# **Final Project: Net Boat Acceleration Visualization and Sonification**

## **Abstract:**

This project will explore the basic functionality of the MPU-6050 accelerometer and how it can be used to visualize and interpret real time acceleration date. Specifically, this project will look to emulate lab based training tools and exploit their functionality using both visual and auditory methods to convey the data.

## **Background:**

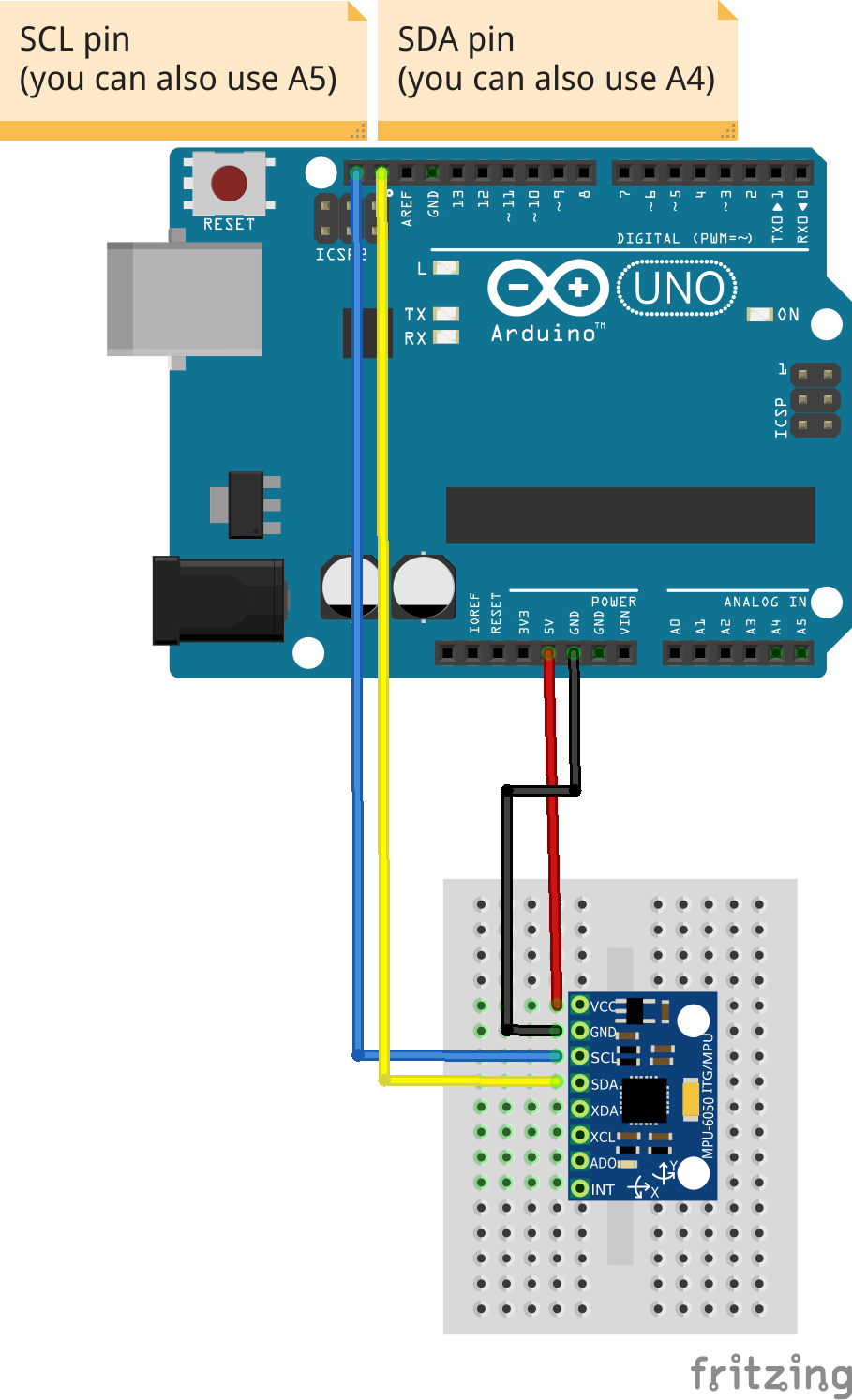
Despite the seemingly simplistic nature of the rowing motion, the rowing stroke carries with it a remarkable amount of technical nuances. A rowing race can be won and lost based on an individual’s technical proficiency. Due to the importance of technique, considerable research has been conducted to quantify variations between athletes and their impact on performance. Specific portions of the rowing stroke have been isolated in order to provide more actionable insights. Drive based force curves are a good example of this as they exclusively look at the force produced in the drive phase and do not consider the three other force pairs also acting during the drive. A growing area of research, albeit one that is considerably less understood, is the translation of the net boat acceleration into similar actionable insights. As the name suggests, the net boat acceleration is the total acceleration of the hull of a boat and is defined by the sum of all of the forces acting on the boat. Although not as simple as a force curve, the net boat acceleration provides a more complete picture of the rowing stroke. Specifically, the net boat acceleration provides insight into the recovery phase of the stroke and also exemplifies timing disparities between rowers. Due to the relative novelty of net boat acceleration as a coaching tool, access to its insights is largely limited to biomechanics labs. The primary purpose of this project is to bridge this gap and produce similar results to those found in biomechanics labs using the techniques and knowledge I have gained throughout the course. To do this I will use an Arduino Uno in conjunction with a common accelerometer. Upon creating a functional accelerometer circuit I have two further goals: generate a graphical representation of the accelerometer outputs and then produce a sonified output that is scaled by the accelerometer outputs.

## **Accelerometer Background:**

For this project, I will be using the GY-521 three-axis accelerometer and gyroscope. This chip has a wide range of functionalities ranging from outputting raw acceleration data to ambient temperature values. Given the primary goal of this project is to look at the net acceleration of a boat this discussion will primarily focus on the acceleration data collection. Simple accelerometers generally make use of a mass on a sensitive axis. When the chip is accelerated the mass moves and exerts a force on a highly sensitive material that outputs an electric charge proportional to the amount of force that is being exerted on it.

## **Circuit Setup and Design:**

The setup of my circuit is remarkably simple and solely consists of wiring the MPU-6050 to the Arduino properly. There are a few nuances that should be noted. First, we will only need to make use of the first 4 pins on our MPU-6050 to suit our needs for this project. These are the VCC, GND, SCL and SDA pins. VCC runs directly into the 5V output pin and GND runs directly to GND as we would expect. It is important to note that VCC can also be run by the 3.3V output pin. Next, we recognize that our Arduino is not equipped with SCL or SDA pins. As such, we will use A5 and A4 as proxies for SCL and SDA. (The circuit diagram below shows the SDA and SCL pins running directly to their respective SDA and SCL pins on the Arduino board. We will use A4 and A5 respectively)



Now that we have created our circuit, we need to write some code to make use of its functionalities. Due to the extensive amount of code that will go into this problem and the incremental nature of each code addition this project will be broken down into chunks of code. The first chunk of code will consist of the basic acceleration measurements. Given that everything else will revolve around these measurements it makes sense to use them as a starting point.

## **Getting Acceleration Data:**

The MPU-6050 board is an extremely common accelerometer and has a variety of libraries that we can pull from. For this project, I will make use of the “tockn” library. Despite being one of the more simplistic libraries, this library provides us with everything necessary in order to access the acceleration metrics we desire. After adding this library to our directory and ensuring that each connection on our breadboard matches our source code we produce the following code to read our acceleration data on our Serial monitor:

**-------------------------------------------------------------------------------------------------------------------------------**

#include <MPU6050\_tockn.h>

#include <Wire.h>

MPU6050 mpu6050(Wire); //using our MPU6050 library as a reference, we define our mpu6050 function based on the Wire connection.

float accSum = 0; //defines initial acceleration to be zero

void setup() {

Serial.begin(9600); //starts serial monitor

Wire.begin(); //starts to read the input dats from the MPU6050

mpu6050.begin(); //starts the chips accelerometer, gyroscope and temperature readings

}

void loop() {

mpu6050.update(); //VERY IMPORTANT, update starts the MPU6050 to start to send in live data

Serial.print("accX : ");Serial.print(mpu6050.getAccX()); //retrieves and prints X acceleration

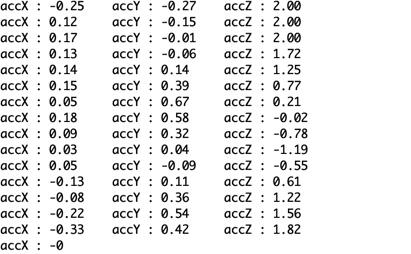
Serial.print("\taccY : ");Serial.print(mpu6050.getAccY());//retrieves and prints Y acceleration

Serial.print("\taccZ : ");Serial.println(mpu6050.getAccZ()); //retrieves and prints Z acceleration

}

**-------------------------------------------------------------------------------------------------------------------------------**

The above code is quite simple and relies on the built-in library functionality in order to quickly and accurately retrieve the real-time acceleration data. In order to check to make sure that it is working properly we open up the Serial monitor, give the accelerometer a quick shake, and view the output:



As we can see our accelerometer is functioning properly! The values being outputted are in terms of g’s. When we simply leave our accelerometer resting on the table we find the output to be approximately 1 g in the Z direction. Given that the Z-direction is measuring the vertical acceleration this makes sense as the acceleration due to gravity is 1 g. While having access to the acceleration values in our 3 planes is great, we are more interested in their sum as that is what will tell us about the net acceleration. Thus, we will now add a net acceleration term into our previous code. We produce the following:

**-------------------------------------------------------------------------------------------------------------------------------**

#include <MPU6050\_tockn.h>

#include <Wire.h>

MPU6050 mpu6050(Wire);

float accSum = 0;

void setup() {

Serial.begin(9600);

Wire.begin();

mpu6050.begin();

}

void loop() {

mpu6050.update();

Serial.print("accX : ");Serial.print(mpu6050.getAccX());

Serial.print("\taccY : ");Serial.print(mpu6050.getAccY());

Serial.print("\taccZ : ");Serial.println(mpu6050.getAccZ());

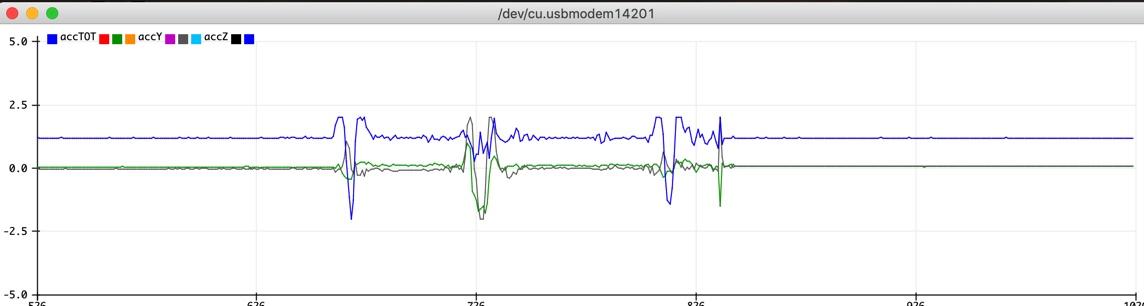
Serial.println("accTOT : ");Serial.print(sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()) + (mpu6050.getAccZ())\*(mpu6050.getAccZ())));

}

**-------------------------------------------------------------------------------------------------------------------------------**

Nothing has changed in the code above except for the very last line. Due to the already messy nature of the line, I felt that it would be neatest to place my comments on the code in this section as opposed to coming immediately after the code. Finding the total acceleration proved to be surprisingly more difficult than I had initially thought. Mathematically it is a rather simple operation involving the squaring of each term and then the square root of their individual sums. The problem with this is that you cannot XOR floats. In order to get around this inconvenient fact, instead of raising each individual component to the second power we simply need to multiply them out by hand. The resulting output works as expected. When the accelerometer is at rest we see that the net acceleration is equal to the acceleration in the z-direction which is approximately 1 g. When we move and shake the accelerometer the output changes accordingly.

We can visually verify that the accelerometer is fully set up by looking at the serial plotter feature that is built into the Arduino library. This will also serve as a brief foray into our next section: Visualizing Acceleration Data. Using the data plotter we receive the following output:

  
There are a few key features to note in the plot above. First, the horizontal lines at the start and end of the graph correspond to when the accelerometer is at rest on the top of the table. The first movement of the accelerometer is a sharp vertical movement, this is clearly reflected in the immediate drop in the z-acceleration as it is accelerating against gravity. After this, we have a brief set of side to side shakes and then another vertical one.

Just before moving on, we will create one more acceleration output function. Similar to the previous section it will be the sum of the acceleration components, however, this time we will only use the X and the Y. The reason for this is because the rowing motion is linear. As such, measuring vertical acceleration is a pointless metric, in fact, by including it into our measurement we are effectively just shifting everything up by 1. Thus, we produce the following code:

**-------------------------------------------------------------------------------------------------------------------------------**

#include <MPU6050\_tockn.h>

#include <Wire.h>

MPU6050 mpu6050(Wire);

float accSum = 0;

void setup() {

Serial.begin(9600);

Wire.begin();

mpu6050.begin();

}

void loop() {

mpu6050.update();

Serial.print("accX : ");Serial.print(mpu6050.getAccX());

Serial.print("\taccY : ");Serial.print(mpu6050.getAccY());

Serial.print("\taccZ : ");Serial.println(mpu6050.getAccZ());

Serial.println("accTOT : ");Serial.print(sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()) + (mpu6050.getAccZ())\*(mpu6050.getAccZ())));

Serial.println("accXY : ");Serial.print(sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()) ));

}

**-------------------------------------------------------------------------------------------------------------------------------**

The only new addition to this code is the bottom-most line. It is almost a mirror image of the line above it, however, it does not include the z-axis acceleration data. When we view the output in the serial plotter we find that at rest the net acceleration in the x and y direction is approximately 0.

Before moving on to our output and sonification sections we go back and clean up our code by defining our acceleration outputs to be float variables as opposed to calling them directly. Our final base code is the following:

**-------------------------------------------------------------------------------------------------------------------------------**

#include <MPU6050\_tockn.h>

#include <Wire.h>

MPU6050 mpu6050(Wire);

float accSum = 0;

void setup() {

Serial.begin(9600);

Wire.begin();

mpu6050.begin();

}

void loop() {

mpu6050.update();

float accX = mpu6050.getAccX();

float accY = mpu6050.getAccY();

float accZ = mpu6050.getAccZ();

float accXYZ = sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()) + (mpu6050.getAccZ())\*(mpu6050.getAccZ()));

float accXY = sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()));

//Serial.print("accX : ");Serial.print(accX);

//Serial.print("\taccY : ");Serial.print(accY);

//Serial.print("\taccZ : ");Serial.println(accZ);

//Serial.println("accTOT : ");Serial.print(accXYZ);

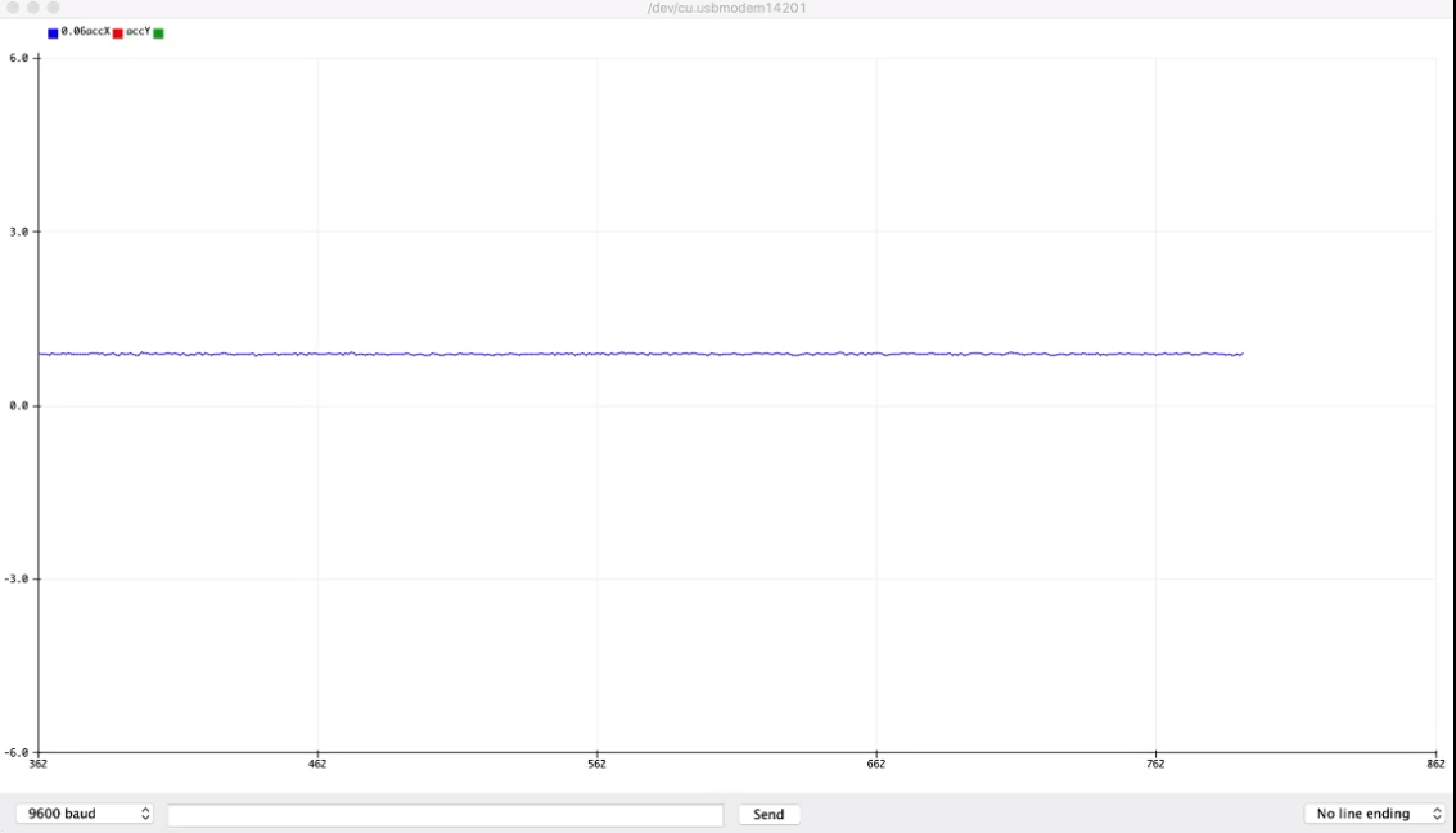
Serial.println("accXY : ");Serial.print(accXY);

}

**-------------------------------------------------------------------------------------------------------------------------------**

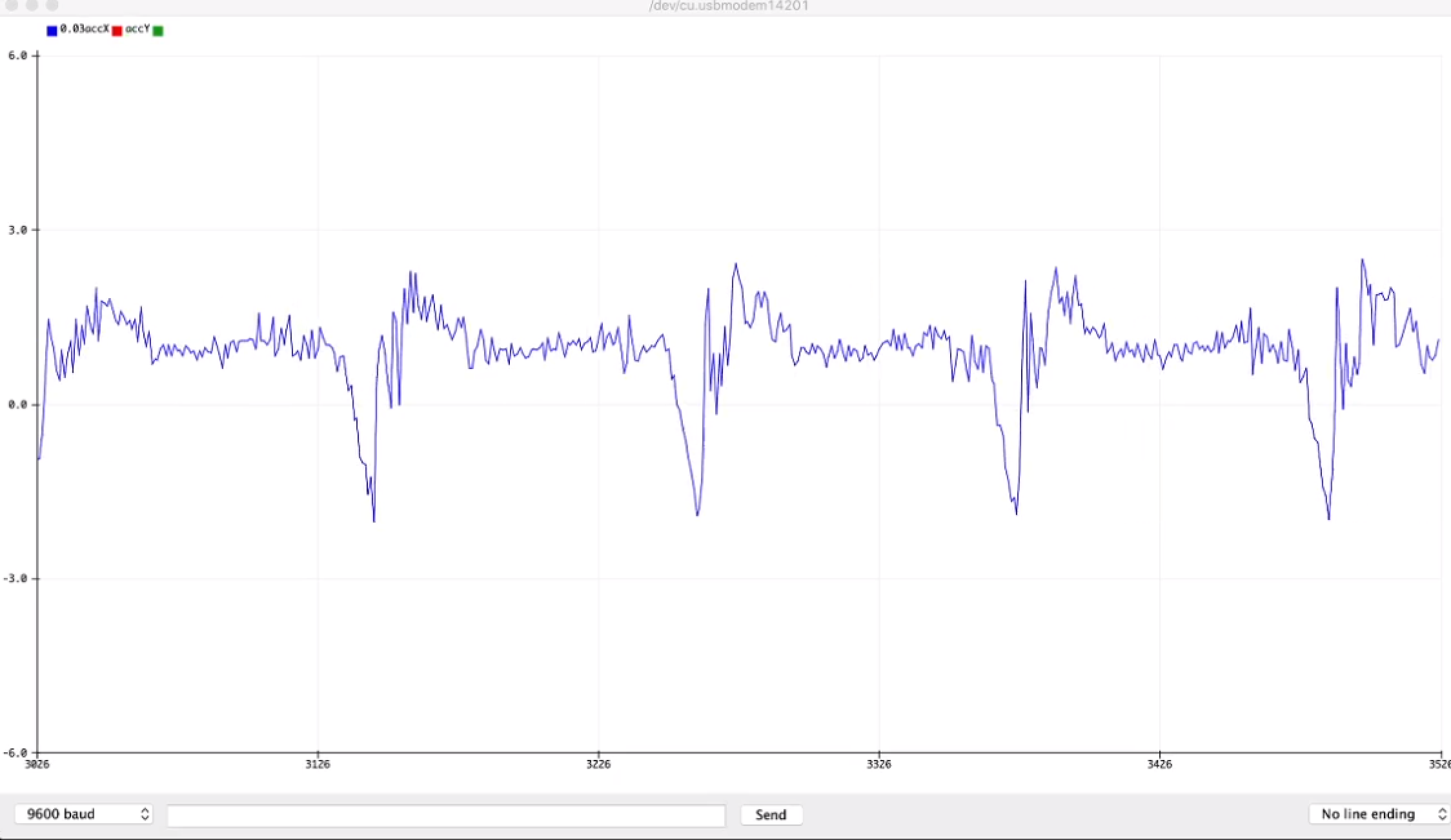
## **Output:**

Plotting our acceleration data is a remarkably simple process. Due to the built in serial plotter functionality of the arduino uno, we can plot our real time acceleration data simply by adding ln to our Serial.print() command: Serial.println(). By simply //the additional outputs out we can focus our Serial Plotter on just one piece of acceleration data. Due to the linear nature of the rowing stroke we can simply choose the X direction and orient our accelerometer accordingly. When we upload the code to our Arduino, open up the Serial plotter and leave the accelerometer at rest we receive the following output:



There are two key things to note here. First, we can see that the accelerometer is providing a stable, constant output. This makes sense as the accelerometer is in fact at rest and there should be no acceleration in the X direction. Second, it is important that we acknowledge that the accelerometer is not reading 0 g’s. This implies that there is some innate offset for our accelerometer. The primary purpose of this project is to evaluate the change in acceleration over time. As such, we do not need to correct for this shift as the change in acceleration will not be affected.

Now it is time for us to see how our accelerometer behaves when it is subjected to the actual motion that it is meant to track. We firmly tape our arduino and accelerometer directly to the base of our erg that is on sliders. The sliders allow us to simulate the acceleration of the boat without any of the inherent risks that come with electronics and water. We then upload our program and take strokes at 220 watts and a stroke rate of 18. We receive the following output:



Above we can see the real time acceleration outputs from four and a half strokes. While there may be some noise with our output we can clearly make out the characteristic phases of the rowing stroke: the catch, drive and recovery. These three characteristic phases are diagrammed below.



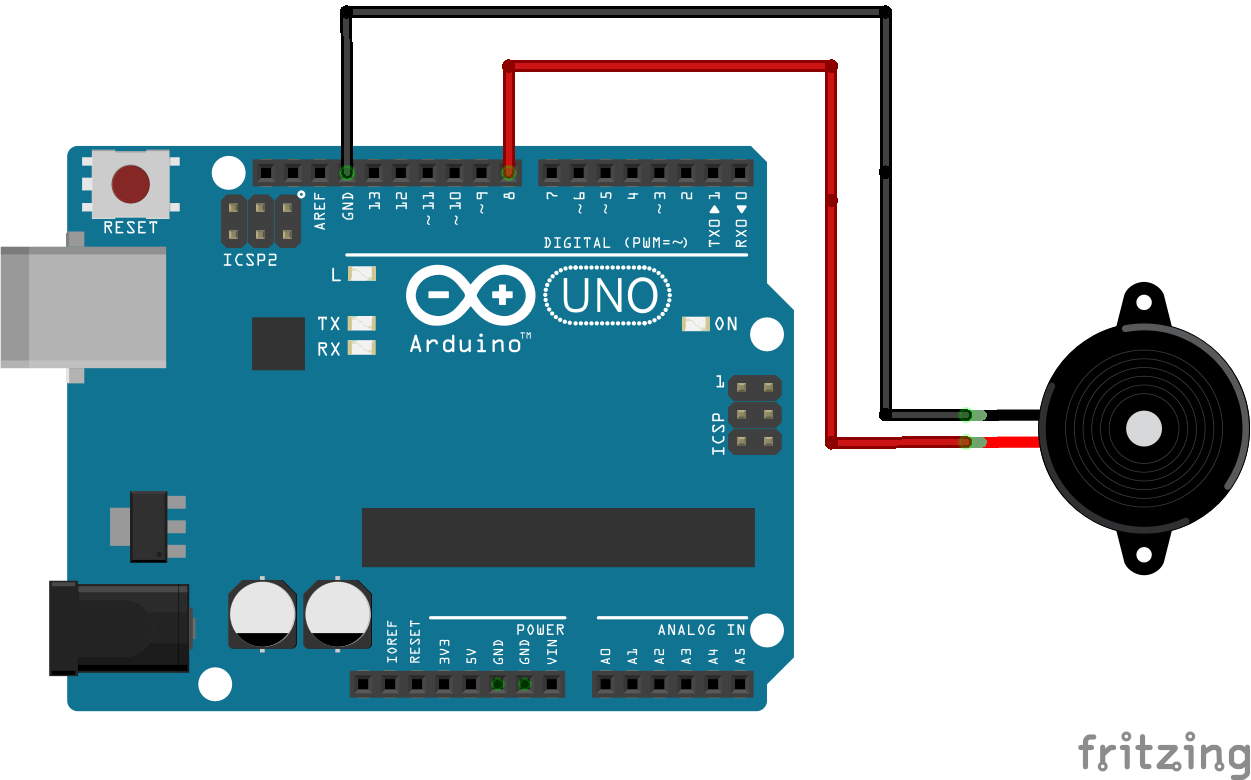
There are some notable features that should be noted from each phase. First, the catch is marked by a rapid and distinct negative acceleration. Significant amounts of time are put into honing a catch such that it is both quick and smooth so as to minimize the amount of time spent in this undesirable yet inevitable state. Next, we look at the drive phase which comes directly after the catch and is distinguishable by its large and relatively smooth positive acceleration. The drive phase force profile is one section of the stroke that has been extensively studied yet there is not a generally held best force profile. Instead, it is generally agreed upon that the force profile should be smooth and relatively bell shaped. The last phase we can look at is the recovery phase. This is a phase of the stroke in which nothing can technically be gained, however, much can be lost. The recovery phase is characterized by a long period in which ideally the only force acting on the boat is the water resistance. Subtle movements by the rower can and will increase this resistance. As such, an ideal recovery consists of minimal fluctuations in acceleration so as to hold on to the speed generated by the drive phase.

## **Sonification Background:**

Sonification is simply the translation of data into audio in order to convey information. It is a highly useful tool for training and reinforcing motor patterns. Returning to the initial premise of this project, which is to emulate the training tools present in large biomechanics lab, our goal is to utilize the sonification of our net boat acceleration data in order to exemplify the technical features and flaws of a given stroke. To do this we are going to utilize a simple 16 Ohm speaker and make use of the Arduino’s built in tone() function. The Arduino’s built in tone() function makes use of PWM to generate square waves at whatever input frequency is specified thus producing a corresponding audio output. The difficult part within the context of this project is deciding on a base frequency and a scalar such that our acceleration data produce meaningful pitch fluctuations that remain within a reasonable range. To do this we rely on a few assumptions, the primary range of “optimal” sound frequencies is between about 120Hz and 1250Hz. While most humans can hear frequencies between approximately 20Hz and 20000Hz this range is where the majority of sounds occur at. As such, we would like our values to fall somewhere between 120Hz and 1250Hz. We know that our accelerometer while capable of producing much higher values will only be subjected to about 2 g’s at the most. Hence, if we choose a value somewhere in the middle of this range to be our base frequency, we can use a scaling factor on the order of 100.

## **Sonification Setup:**

Going with the simplistic circuit design theme we saw in the first section, the wiring for our circuit is quite simple: the ground pin runs to ground and the input runs directly from one of our digital Arduino output pins.



## **Implementation:**

As we discussed in the background section, the built in tone() function will be our primary tool for sonifying our data. Our tone function takes in two inputs, the first being a pin and the second being a specific frequency. We wired our speaker directly into pin 5 so we will be using that as our speakPin. In order to actually sonify our data, we need to create a function that relates the g-force outputs from our accelerometer and scales them into appropriate frequencies. The basic form of this function is our standard point slope form with our initial frequency value being our point of oscillation. As we mentioned above, the range of frequencies that we are looking to target is from 120Hz to 1250Hz. After some trial and error we find that 500 seems to produce a sound that is both near the middle of our range and is not unbearable to listen to. Now we must tackle our scaling issue. We know that our accelerometer values range from approximately -2 to +2 g’s. The greater range in frequencies that we can produce the easier it will be to use as a functional training tool. Hence, since we chose 500Hz to be our base frequency, we choose 200 to be our scaling factor. This gives us a working range of 100Hz to 900Hz. Implementing this into our code we are left with:

**-------------------------------------------------------------------------------------------------------------------------------**

#include <MPU6050\_tockn.h>

#include <Wire.h>

MPU6050 mpu6050(Wire);

float accSum = 0;

void setup() {

Serial.begin(9600);

Wire.begin();

mpu6050.begin();

}

void loop() {

mpu6050.update();

float accXY = sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()));

int speakPin = 5;

int Frequency = 0;

Serial.println("accX : ");Serial.print(4\*mpu6050.getAccX());

Serial.println("\taccY : ");Serial.print(mpu6050.getAccY());

//Serial.print("\taccZ : ");Serial.println(mpu6050.getAccZ());

//Serial.println("accTOT : ");Serial.print(sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()) + (mpu6050.getAccZ())\*(mpu6050.getAccZ())));

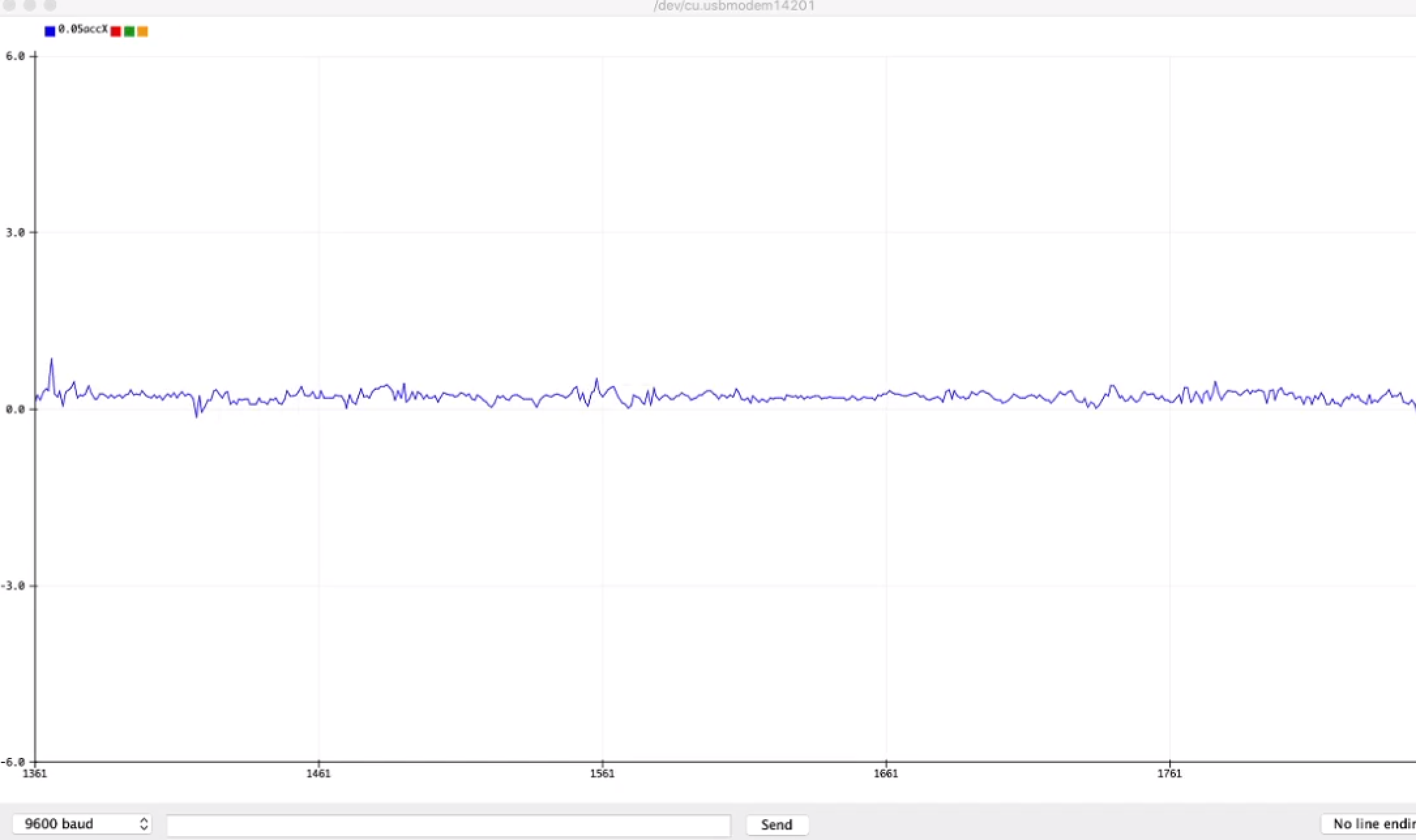
//Serial.println("accXY : ");Serial.print(sqrt((mpu6050.getAccX())\*(mpu6050.getAccX()) + (mpu6050.getAccY())\*(mpu6050.getAccY()) ));

tone(speakPin, 500 + (accXY\*200));

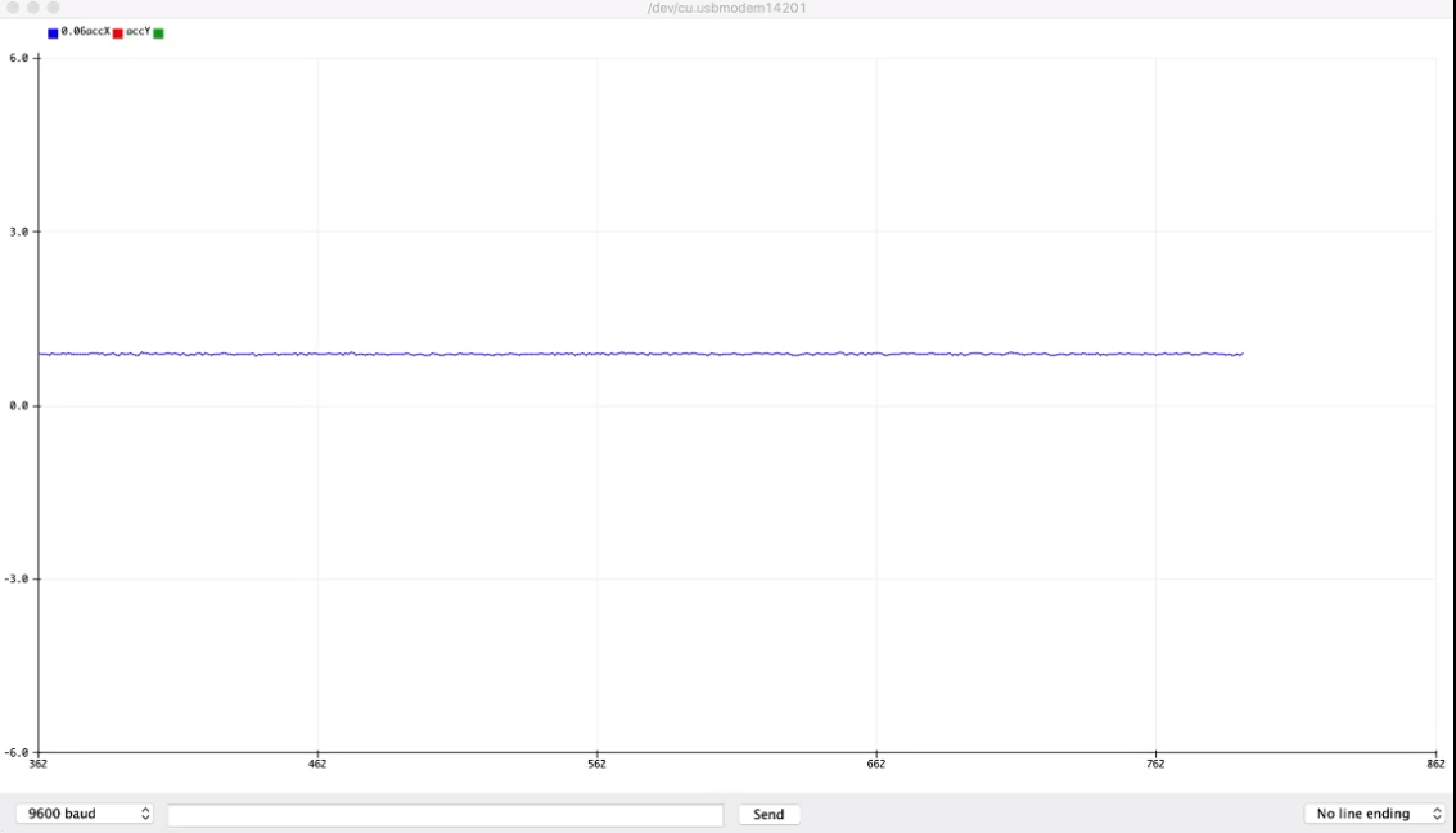
}

**-------------------------------------------------------------------------------------------------------------------------------**

This new addition to our code is quite simple. The only thing that we changed is that we defined the speaker pin and then set up the tone() function in our void loop using the function that we discussed above. The code functions as intended, while at rest the speaker produces a constant tone output of 500Hz and fluctuates when we accelerate the system. The output can be heard in the final output video in the next section. Something else to note is the fact that we scaled the output of our X acceleration by a factor of 4. Given that this is the only output that we are actually looking at scaling it alone won’t result in any stretching. Instead, it simply allows us to visualize the fluctuations in acceleration values easier.



Initially the addition of the speaker to our Accelerometer circuit caused some additional noise to be introduced as can be seen above. The plot above shows the accelerometer at rest. After some trial and error I realized that this noise came from the positioning of the circuit directly above the accelerometer. By removing the accelerometer from the box we see the resting accelerometer’s output revert back to:



As we can see there is a significant reduction in noise! Now that we have made these final changes to our sonification setup, all that is left to do is to put it all together and measure the net boat acceleration.

## **Results and Discussion:**

Using the code we developed above, we now look to utilize our accelerometer and sonifier to produce realtime outputs and insights whilst erging on slides. We mount our accelerometer and arduino directly to the base of the erg. Since the erg is one continuous piece, the acceleration will be the same regardless of the location we place the accelerometer. Hence, we choose the rear right leg out of convenience. All that is left to do is to upload the code to the Arduino and start rowing! The following video contains rowing footage with the real time arduino acceleration data overlaid. Additionally, you can hear the sonification of the data. (If the inserted drawing video does not work the link should function regardless).

<https://drive.google.com/file/d/1czf8II-eHJpHEFu-xsEnMAHbjy918zFr/view?usp=sharing>



As we can see, the device functions as intended! We are able to simultaneously visualize the real time acceleration outputs and hear the outputs sonified. Hence, we have fulfilled the primary goals of this project outlined in the initial background section. To illustrate some of the technical functionalities of this device lets look at a particular acceleration curve in order to illustrate how our data can help generate actionable insights. In particular we will look at a technical flaw known as the drive hump. The drive hump is characterized by a disconnect within the drive phase that usually comes as a result of timing disparities between individuals in the same boat. We can simulate a drive hump by artificially splitting our drive into two distinct phases, legs and back. This results in the following acceleration profile:



The profile above was taken from two rowers on the Bates Men’s rowing team. The drive hump is quite distinct in this particular profile and can be heard clearly on our speaker output. Interestingly, the drive hump is a phenomenon that for the most part goes unseen when we look at individual force curves. This makes sense as it is a byproduct of timing disparities and not necessarily a reflection of individual outputs. Hence, our device is able to provide a unique insight into the efficiency of a pair or more of rowers. Let's now look at how the device can be used to improve and get rid of the drive hump. There are two ways in which our device accomplishes this. Frist, as we saw above, by providing a visual output athletes are able to see how their respective changes reduce and increase the magnitude of their drive hump. Athletes are able to use this in conjunction with the realtime sonification to smooth things out. When a drive hump is present you can hear three distinct regions of the drive phase when we would like to only hear two. Hence, by listening to each stroke in real time athletes can smooth out their acceleration profile. The following shows the result of two athletes listening to their acceleration profile and trying to smooth things out:



As we can see in the acceleration profile above the drive hump is considerably reduced. In fact, the dips that we do see may be a byproduct of noise within the accelerometer's sensor reading as opposed to this particular rowers technical flaws. This is a massive improvement from the previous acceleration profile and is particularly exciting as it demonstrates the potential of our device as a viable training tool.

## **Future Steps:**

There are a number of future steps that can be made to increase the scope and functionality of our new device. First, it would be interesting to explore the possibility of real time data smoothing. As it stands our output still contains a large number of fluctuations that is inherent with the high rate at which we are gathering data. This data smoothing could potentially tell a more complete story about what is going on from stroke to stroke. Certainly, there is a high potential for over-smoothing the data and therefore losing access to the insights that the net acceleration curve actually provides. As such, this would be a more involved process and would require extensive tweaking to produce an output that is still accurate to the data. Another functionality that I considered implementing was to reset the graphing program after each stroke period. This would allow you to just visualize the net acceleration curve for the previous stroke. This problem was remarkably nuanced and despite multiple attempts within the Arduino and even using Python to plot the data I was unable to produce an output better than what was already housed in the built in Serial Plotter function. The solution to this problem certainly lies within exporting the real time data to Python. Upon successfully establishing a connection between the Arduino and Python this problem would likely be rather trivial. The last future step that we could take would be to output realtime stroke rate data overlaid onto our graph. This is not a super useful addition as you have access to realtime stroke rate data from the erg screen and also by watch. Due to the consistent shape of the acceleration curve we could quite easily produce an if statement that measures the time between two catches and outputs a corresponding stroke rate output. While all of these future steps would be worthy additions, the device that we have produced is capable of producing all of the insights we intended for it to produce in its current state.

## **Work Cited:**

Accelerometer setup, schematic and tutorial: <https://www.mschoeffler.de/2017/10/05/tutorial-how-to-use-the-gy-521-module-mpu-6050-breakout-board-with-the-arduino-uno/>