Haptic Headset Design Doc

Table of Contents

Introduction

The goal of the Hackable Haptic Headset is to design a headset that would allow the wearer to feel vibrations, or other forms or haptic feedback, while listening to music. The headset would be able to both generate feedback by following the music automatically, or by allowing a musician to program set vibrations for how they feel the user should “feel” the music. This project is like the many haptic feedback full body suits for VR that are currently available, such as the one developed by Teslasuit. However, this headset is designed for haptic feedback from music specifically.

Explanation of Project

The user of the headset should be able to put on some music, either from headphones or some speakers, and the headset. When the user plays music, specific zones on the headset will vibrate according to signals sent by Max. The headset itself won't be able to play music but only vibrate to the music. It should be an add on to music, like putting on some headphones, to feel more of the music. Figure 1 shows a concept design of the headset, without any haptic feedback devices attached.

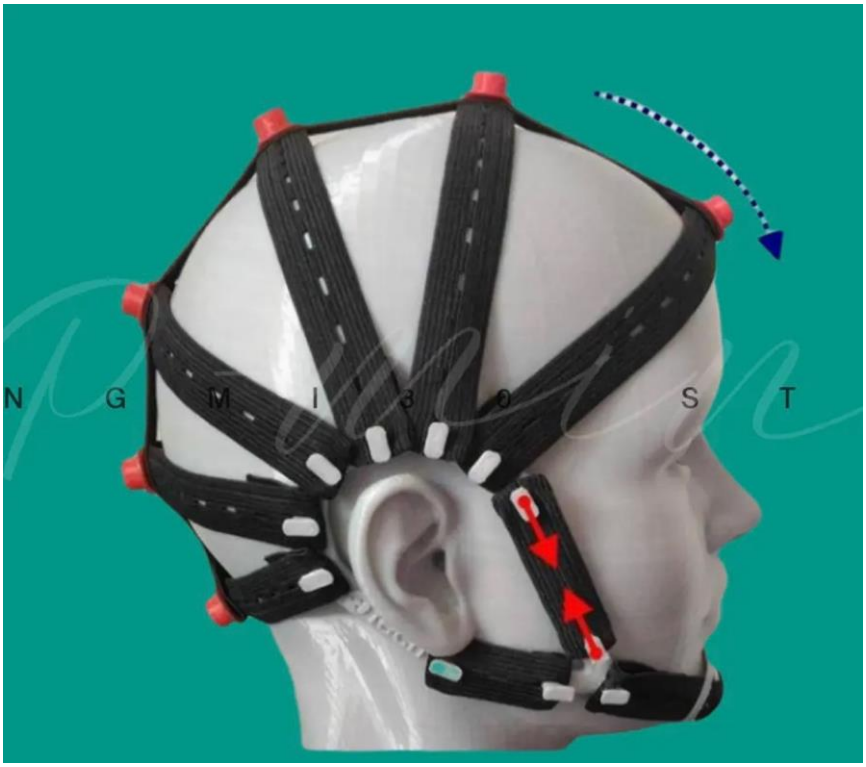

Figure 1 - Concept Design of the Haptic Headset

Design Broken Down

Frontend

The frontend of this project uses Max to allow the user to program the signal to the headset. Currently, the Max Patch takes the notes from a song, converts them to numbers specifying a zone and a strength, which will be read by the Arduino on the headset.

The UI (Figure 2) has 3 main parts: range selection, zone selection, and outputs. These sections allow the user to 'program' the headset by assigning different sounds in their song to different zones on the headset. The user can also see the signal strengths being sent to the headset as the audio plays.

The user has access to 6 filtergraph objects in Max that allow the audio input to be filtered out into separate ranges. The ranges have default center values of 30, 155, 375, 1250, 4000, and 13000 hz. The user may edit these ranges by clicking and dragging on the filtergraphs. There is also a button to reset the ranges to the default values.

After selecting the ranges they would like to filter for, the user will choose which headset zones each range sends a signal to. This is represented by a 6x12 matrix of buttons, where each row is a range, and each column is a zone corresponding to a vibration motor on the headset. When the user clicks the 'confirm' button, a message is sent to the Arduino that updates which zones are associated with each range.

Once the user-programmed settings are in place, the patch is ready to translate the audio into vibration signals. The audio signal is filtered by the 6 selected ranges, and the volume of each filtered signal is scaled into a number 0-127. A number 0 is a volume of –50 dB or lower, and a number 127 is –10 dB or higher. The user can see meters that represent these numbers on the screen as the audio plays, as well as the number below. These numbers are then computed into the proper 'signal message' form (as described in the Backend section) and sent through the serial port to the Arduino. There is also an option to change the refresh rate of the messages, in case lag is causing the audio and vibrations to become out of sync.

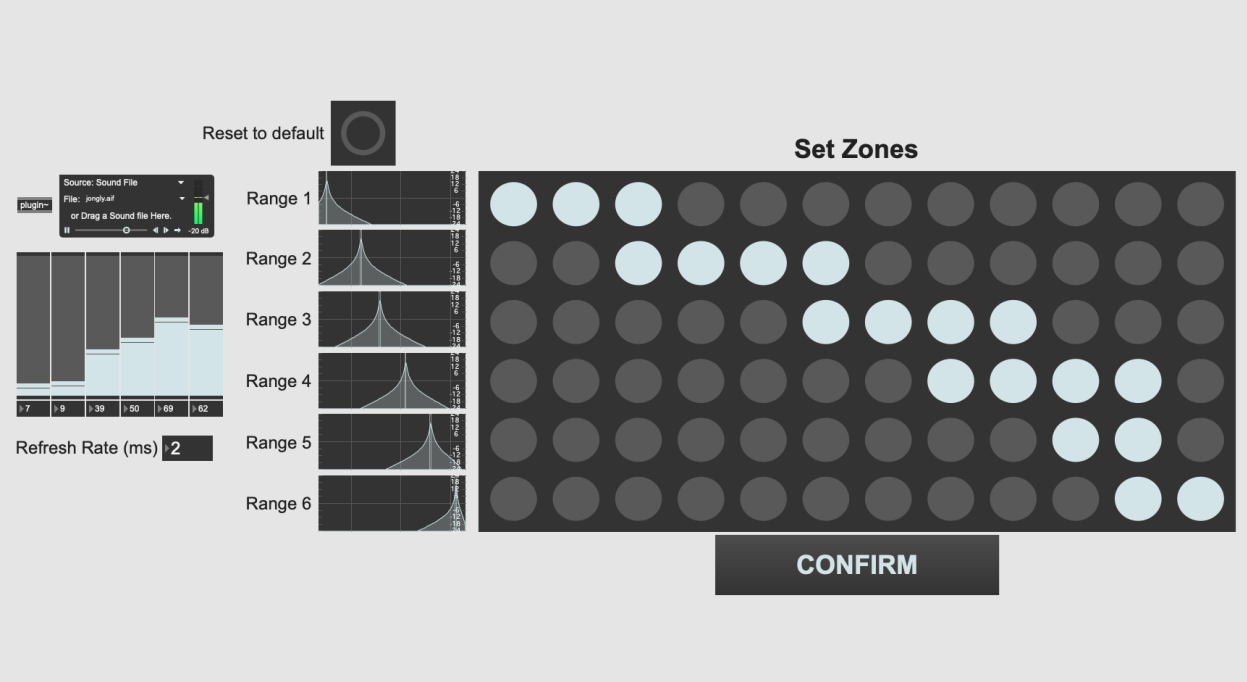

Figure 2 - Screenshot of Developer UI while music is playing

Backend

This will be a dive into the backend side of the project. To go over key words, when I refer to the zones that means each individual Vibrating Mini Motor and a Range is a group of zones. All of the backend code is in an Arduino project and is based in C/C++

The backend will receive a number from MAX though the serial port on the Arduino that can range from 0 through 32767 due to it being the max number the Arduino can store. When this number is passed in, it's treated as a binary number and translated as such. The first bit in the sequence is to tell what type of message we are dealing with. If the first bit is a 1 then the next twelve bits are what zones we wish to set, then the final 3 bits are for the range number which it translated back into a regular number. If the first bit is a 0 then the message from MAX is to play the ranges, the next 9 bits are for the Range number and the last 6 bits are for the strength that the range is going to be set/play to.

An example of a message would be this, the Arduino receives a number that translates to [1000001010111011]. Breaking this down we can see this [1,000001010111,011], we see the first bit is a 1 so that means that we are setting the 1,2,3,5 and 7th zones to the 3rd range. Another example would be [0,110001111,101010], here we are reading what strength we are playing on some ranges. The 1,2,3,4,8 and 9th ranges are being set to a strength of 42 (101010 converted to decimal).

```
//What the arduino should recive
//0-32767

//Message
//type   Zones   Range
// V       V       V
//[1,000000000000,000]   This sets zones


//Message
//type   Rages  Strength
// V       V       V
//[0,000000000,000000]   This send singal strength
```

Figure 3 – Explanation of Message Structure

An issue that we faced is that because MAX sends a whole number though its serial block, the Arduino treats it as a byte. Due to this, it limits the input message to 256 and no more. We only found this out late in development due to issues with connecting MAX to the Arduino.

Hardware

The hardware consists of 12 vibration motors attached to a fabric headset, in a 3x4 grid as shown by Figure 4. On the back of the headset is the Arduino, shown by the black box, to which the vibration motors will be attached. Due to the limited ports on the Arduino, the ground wires of the vibration motors are first connected to a mini breadboard, then to the Arduino. Currently, the headset must be wired to a computer to work. Possible improvements for this headset include a battery pack and wireless connection hardware so that the headset can be used wirelessly.

Table 1 shows the parts to be used for the headset. Currently, all but the headstraps and the Velcro roll have been ordered and are ready for assembly.

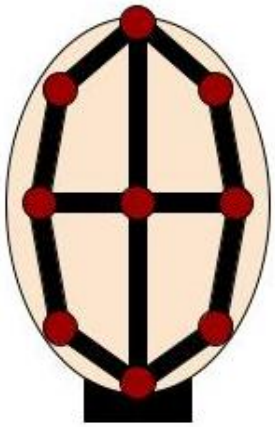

Figure 4 – Layout of Vibration Motors on the Headset

Table 1 – Haptic Headset Parts List

Part	Source	Quantity
Vibrating Mini Motor	Adafruit	12
Arduino Mega 2560 Rev3	Amazon	1
Nylon Headstraps	Aliexpress	1
Velcro Straps Roll	Cable Ties and More	1
Mini Breadboard	Adafruit	1
3” M/F Wires	Adafruit	20
12” M/F Wires	Adafruit	20

Conclusion and Future Improvements

Overall, we designed a Max Patch to handle the logic of encoding vibrations for a certain zone on the headset and sent those numbers to an Arduino, which then decodes those numbers as vibration frequencies for the specified zone. As of the end of us working on this project, the headset is not functioning fully, as we were not able to assemble the headset in time. Future goals for this project would be to assemble the headset and get it working on a base level, improve the function of the headset through better haptic feedback mechanisms undiscovered by us, and make the headset wireless so it does not need to be attached to a computer to function. Additionally, as the current capacity of variable feedback devices is 12, an upgrade to the Arduino that would allow more devices would be beneficial. After these improvements, we hope to see improvements in the feel of the headset, possibly by designing it in a way that looks like a typical hat, but still with all the functionality of the original design.