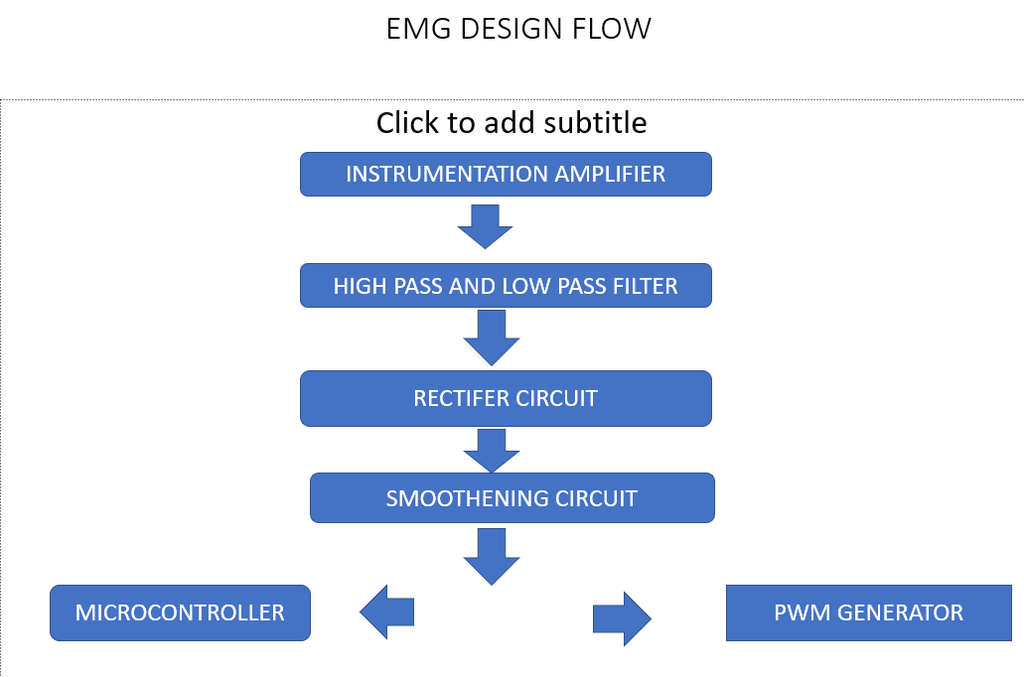
Whenever our body needs to make a movement, the nervous system sends out tiny electrical signals that control our muscles. The technique of [electromyography](https://en.wikipedia.org/wiki/Electromyography) (EMG) allows us to amplify and measure these electrical signals. In addition to being a useful clinical tool for diagnosing various neurological disorders, EMG recordings can be used for projects such as [controlling prosthetic devices](https://www.youtube.com/watch?v=MLvwTlbj1Y8&feature=youtu.be&t=440) or [playing music](https://www.instructables.com/id/Make-Muscle-MIDI-Music/).

An EMG, or electromyogram, is a measurement of the electricity produced by the movement in muscle tissue. Three electrodes will be used as sensors to provide input voltage to the circuit, and the output voltage will provide a reading of muscle activity. The stages of the circuit are as follows: an instrumentation amplifier, a band pass filter, and a non-inverting amplifier. The instrumentation amplifier provides high input impedance to match the high output impedance of skin. The band pass filter removes frequency content out of the bandwidth of the EMG. Finally, the non-inverting amplifier provides enough gain to make the small EMG signal large enough to be usable.

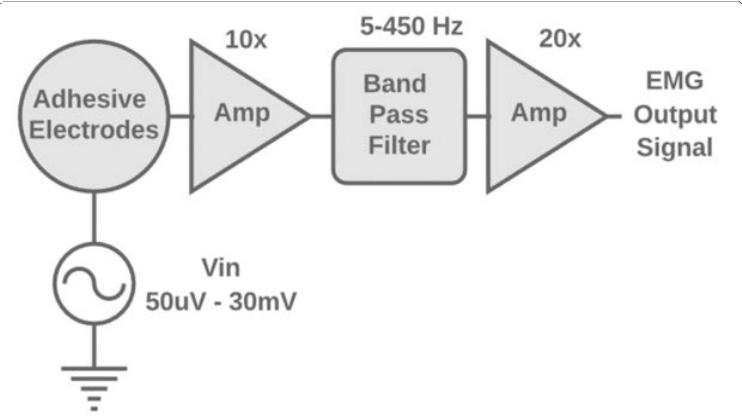
Here we are building an EMG amplifier that allows us to translate the tiny electrical signals produced by motor-neurons into mechanical action using servo to control the prosthetic arm.



EMG signals are measured differentially, meaning the signal we amplify and analyze is actually the difference in electrical potential between two points on the muscle.

Our main goal with this EMG amplifier is to amplify the signal we're interested in (muscle activity) without increasing the noise (often from external electrical interference). A differential amplifier operates on the assumption that any noise signal interfering with our EMG recording will uniformly affect the local region of muscle we're recording from. In other words, a large source of interference across the room will introduce the same amount of electrical noise in our recordings whether we record from a muscle in the middle of your forearm or a few centimeters away from that same position.

Making use of this assumption, we're able to achieve our goal of maximizing the [signal to noise ratio](https://en.wikipedia.org/wiki/Signal-to-noise_ratio) by amplifying the difference between two recording sites close together on a muscle. The components of the signal that are common between the recording sites (the electrical noise) will be removed when we take the difference (i.e. subtraction) between the two recordings. The ability of an amplifier to reject the common signal is referred to as the [common-mode rejection ratio](https://en.wikipedia.org/wiki/Common-mode_rejection_ratio) (CMRR). An ideal differential amplifier would have an infinite CMRR, rejecting all the noise that is the same at either input, but when it comes to practical the CMRR is defined so it removes some part of noise.

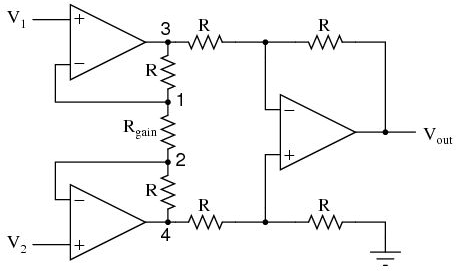


Basic over view of EMG sensor circuit blocks

# Step 1: Instrumentation Amplifier

The instrumentation amplifier is a circuit with high input impedance which amplifies the difference between two input signals. Because the skin has high output impedance, its voltage signal can only be measured by a circuit that has high input impedance. This phenomenon can be easily understood by thinking of the skin impedance and circuit impedance as resistors in a voltage divider. If both resistors are of equal value, only half of the input voltage will be measured across the circuit impedance. As the circuit impedance is increased above the skin impedance, more voltage will be applied across the circuit. We want to maximize the voltage going into the circuit.

Furthermore, the EMG is the difference of the voltage signals at the ends of a muscle, so a differential amplifier is required for the first stage.



Differential amplifier

# Specifications of a good instrumentation amplifier

**Accurate and stable gain:**

As the device amplifies signals of the very low level, thus its basic need is its gain must be finite and accurate. Usually, gain lies in the range of 1 to 100.

**Easy gain adjustment:**

When we are talking about the gain. Then it is important that it varies properly inside the specified limit. Usually, the gain is adjusted using a potentiometer or by making use of switches like JFET and MOSFET.

**High CMMR:**

An infinite CMMR is the most preferred range in case of the instrumentation amplifier. As the output of transducer has large common mode noise signals during its long-distance transmission.

An instrumentation amplifier must completely eliminate the common mode noise components in order to amplify the difference of input only.

**High input impedance:**

It is preferred to have an almost infinite value of input impedance in order to avoid the loading effect at the input.

**Low output impedance:**

The low value of impedance at the output must be exhibited by the instrumentation amplifier. In ideal cases, it is assumed to be approximately 0 to avoid loading.

**Low time and temperature drift:**

To have the desired output, it is always recommended that various characteristics and elements of the device must not change with variation in time or temperature.

**Low power consumption:**

For any device, it is always recommended that it must be power efficient. So, an instrumentation amplifier must also consume less power.

**Differential input:**

To have the desired amplification, the device must amplify only the difference of the input signal.

**High slew rate:**

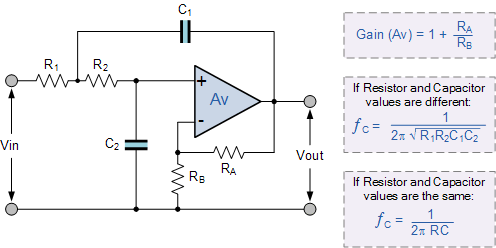
Slew rate provides us with the idea about the change in output voltage with any change in the applied input. So, for an instrumentation amplifier, slew rate must be high.

# Reason to Instrumentation Amplifier

* The gain of a three op-amp instrumentation [amplifier circuit](https://www.elprocus.com/stereo-amplifier-circuit-using-tda2822/) can be easily varied by adjusting the value of only one resistor Rgain.
* The gain of the amplifier depends only on the external resistors used.
* The input impedance is very high due to the emitter follower configurations of amplifiers 1 and 2
* The output impedance of the instrumentation amplifier is very low due to the difference amplifier3.
* The CMRR of the op-amp 3 is very high and almost all of the common mode signal will be rejected.

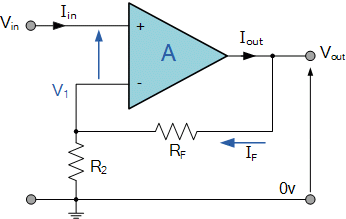
# Step 2: Band Pass Filter (Low pass and then high pass)

The majority of the EMG signal is between 5-450Hz, so we chose our cutoff frequencies for this range. Our band pass filter was a Salley-Key low pass filter followed by a Sallen-Key high pass filter.



# Step 3: Non-inverting Amplifier

The final stage is meant to increase the output signal to be read. Depending on how we intend to use our EMG signal, what muscle groups we intend to measure from, and the quality of our electrodes, we will require a different output voltage, and therefore, a different gain. Using this amplifier we will be increasing the amplitude of the EMG signal

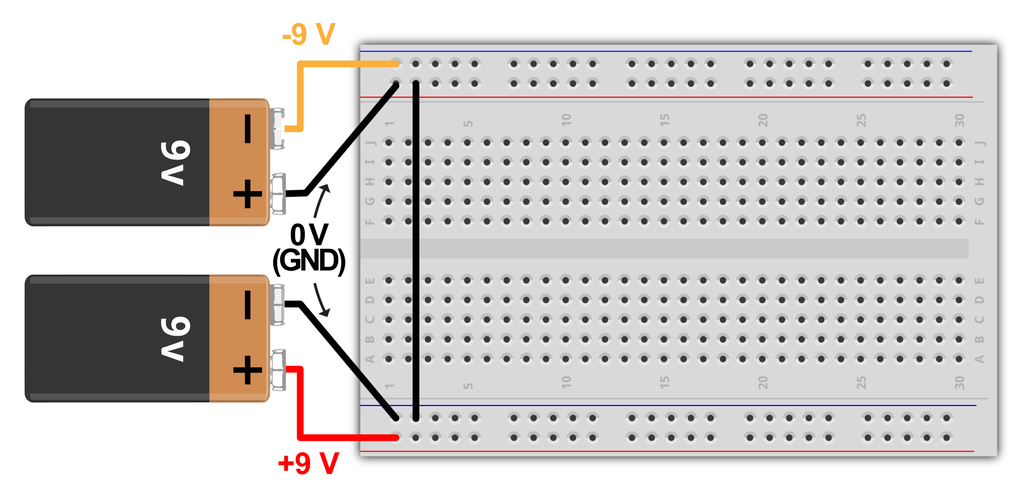


# Implementation

# Power Supply

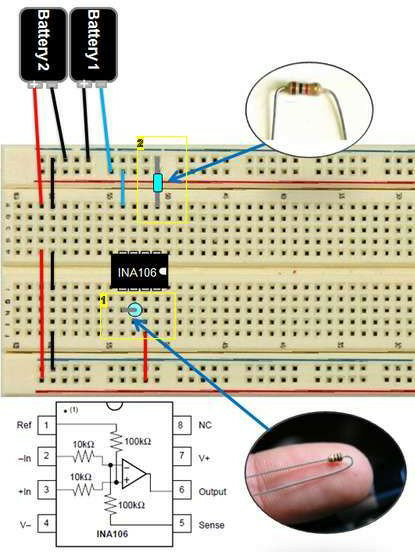
the electrical activity generated by muscles can either be positive or negative with respect to zero volts. The amplifier must therefore have both a negative and positive supply in order to accommodate and amplify these bipolar muscle signals. Without a negative supply, the amplifier would be unable to produce any outputs less than zero volts.

Two 9 volt batteries are both a convenient and safe way to power our EMG amplifier. Because we'll be attaching ourselves to the circuit, we don't want our circuit connected in any way to AC power from the wall. The diagram below illustrates how to connect the batteries in a ±9 volt arrangement.



# SIGNAL ACQUISITION

Next, we will work on the signal acquisition phase of your EMG circuit which we will use to measure our body’s nervous system’s electrical impulses used to activate muscle fibers.



# SIGNAL CONDITIONING - Amplification

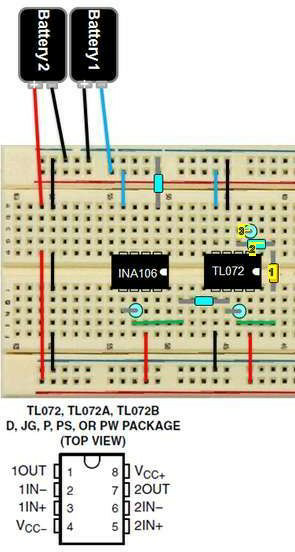
In this phase, we’re going to take those very small signals measured in the SIGNAL ACQUISITION phase and amplify them.

First we will be inverting amplifier with a gain of -15. An inverting amplifier does exactly what it sounds like. It amplifies your signal but also inverts it.

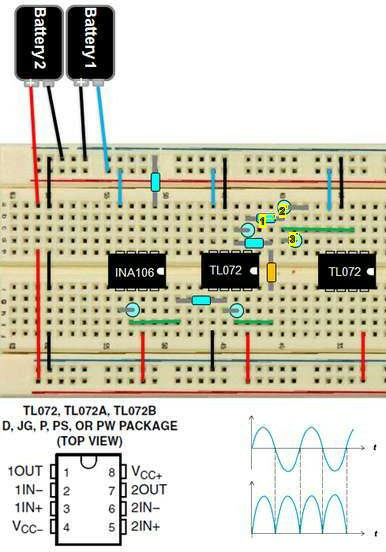
We can calculate the gain by G=-R2/R1 or in this case G=-150 kOhm / 10 kOhm. (See image 1)

Next, we are going to add a capacitor to AC couple the signal. AC coupling is useful in removing DC error offset in a signal.

Next we are going to add another inverting amplifier with a gain



# SIGNAL CONDITIONING - Rectification

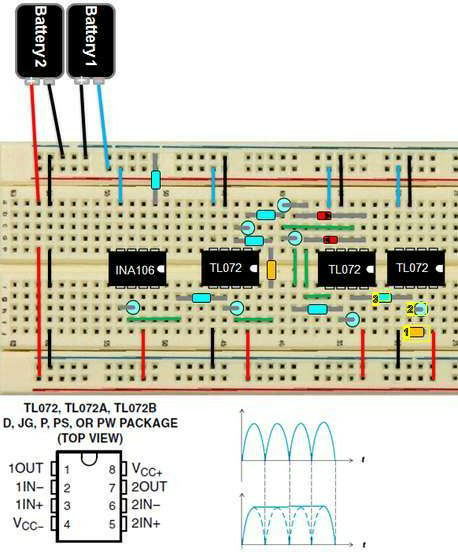
In this phase, we will be rectifying the signal using an active [full-wave rectifier](http://en.wikipedia.org/wiki/Rectifier#Full-wave_rectification). Our rectifier will take the negative portion of our signal and turn it positive so the entire signal falls within the positive voltage region. We will use this coupled with a low pass filter to turn our AC signal in to a DC voltage; readying the signal to be passed to a microcontroller.

# SIGNAL CONDITIONING - Smoothing + Amplification

In this phase of circuit assembly, we will be using an active low-pass filter to filter out the humps of our signal to produce a smooth signal for our microcontroller.

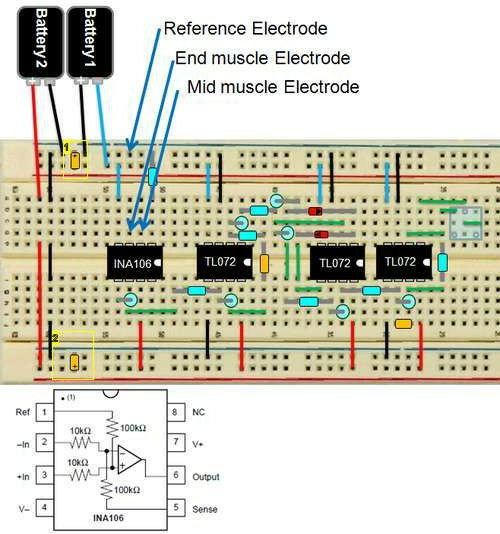
However, since this is an active filter, there is a side effect of inverting the signal. We will need to invert the signal one more time (and have the ability to amplify it more if desired) using another inverting amplifier circuit with a trimmer configured as a variable resistor.

By using a screw driver and turning the trimmer, you will be able to adjust the gain of your signal to account for different signal strengths from different muscle groups.



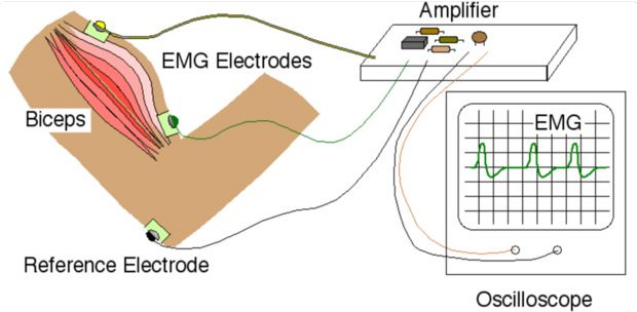
# Connecting Electrode Cables

Connect the reference electrode to the GND rail of the circuit. Connect the mid muscle electrode to chip A's pin 2 connect the end electrode to chip A's pin 3

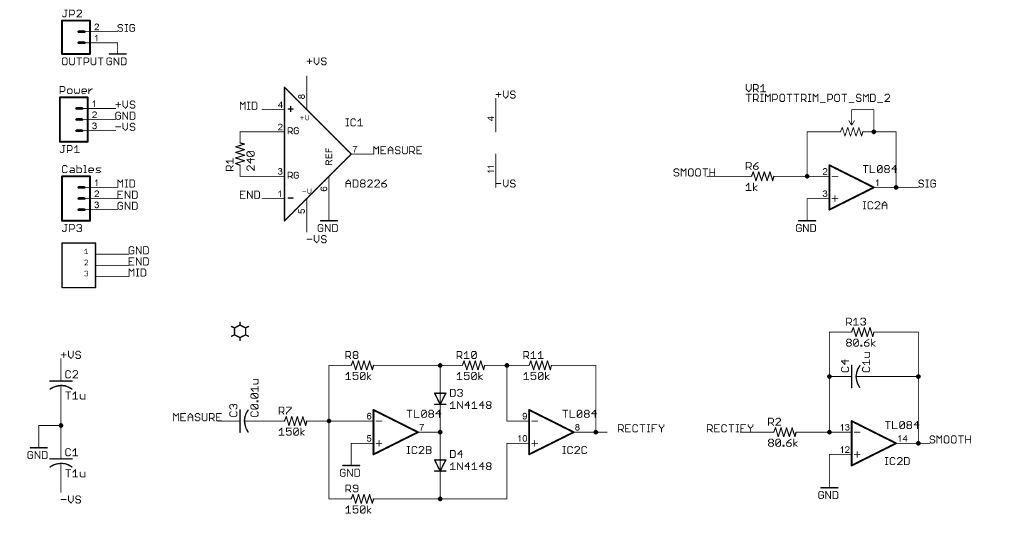


# Connecting the Electrodes

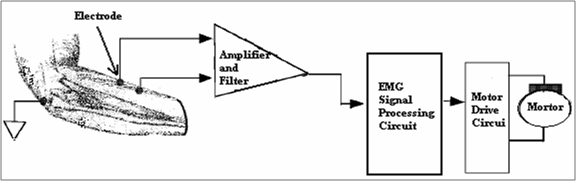
The EMG circuit requires three electrodes: positive input, negative input, and ground. The placement of the electrodes will vary based on the muscle that you intend to measure. For the bicep, the elbow is a suitable placement for ground. The positive and negative electrodes should be placed on the upper arm as shown in the figure. In my experience, the signal was stronger when electrodes were placed closer to the center of the body (medially) when the palm is facing upward.



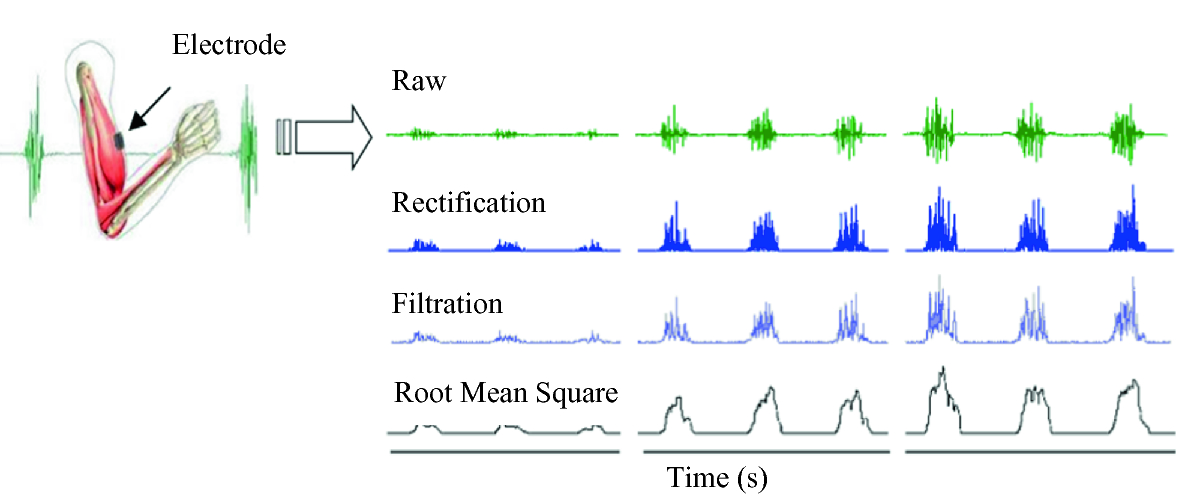
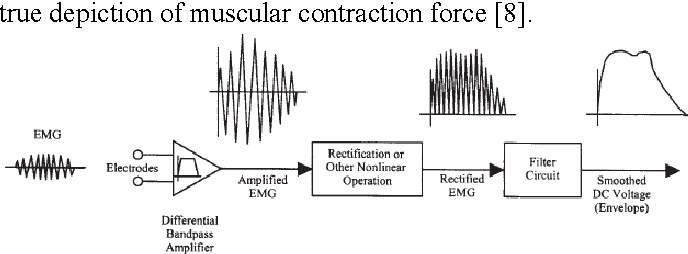
Circuit Diagram



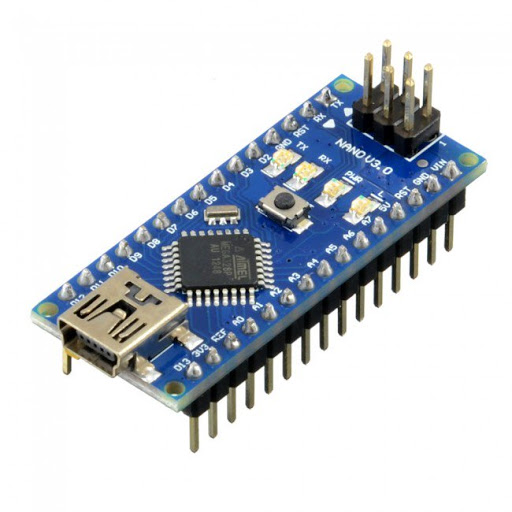
Project Over view:

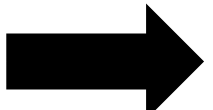
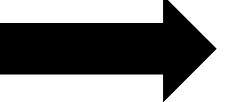
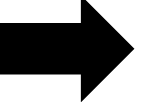
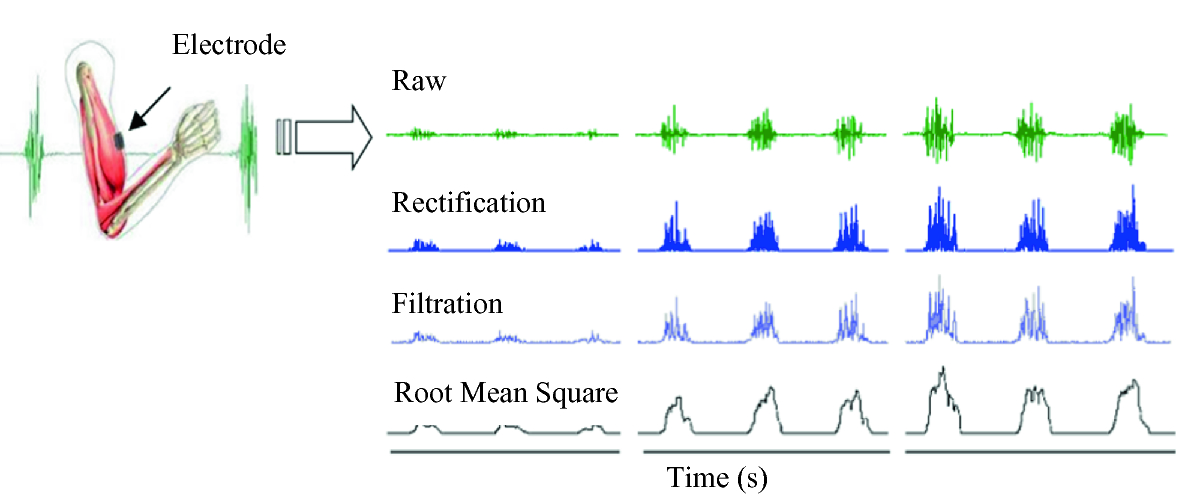


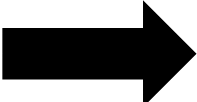
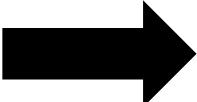
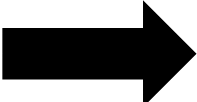
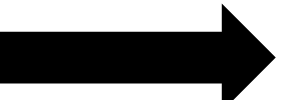
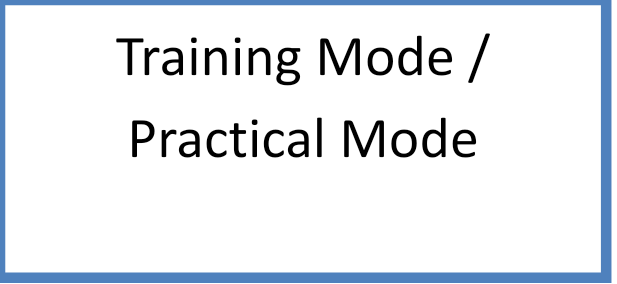
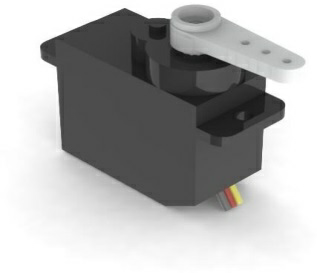
**Signal processing:**

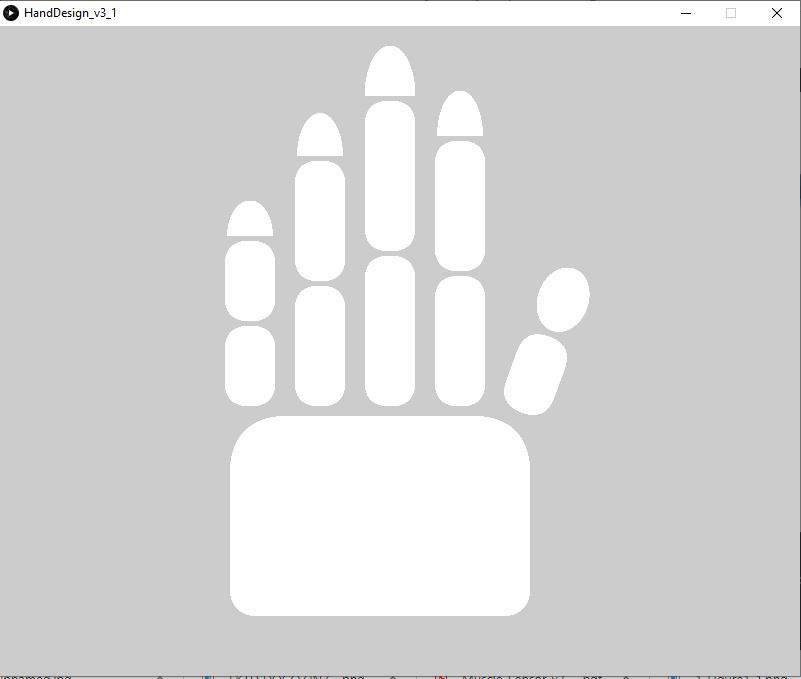
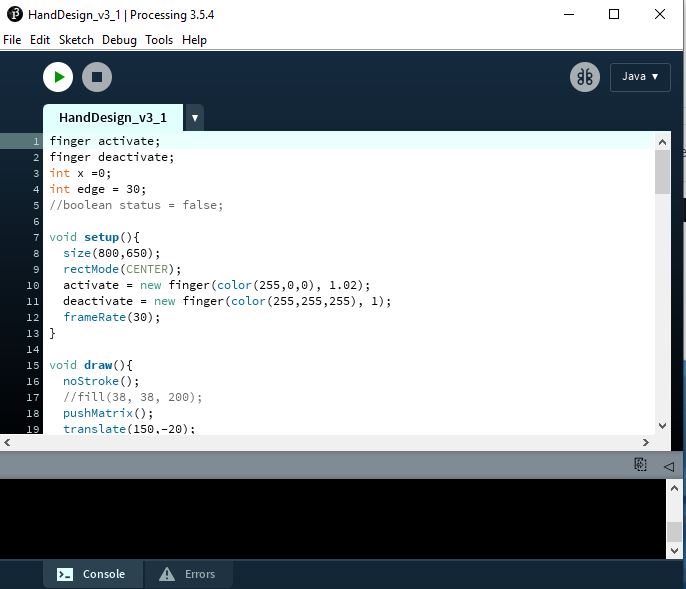


**Connection Diagram:**









Here Write details about Processing IDE

# Arduino EMG sensor reading code:

int lastsensorState = HIGH; // the previous reading from the input pin

unsigned long lastDebounceTime = 0; // the last time the output pin was toggled

unsigned long debounceDelay = 30; // the debounce time; increase if the output flickers

bool sensorstate = HIGH; //saving state of the switch

byte tapCounter; //for saving no. of times the switch is pressed

int timediff; //for saving the time in between each press and release of the switch

bool flag1, flag2; //just two variables

int analogpin = 0;

int val = 0;

int ledPin = 13;

//calibration

int temp=0;

int sens\_max = 0;

int sens\_min = 1023;

int threshold = 0;

bool state = HIGH;

//fist release should be last state

int prev\_state = 0;

int hand\_state = 0;

long double presstime, releasetime; //for saving millis at press and millis at release

void setup() {

Serial.begin(115200); //for serial monitor

pinMode(ledPin, OUTPUT);

digitalWrite(ledPin, HIGH);

hand\_calibration();

digitalWrite(ledPin, LOW);

threshold = (sens\_max - sens\_min) \* 0.25;

// digitalWrite(red, LOW);

// delay(1000);

// digitalWrite(red, HIGH);

// delay(1000);

// digitalWrite(green, LOW);

// delay(1000);

// digitalWrite(green, HIGH);

// delay(1000);

// digitalWrite(blue, LOW);

// delay(1000);

// digitalWrite(blue, HIGH);

}

void loop() {

val = analogRead(analogpin); //muscle sensor connected to pin A0 being stated as val

//Serial.println(val);

if( val < threshold) //if you flex and the sensor value is greater than 550 then close servos--adjust this value to your muscle sensor value

{

digitalWrite(ledPin, LOW);

delay(20);

//Serial.println("LOW");

state = HIGH;

}

else if(val > threshold)

{

digitalWrite(ledPin, HIGH);

delay(20);

//Serial.println("HIGH");

state = LOW;

}

int reading = state;

if (reading != lastsensorState) {

// reset the debouncing timer

lastDebounceTime = millis();

}

if ((millis() - lastDebounceTime) > debounceDelay) {

// whatever the reading is at, it's been there for longer than the debounce

// delay, so take it as the actual current state:

// if the button state has changed:

if (reading != sensorstate) {

sensorstate = reading;

}

}

//Serial.println(sensorstate);

//when switch is pressed

if (sensorstate == 0 && flag2 == 0)

{

presstime = millis(); //time from millis fn will save to presstime variable

flag1 = 0;

flag2 = 1;

tapCounter++; //tap counter will increase by 1

//delay(10); //for avoiding debouncing of the switch

}

//when sw is released

if (sensorstate == 1 && flag1 == 0)

{

releasetime = millis(); //time from millis fn will be saved to releasetime var

flag1 = 1;

flag2 = 0;

timediff = releasetime - presstime; //here we find the time gap between press and release and stored to timediff var

//Serial.println(timediff);

//delay(10);

}

if ((millis() - presstime) > 400 && sensorstate == 1) //wait for some time and if sw is in release position

{

if (tapCounter == 1) //if tap counter is 1

{

if (timediff >= 500) //if time diff is larger than 400 then its a hold

{

//Serial.println("Fist release");

hold(); //fn to call when the button is hold

}

else //if timediff is less than 400 then its a single tap

{

//Serial.println("fist full close");

singleTap(); //fn to call when the button is single taped

}

}

else if (tapCounter == 2 ) //if tapcounter is 2

{

if (timediff >= 500) // if timediff is greater than 400 then its single tap and hold

{

//Serial.println("Fist half close");

tapAndHold(); //fn to call when the button is single tap and hold

}

else // if timediff is less than 400 then its just double tap

{

//Serial.println("point");

doubleTap(); //fn to call when doubletap

}

}

else if (tapCounter == 3) //if tapcounter is 3 //then its triple tap

{

//Serial.println("pinch");

tripleTap(); //fn to call when triple tap

}

else if (tapCounter == 4) //if tapcounter is 4 then its 4 tap

{

//Serial.println("fingers close and open");

fourTap();//fn to call when four tap

}

tapCounter = 0;

}

lastsensorState = reading;

//Serial.print("status = ");

//Serial.println(state);

//delay(100);

Serial.print(hand\_state);

Serial.print("/");

// Serial.print(prev\_state);

// Serial.print("/");

Serial.println(val);

}

void nolight()

{

//digitalWrite(red, HIGH);

//digitalWrite(green, HIGH);

//digitalWrite(blue, HIGH);

//Serial.println("No light");

}

void singleTap()

{

nolight();

if(prev\_state == 0)

{

hand\_state = 1;

prev\_state = hand\_state;

}

else

{

hand\_state = prev\_state;

}

return hand\_state;

return prev\_state;

}

void doubleTap()

{

nolight();

if(prev\_state == 0)

{

hand\_state = 3;

prev\_state = hand\_state;

}

else

{

hand\_state = prev\_state;

}

return hand\_state;

return prev\_state;

}

void tripleTap()

{

nolight();

if(prev\_state == 0)

{

hand\_state = 4;

prev\_state = hand\_state;

}

else

{

hand\_state = prev\_state;

}

return hand\_state;

return prev\_state;

}

void fourTap()

{

nolight();

if(prev\_state == 0)

{

hand\_state = 5;

prev\_state = hand\_state;

}

else

{

hand\_state = prev\_state;

}

return hand\_state;

return prev\_state;

}

void hold()

{

nolight();

hand\_state = 0;

prev\_state = 0;

return hand\_state;

return prev\_state;

}

void tapAndHold()

{

nolight();

if(prev\_state == 0)

{

hand\_state = 2;

prev\_state = 2;

}

else

{

hand\_state = prev\_state;

}

return hand\_state;

return prev\_state;

}

void hand\_calibration()

{

//Serial.println("Calibrating");

while(millis() < 5000)

{

temp = analogRead(analogpin);

if(temp < sens\_min)

{

sens\_min = temp;

}

if(temp > sens\_max)

{

sens\_max = temp;

}

}

}

# Hand virtual training mode processing Code:

import processing.serial.\*;

import java.awt.event.KeyEvent;

import java.io.IOException;

Serial myPort;

String data="";

//float roll, pitch;

finger activate;

finger deactivate;

int x =0;

float val = 0;

float state = 0;

void setup() {

size(800,600);

rectMode(CENTER);

activate = new finger(color(255,0,0), 1.02);

deactivate = new finger(color(255,255,255), 1);

frameRate(30);

myPort = new Serial(this, "COM17", 115200); // starts the serial communication

myPort.bufferUntil('\n');

}

void draw() {

noStroke();

pushMatrix();

translate(150,-20);

textSize(22);

text("state: " + int(state) + " Pitch: " + int(val), 240, 300);

if (state == 0){

deactivate.pinky();

deactivate.ring();

deactivate.middle();

deactivate.index();

deactivate.thumb();

}

else if (state == 1){

activate.pinky();

activate.ring();

activate.middle();

activate.index();

activate.thumb();

}

else if (state == 2){

deactivate.pinky();

activate.ring();

deactivate.middle();

activate.index();

deactivate.thumb();

}

else if (state == 3){

deactivate.pinky();

deactivate.ring();

deactivate.middle();

activate.index();

deactivate.thumb();

}

else if (state == 4){

deactivate.pinky();

deactivate.ring();

deactivate.middle();

activate.index();

activate.thumb();

}

else if (state == 5){

deactivate.pinky();

deactivate.ring();

activate.middle();

deactivate.index();

deactivate.thumb();

}

fill(255);

arc(240,width/2+20,330,350, radians(0), radians(180), CHORD); // Palm

popMatrix();

}

// Read data from the Serial Port

void serialEvent (Serial myPort) {

// reads the data from the Serial Port up to the character '.' and puts it into the String variable "data".

data = myPort.readStringUntil('\n');

// if you got any bytes other than the linefeed:

if (data != null) {

data = trim(data);

// split the string at "/"

String items[] = split(data, '/');

if (items.length > 1) {

state = float(items[0]);

val = float(items[1]);

}

}

}

class finger {

color c;

float zoom;

finger(color tempC, float tempzoom) {

c = tempC;

zoom = tempzoom;

}

void pinky()

{

pushMatrix();

//stroke(1);

fill(c);

//scale(zoom);

rect(100,width/2-40,50,80);

rect(100,width/2-125,50,80);

arc(100,width/2-170,50,100, radians(180), radians(360), CHORD);

popMatrix();

}

void ring()

{

fill(c);

rect(170,width/2-60,50,120);

rect(170,width/2-185,50,120);

arc(170,width/2-250,50,100, radians(180), radians(360), CHORD);

}

void middle()

{

fill(c);

rect(240,width/2-75,50,150);

rect(240,width/2-230,50,150);

arc(240,width/2-310,50,100, radians(180), radians(360), CHORD);

}

void index()

{

fill(c);

rect(310,width/2-65,50,130);

rect(310,width/2-200,50,130);

arc(310,width/2-270,50,100, radians(180), radians(360), CHORD);

}

void thumb()

{

fill(c);

pushMatrix();

translate(380,width/2-60);

rotate(radians(20));

rect(15,-5,50,120);

popMatrix();

pushMatrix();

translate(420,width/2-125);

rotate(radians(20));

arc(0,0,50,100, radians(180), radians(360), CHORD);

popMatrix();

}

}