHOLD = 130;


// Autocorrelation variables
byte guitarInput[INPUT_SIZE];
int count;
int len = sizeof(guitarInput);
int i, k;
long sum, sum_old;
int thresh = 0;
double freq_per = 0;
byte pd_state = 0;

// Neopixel variables
int numLights = 60;
Adafruit_NeoPixel strip = Adafruit_NeoPixel(numLights, PIN, NEO_GRB + NEO_KHZ800);
uint32_t off = strip.Color(0,0,0);

// Helmet light segment helper variables
int leftEar_start = 0;
int leftEar_end = 13;
int head_start = 14;
int head_end = 45;
int rightEar_start = 46;
int rightEar_end = 60;

// Additional helper variables
int brightnessMonitor = 0;
int microphoneSum = 0;
```

```
40.
41.   /**
42.    * Set-Up the System
43.    */
44.   void setup() {
45.     Serial.begin(115200);
46.
47.     strip.begin();
48.     strip.setBrightness(20); //adjust brightness here
49.     strip.show(); // Initialize all pixels to 'off'
50.
51.
52.
53.   for(int i = 0; i < INPUT_SIZE; i++) {
54.       guitarInput[i] = 0;
55.     }
56.   }
57.
58.
59.   /**
60.    * Constantly loop the code
61.    */
62.   void loop() {
63.
64.     // Retrieve microphone data as fht_log_out array
65.     getMicrophoneReadings();
66.
67.     // Retrieve guitar data as freq_per
68.     getGuitarFrequency();
69.
70.     // Smooth out microphone output by averaging the output in groups of size
      MICROPHONE_SAMPLE_SIZE
71.     brightnessMonitor = (brightnessMonitor + 1) % MICROPHONE_SAMPLE_SIZE;
72.     microphoneSum += fht_log_out[2];
73.     if(brightnessMonitor == 0) {
74.       int avg = microphoneSum / MICROPHONE_SAMPLE_SIZE;
75.
76.       int brightness = map(avg, 0, 150, 0, 255); // The input min and input max values (0
      0 && 150) were retreived from empirical testing, they might differ in different context
      s
77.       brightness = constrain(brightness, 0, 255);
78.       microphoneSum = 0;
79.       strip.setBrightness(brightness);
80.     }
81.
82.     // Only update if a frequency was detected and playing is detected
83.     if(freq_per < 1100) {
84.       uint32_t newColor = map(freq_per, 80,1100, 0, 42000);
85.       strip.fill(strip.ColorHSV(newColor), 0, 0);
86.     } else {
87.       strip.fill(off, 0, 0);
88.     }
89.     strip.show();
90.   }
91.
92.   /*********************************/
93.   /**        UTILITY METHODS        **/
94.   /*********************************/
95.
96.   /**
97.    * Gets the microphone frequency spectrum and stores it into fht_log_out
```

```
98.    * Adapted from example code from Open Music Labs
99.    * http://wiki.openmusiclabs.com/wiki/FHTExample
100.   * Each element of the array represent a specific frequency band
101.   * Lower frequencies (elements 0 and 1) tend to always record high due, they aren't
       really useful
102.   * Example response peaks by element
103.   * 2 - 262 Hz
104.   * 3 - 392 Hz
105.   * 4 - 550 Hz
106.   * 5 - 700 Hz
107.   * 6 - 830 Hz
108.   * 7 - 950 Hz
109.   * 8 - 1100 Hz
110.   */
111.  void getMicrophoneReadings() {
112.    cli();  // UDRE interrupt slows this way down on arduino1.0
113.    for (int i = 0 ; i < FHT_N ; i++) { // save 256 samples
114.      int k = analogRead(A5);
115.      fht_input[i] = k; // put real data into bins
116.    }
117.    fht_window(); // window the data for better frequency response
118.    fht_reorder(); // reorder the data before doing the fht
119.    fht_run(); // process the data in the fht
120.    fht_mag_log(); // take the output of the fht
121.    sei();
122.    Serial.write(255); // send a start byte
123.    Serial.write(fht_log_out, FHT_N/2); // send out the data
124.  }
125.
126.  /**
127.   * Collect guitar signal and process using auto-
       correlate algorithm adapted from MrMark code
128.   * https://forum.arduino.cc/index.php?topic=540969.msg3687113#msg3687113
129.   * Result is stored in freq_per variable
130.   */
131.  void getGuitarFrequency() {
132.
133.     // Collect analog signal for autocorrelation
134.    for (count = 0; count < INPUT_SIZE; count++) {
135.      guitarInput[count] = analogRead(A2) >> 2;
136.    }
137.
138.    // Calculate mean to remove DC offset
139.    long meanSum = 0;
140.    for (count = 0; count < INPUT_SIZE; count++) {
141.      meanSum += guitarInput[count];
142.    }
143.    int mean = meanSum / INPUT_SIZE;
144.
145.    // Autocorrelation Logic
146.    sum = 0;
147.    int period = 0;
148.    for (i = 0; i < len; i++)
149.    {
150.      // Autocorrelation
151.      sum_old = sum;
152.      sum = 0;
153.      for (k = 0; k < len - i; k++) sum += (guitarInput[k] - mean) * (guitarInput[k + i]
       - mean);
154.
155.
```

```
156.      // Peak Detect State Machine
157.      if (pd_state == 2 && (sum - sum_old) <= 0)
158.      {
159.        period = i;
160.        pd_state = 3;
161.      }
162.      if (pd_state == 1 && (sum > thresh) && (sum - sum_old) > 0) pd_state = 2;
163.      if (!i) {
164.        thresh = sum * 0.5;
165.        pd_state = 1;
166.      }
167.    }
168.    // Frequency identified in Hz
169.    if (thresh > GUITAR_THRESHOLD) {
170.      freq_per = SAMPLE_FREQ / period;
171.    }
172.  }
173.
174.  /**
175.   * Sets the entire LED strip to the set color
176.   */
177.  void colorFill(uint32_t c){
178.    for(uint16_t i=0; i<strip.numPixels(); i++) {
179.      strip.setPixelColor(i, c);
180.    }
181.    strip.show();
182.  }
```