MyoMusic

Converting The contraction of muscles into sounds

Hardware

arduino uno
Sparkfun Myoware (with cable shield)
LED shield for alternate power source
buzzer

Buzzer: (27/8/18)

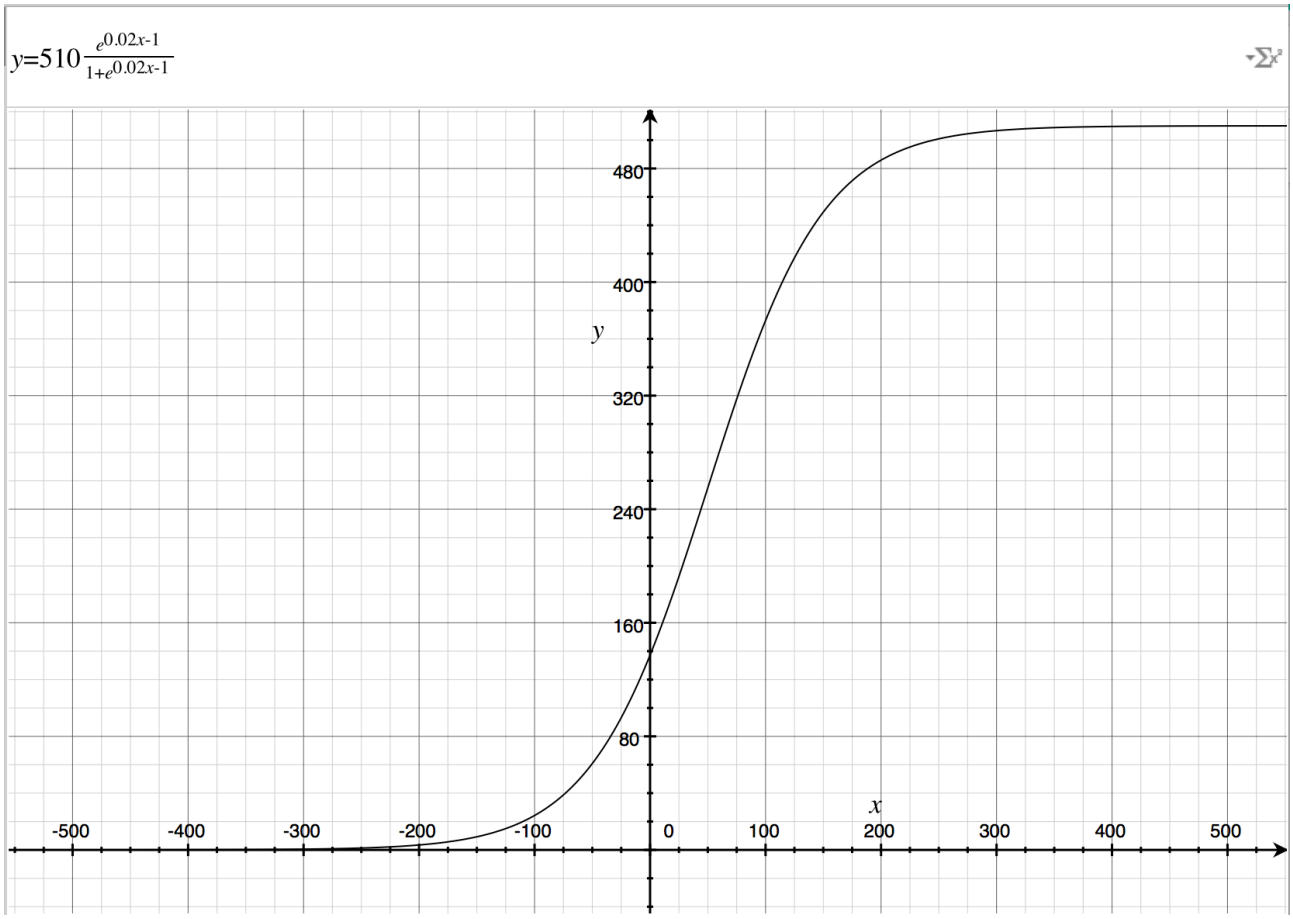
After hooking up the myoware to the arduino to get input readings, the next step was to do something with the data. The idea was to convert the raw input into audio frequencies.

Directly reading the output and converting it was a simple matter. However, the values had a tendency to behave rather digitally. When the muscle was loose, the output was low, but any contraction would cause the output to exponentially increase. In other words, the myoware's data favoured only the extreme values.

In order to contrast the more subtle muscle activations detected by the myoware and limit the data to 0-510 range so it can eventually be converted to MIDI, I used a sigmoid function.
The sigmoid function

$$\sigma(x) = e / (1+e^{(-x)}$$

Is a function that starts increasing slowly for low x values, speeds up at a certain point, then reaches a steady gradient, then begins to plate slowly. The sigmoid curve is shown below



What this means for the data is that all the extreme values x values (the raw input from the myoware) get capped at 0 and 510, but the middle values are contrasted. In other words, most of the output numbers in the range 0-510 are from the middle 200 raw input points. This allows much better control of the myowares data, and it seemed to work.

The next step after processing the raw data was to convert it into notes.
To do this, I followed an excellent guide on 8-bit sound generation with the arduino by David Cuartielles (https://www.elektormagazine.com/files/attachment/331). He developed a library for arduino (which is available by default in the IDE) wherein you can play frequencies on a buzzer without setting delays, etc.

I first discretised the number range into 8 steps, then assigned a frequency from the 4th octave to each step. Then using a set of if statements, i made the arduino play the corresponding note for 1 second when it detected each step. The code used is below:

Code:

```
// Myoware -> sound sketch

//pin 13 used for buzzer
int pin = 13;

void setup() {
   pinMode(pin, OUTPUT);
  Serial.begin(9600);

}

void loop() {
  // read input on A0
  float sensorValue = analogRead(A0);

  float x = sensorValue;

  //pass raw sensor value (which ranges from 150 - 800) through sigmoid function for midpoint contrast
  float modvalue = 510*((exp(0.02*x-1))/(510+exp(0.02*x-1)));

  Serial.println("Sensor: " + (String)sensorValue + ", Mod: " + (String)modvalue);

  //link muscle contraction data to notes by discretising input domain
  if((0 <= modvalue) && (modvalue <= 63.75)){
    tone(13, 261.63, 1000);
    Serial.println("C4");
  }
  else if((63.75 < modvalue) && (modvalue <= 127.5)){
    tone(13, 293.66, 1000);
    Serial.println("D4");
  }
  else if((127.5 < modvalue) && (modvalue <= 191.25)){
    tone(13, 329.63, 1000);
    Serial.println("E4");
  }
  else if((191.25 < modvalue) && (modvalue <= 255)){
    tone(13, 349.23, 1000);
```

```
    Serial.println("F4");
  }
  else if((255 < modvalue) && (modvalue <= 318.75)){
    tone(13, 392.00, 1000);
    Serial.println("G4");
  }
  else if((318.75 < modvalue) && (modvalue <= 382.5)){
    tone(13, 440.00, 1000);
    Serial.println("A4");
  }
  else if((382.5 < modvalue) && (modvalue <= 446.25)){
    tone (13, 493.88, 1000);
    Serial.println("B4");
  }
  else if((446.25 < modvalue) && (modvalue <= 510)){
    tone (13, 523.25, 1000);
    Serial.println("C5");
  }
  else {

  }

}
```

Scales: (29/08/18)

The next step was to implement scales. The code above only played the C scale, but if you want to produce music from your muscles you would want to choose the scale.

To do this, I moved the whole if-else block into its own method that took two arguments: scale (as an array of frequencies) and modvalue which is processed raw data.

The code then looks like this.

```
// Myoware -> sound sketch

int pin = 13;

float C4scale[] = {261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25};
float B4mscale[] = {493.88, 554.37, 587.33, 659.25, 739.99, 783.99, 880.00, 987.77};

//function that discretizes data range and assigns a note within defined scale to each step
void linkTones(float scale[], float modvalue){
 if((0 <= modvalue) && (modvalue <= 63.75)){
    tone(pin, scale[0], 1000);
    Serial.println("C4");
  }
  else if((63.75 < modvalue) && (modvalue <= 127.5)){
    tone(pin, scale[1], 1000);
    Serial.println("D4");
  }
  else if((127.5 < modvalue) && (modvalue <= 191.25)){
    tone(pin, scale[2], 1000);
    Serial.println("E4");
  }
```

```
  else if((191.25 < modvalue) && (modvalue <= 255)){
    tone(pin, scale[3], 1000);
    Serial.println("F4");
  }
  else if((255 < modvalue) && (modvalue <= 318.75)){
    tone(pin, scale[4], 1000);
    Serial.println("G4");
  }
  else if((318.75 < modvalue) && (modvalue <= 382.5)){
    tone(pin, scale[5], 1000);
    Serial.println("A4");
  }
  else if((382.5 < modvalue) && (modvalue <= 446.25)){
    tone (pin, scale[6], 1000);
    Serial.println("B4");
  }
  else if((446.25 < modvalue) && (modvalue <= 510)){
    tone (pin, scale[7], 1000);
    Serial.println("C5");
  }
  else {

  }
}


void setup() {
  // put your setup code here, to run once:
  pinMode(pin, OUTPUT);
  Serial.begin(9600);

}

//Bröther may I have some lööps
void loop() {
  // read input on A0 from Myoware
  float sensorValue = analogRead(A0);
  delay(100);

  //assign read value to float
  float x = sensorValue;

  //applies sigmoid activation function to process data
  float modvalue = 510*((exp(0.02*x-1))/(510+exp(0.02*x-1)));

  //Prints both the raw sensor value and the processed value for debugging
  Serial.println("Sensor: " + (String)sensorValue + ", Mod: " + (String)modvalue);

  //method uses processed data to play tones within defined scale
  linkTones(B4mscale, modvalue);

}
```

Here the method linkTones again discretises the data range of mod value (0-510), and assigns each of the 8 discrete steps a frequency from the scale array. It does so by accessing each index of the array one at a time for each ascending step of the discretised data.

The method is then called with the scale you want to use. I have defined two scales in the above code: C4, and B4 minor.

Tempo (29/08/18)

The tempo of the song is how quickly the notes change. A faster tempo song will have a higher bpm (beats per minute) than a low tempo song. In classical, there are a range of tempos going from largissimo (very slow, 24 bpm) to prestissimo (very fast, 200+ bpm).

To increase the variety of music you can make with myomusic, you should be able to choose the duration of the music your'e making based on a bpm value you define as well as note durations that are relative to each other.

I used Cuartielle's formula for setting note duration based on bpm:

time = (1000 /bpm*120)/pow(2,duration);

Where duration is a number between 1 and 7 that represents the standard note conventions; 1 being a whole note, and 7 being a hemi demi semi quaver (1/64th of a whole note). At 120 bpm, the duration of a whole note is 500 ms.

Using this formula you can play complex melodies be keeping a steady melody but varying the frequency and durations of each note you play.

Notes:

With myomusic its not as simple as playing a predefined melody. The myoware sends out a steady stream of data that may or may not be used depending on whats currently happening in the code. If the code is currently playing a note with duration **x**, the code will only sample the myoware's data stream once **x** ms are over. Therefore, predefining the pattern of notes forces you to match your muscle contractions to the tempo you defined previously.

This is useful for sampling the myoware at unorthodox and interesting rhythms, but the purpose of the myoware is for it to be used as an instrument with all the versatility you would expect of one.

The biggest obstacle at the moment is the fact that the myoware does not "hold" notes. Once you clench your fist and the myoware detects it and the data peaks, it begins to fall immediately. Keeping it clenched does not sustain the note.

This is a phenomenon that has to do with biology, so we need a mathematical function that will sustain the notes for any duration until a change is detected, but will not limit the changing of notes up and down the scale too much.

Possible: implement a potentiometer to vary bpm

Idea: contraction peaks are similar to beats. Flexing muscles could be used as a percussion instrument. Depending on the muscle, could come to some interesting patterns (walking -> legs, air drumming-> triceps)

The formula is implemented as its own method

```
long tempo(int duration, int bpm) {
  long t = (1000/bpm*120)/pow(2,duration);
  return t;
}
```

The duration quantity in the linkNotes method is then changed to call this method, and **int bpm** is added as a new method parameter.

For example:

```
void linkTones(float scale[], float modvalue, int bpm){
 if((0 <= modvalue) && (modvalue <= 63.75)){
    tone(pin, scale[0], tempo(1,bpm));
    Serial.println("C4");
  }
  else if((63.75 < modvalue) && (modvalue <= 127.5)){
    tone(pin, scale[1], tempo(1,bpm));
    Serial.println("D4");
  }
```

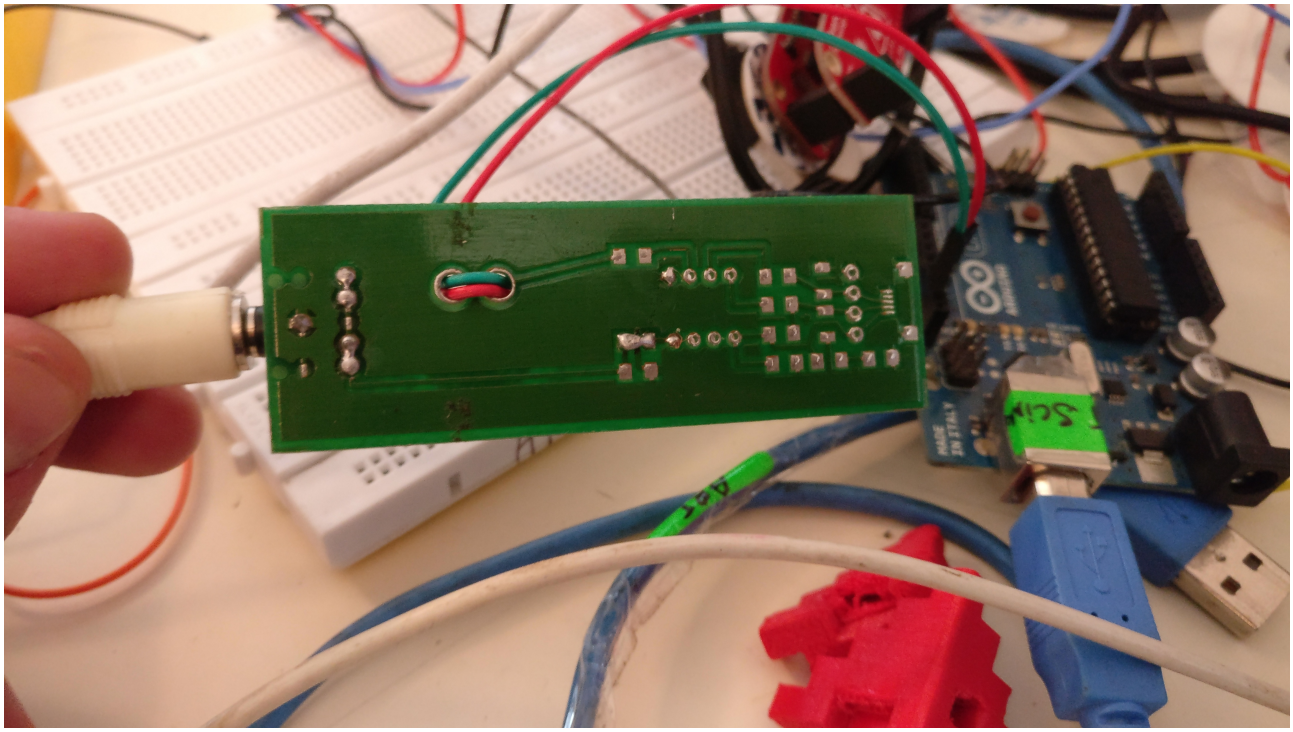
And so on.

Speakers 31/08/18

The next step for me was to connect the arduino to a speaker so I could get volume control and better sound. This was a simple process as all I had to do was wire a 3.5 mm audio jack socket to the digital out pin that was previously used for the buzzer.

This was done by shorting all the signal pins of the socket, and connecting this to the digital out of the Arduino. The ground of the socket was then connected to the ground pin of the Arduino.

Then a speaker was plugged into the socket, as you would plug it into any jack.



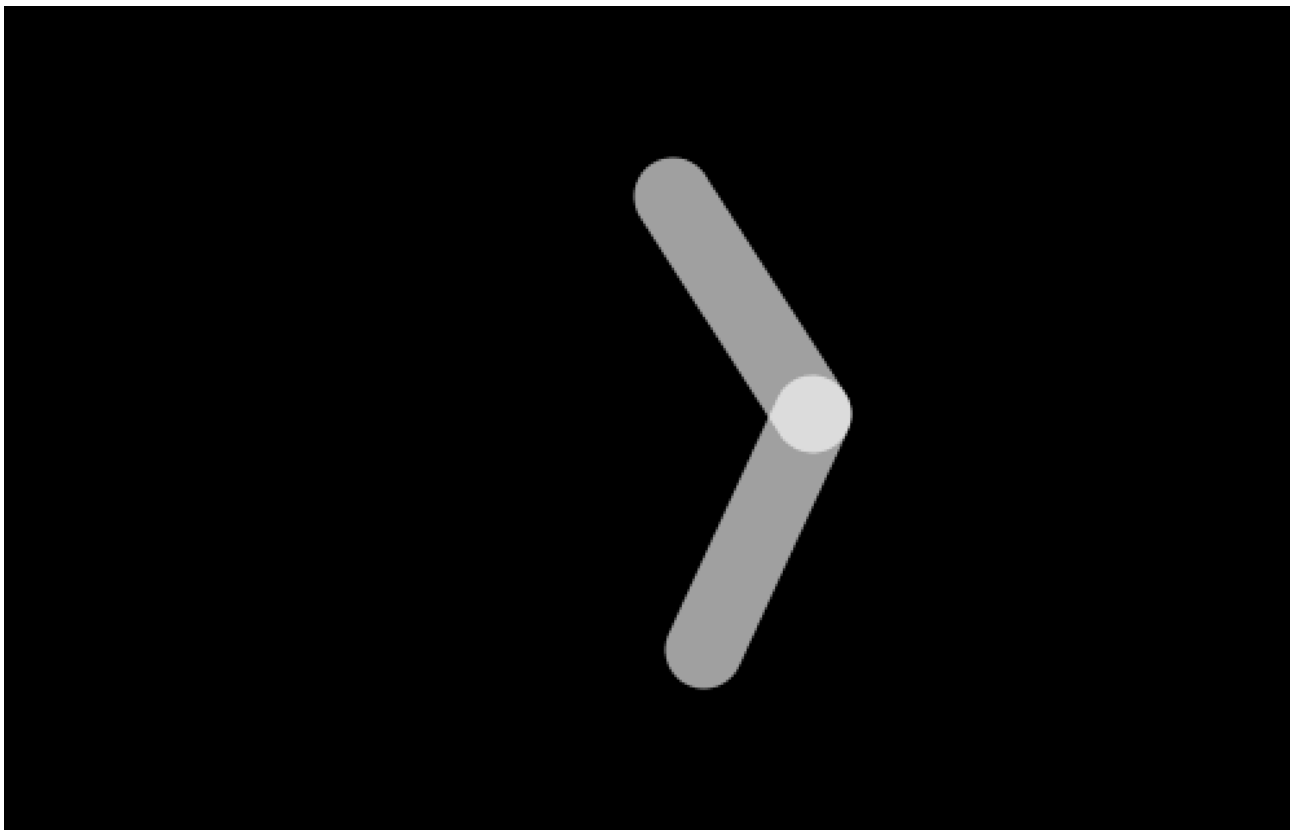
Interlude 04/09/18

I took a short break from the music side of things to see if I could use the myoware's data in another way.

I hooked up the myoware to the arduino, and made the arduino send out the myoware's (modified) data via serial.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  float sensorValue = analogRead(A0);
  delay(5);

  float x = sensorValue;

  float modvalue = 510*((exp(0.02*x-1))/(510+exp(0.02*x-1)));

  Serial.println((String) modvalue);
}
```

I then made a processing script that takes in this data, and uses it to control a virtual arm, such that when I flex my bicep, the virtual arm also bends.



The script was adapted from one of the examples in processing. It takes in the numerical input from the arduino, converts it into a float, and divides it by 510 to change the 0-510 range to 0-1. This is because the “bicep” position of the virtual arm is controlled by an angle in radians. Therefore, the maximum flexing of the arm is 1, and the minimum is 0.

This angle is then implemented in the draw() loop to match the data.

```
import processing.serial.*;

//defines port
Serial myPort;

String val;

//initializes
float x, y;
float angle1 = 0.0;
float angle2 = 0.0;
float segLength = 100;

void setup(){
  //sets background size
  size(1400, 700);
  //creates strokes with specific colour and thickness to represent segments of arm
  strokeWeight(30);
  stroke(255, 160);

  //sets position of fake arm
  x = 750;
  y = 300;
  //port initialization and constructor
  String portName = Serial.list()[4];
  myPort = new Serial(this, portName, 9600);
}

float v = 0.0;

void draw(){
  //sets background color to black
  background(0);
  //if port is available,
  if (myPort.available() > 0) {
    val = myPort.readStringUntil('\n');
  }
  try {
    v = float(val);
  }
  catch(NullPointerException e) {
  }

  //println(v);

  angle1 = 1;
  angle2 = v/510;
  println(angle2);

  pushMatrix();
  segment(x, y, angle1);
  segment(segLength, 0, angle2);
```

```
  popMatrix();

}

void segment(float x, float y, float a) {
  translate(x, y);
  rotate(a);
  line(0, 0, segLength, 0);
}
```

Unfortunately, the flexing only seems to work once. The data input has a delay compared to the action, and the readings are not as steady as when viewed in the arduino IDE. Furthermore, after the first flex, it undergoes a sort of oscillating behaviour (possibly due to noise).

Noise and the Faraday Cage (07/09/18)

Throughout the process of creating the myomusic setup, I was faced with days when the data refused to "behave". It was not clean, and any spikes were drowned out by massive noise. This noise was repeating, and typically consisted of values between 0 and 1000 oscillating at a frequency of about 1 Hz.

This noise was present even if I removed the myoware and left the arduino as is. It could be amplified by adding a single wire to any of the analog inputs, which would act as an antenna.

I discovered that the first main source of noise was my laptop charger. However, even switching that off didn't help sometimes, and there would still be noise. I discovered that this noise came from powerlines, soldering irons, tubelights, and a range of other devices that used strong electrical oscillations.

To remove the noise I had to build a faraday cage.

A faraday cage is essentially a metal cage. According to the laws of electromagnetism, any solid hollow conductor will have no electric field inside it. This is because any electric field across the conductor will cause electrons to move away from due to electromagnetic repulsion and attraction. However, the imbalance of electrons on on either side of the conductor will cause a difference in charge between the two sides. Any difference in charge generates an electric field within the conductor, just like it does in a capacitor. It turns out that the outside field and the inside field will always perfectly cancel each other out, so the inside of the conductor will have no electric field.

Interesting article on the mathematics of faraday cage constraints: https://www.turing-gateway.cam.ac.uk/sites/default/files/asset/doc/1703/FaradayTalk%20-%20Chapman.pdf

However, the faraday cage only minimally decreased noise. It did not remove the noise from my laptop charger. It did so, however, when I touched it. I assumed that this was due to charge accumulation on the surface of the cage, which effectively meant that the grounding of the cage was off. I had used the arduino's GND pin to ground the cage, so I had to find an alternative. I discovered that connecting the reference electrode pin of the myoware acted as a better ground, but in return, it changed the signal slightly. It attenuated it so it would not be larger than 500 units.

While this worked, it was not reliable. I needed a way to make sure that any and all sources of noise would be blocked, leaving only the raw signal. To do this, I attempted to analyse and process the signal and its frequncies.

All signals, no matter how complex can be reconstructed mathematically from single frequency sinusoidal waves. These signals can therefore be said to consist of these single frequencies. The

frequencies making up a signal can be found through a mathematical function called fourier transform.