Myoelectric Bionic Arm

School of Engineering and Applied Sciences
Harvard University

Cambridge, MA

Description: A myoelectric bionic arm that uses EMG signals from the forearm to control a 3D-printed prosthetic hand.

By: Arav Bhargava, Finn Seyffer, Anthony Bynum, Xavier Ayala-Vermont

Class: Engineering Sciences 50
Instructor: Gu-Yeon Wei, Vijay Janapa Reddi

May 2025

ABSTRACT

Our project aims to demonstrate the effectiveness of circuit-based EMG signal preprocessing to enhance the accuracy of myoelectric prosthetic control. By integrating analog filtering, amplification, and machine learning classification on an Arduino, we created a cost-effective, real-time trainable bionic hand. This approach addresses common challenges in myoelectric prosthetics, such as weak EMG signals and lengthy training times, and has a future use to bring advanced prosthetic technology within reach of more users.

INTRODUCTION

In the past few years, prosthetic control has transformed completely, moving from body-powered, harness-controlled hooks to electrode-controlled neuroprosthetics. Currently, the most common high-tech prosthetic is a myoelectric prosthetic, which is used for both transradial and transhumeral prosthetic devices. These devices are often over $100,000 and require extensive fitting and training processes, often making this process long and impractical.

The ideal, "holy grail" of upper limb prosthetics would be a bionic that does not require custom-fit, is low-cost, has high-accuracy bionic control that is easy and quick to train, and has a long battery life. One of our team members, Arav Bhargava, has developed and patented a size-adjustable, high-quality socket that costs less than $40. To improve on the prosthetic control element, our goal was to create a low-cost, real-time, trainable electrode-controlled bionic arm that could be fit to an amputee in less than a minute.

Myoelectric prosthetics typically function with the following processes. Electrical signals are picked up by electrodes on the arm, processed through a microcontroller, classified through a machine learning algorithm, and then the inputs are sent to the motors for fully functional hand control.

DESIGN

Originally, our intent was to amplify and filter three independent channels targeting three muscles. By flexing the arm in the same way one would to perform the coded gesture, a classifier would output the intended gesture. This was because our initial understanding was that the signal from each electrode could be interpreted individually, which is conveyed in Figure 1, our concept drawing.

After further research, we understood that there is a large amount of noise across many different frequencies for a single electrode signal. To accurately detect the peak of a muscle activation voltage (which we read to be 0- 10 mV AC, 20- 500Hz (SFU department of biomedical physiology and kinesiology)), common practice is to use a differential amplifier to eliminate the noise shared by two electrodes on the same muscle, isolating the activation. This is frequently done with an instrumentation amplifier, however, we had only low-power, high impedance op amps to use in their place.

We considered several options for the circuit, specifically with the order of the differential amplifier, amplification, and filtering. Initially, when the channels were separate, we used a first amp stage gain of 100 before passing the current through a high pass and low pass passive filter in series. We used values of 820nF and 9.84kohm to isolate frequencies above 20Hz, and 330nF and 1kohm to attenuate signals above 500 Hz. Then, the signals were once again raised by a gain of 10 before being fed into a full wave bridge rectifier to yield only positive voltage outputs. With a positive 0-3V output signal estimated, we believed we could pass three outputs to the analog pins of an Arduino mkr0 for use in classification. See figures 3, 4. We later rectified (pun intended) this error.

We faced too much noise introduction with the use of the rectifier, likely due to the added

complexity and low component tolerances. Additionally, we suspected that we were re-introducing and then amplifying noise outside of our intended range after the filtering in the circuit, resulting in an unusable signal. To fix this, we re-build with a stage of pre-amplification for two active input electrodes (gain of 30), a differential amplifier to eliminate shared noise (gain 1), then the second amplification stage (gain 10), and finally the passive filtering at the very end (unchanged). Common ground was attached to a reference electrode on a distal bone from the muscle being measured, allowing us to actually read an analog voltage from the arduino as intended.

We estimated a gain of 300, meaning 0-3V input to our arduino. In reality, we experienced a great deal of losses and interference, resulting in hundreds of millivolts at the maximum. These losses were again likely derived from poor tolerancing on components, in addition to the parasitic losses from using a breadboard for the rebuild of our circuit (see figure 5). In the future, a re-build would involve higher tolerance components, the use of an instrumentation amplifier as a differential amplifier as the first stage, and the combination of our filters with gain (active filtering) to reduce the overall number of stages.

The output of the circuit was fed to the Arduino mkr0 which used analog values to classify what gesture was being flexed. With the reduction from three active channels with three wires to one active channel with three wires, the scope of our project was reduced from a multi-gestural classifier to binary, open/close sensing. To control all five servos actuating the fingers, we made use of an Adafruit PCA9685 pwm breakout board which communicates via 12C on the SCL and SDA pins of the arduino. Luckily, implementing more gestures in the future is as simple as increasing precision and using more instances of our final circuit.

For the structure of our bionic hand, we decided to use an open-source project from Will Cogley. We made minor modifications to the hand to accommodate our specific PTFE tuning and tensioning line and restrict some joints.

PARTS LIST

Product Name	Description	Vendor	Catalog #	Quantity	Unit Cost	Total Cost	Notes	URL
Arduino MKR Zero	Arduino	Arduino	1050-1137-ND	1	$38.00	$38.00		Link
Adafruit 2773	Muscle Sensor	Adafruit	2773	2	$4.95	$9.90		Link
Amazon nylon line	Nylon Line	Amazon		1	$11.59	$11.59		Link
Amazon PTFE tube	Teflon Tubing	Amazon		1	$7.59	$7.59	14-57 NanoFarad	Link
LMC6482AIN/NOPB	Digikey		LMC6482AIN/NOPB-ND	4	$4.90	$19.60		Link

6 Pack Servo Motors	Amazon			6	$4.50	$27.00		Link
Electrolytic capacitors				10	$0.00	$0.00	In lab	
Resistors				15	$0.00		In lab	N/A
DKS-SOLDERBREADBOX1	Breadboard	Digikey	DKS-SOLDERBREADBOX1-ND	2	$3.79	$7.58		Link
					Total:	$121.26		

PROJECT IMPLEMENTATION

The implementation of our myoelectric bionic arm began with the planning and sequential execution of both hardware and software components. First, we focused on EMG signal acquisition so that we could get reliable control by capturing clean muscle signals. We placed two surface electrodes along the forearm's flexor muscles, specifically targeting the muscle groups responsible for hand closure, while a reference electrode was placed on a neutral site near the elbow. This positioning was critical for maximizing the amplitude of the EMG spikes associated with the closing of the fist and minimizing noise from unrelated muscle activity or environmental sources. The changes we made between the early and final circuit designs are noted above.

This signal feeds directly into an Arduino MKR Zero, which has a binary classification algorithm that computes the standard deviation of the electrode signal when the user has an open hand and compares this to the standard deviation of the electrode signal when the user has a closed hand. This is real-time trainable, meaning that for the user, all they have to do is place electrodes, hold their hand open and press "return," hold their hand closed and press "return," and then they have a functional myoelectric prosthetic. We implemented this in the code that is shown below.

Simultaneously, we 3D-printed the prosthetic hand using a modified Epsilon 1 open-source model and fabricated custom boxes for wire management and housing the Arduino and circuitry. The mechanical assembly involved mounting five SG90 micro servos, one for each finger, and threading nylon tendon lines through PTFE tubing to ensure smooth, low-friction movement. We then connected the output of the analog circuit to the Arduino's analog input, where the digitized EMG signal underwent software-based preprocessing. This included full-wave rectification to convert the AC signal to a usable DC level, followed by a digital low-pass filter to further reduce noise.

Throughout the process, we encountered and addressed several challenges. We did not have an instrumental amplifier or press-on electrode studs, which both contributed to more signal noise. To fix this, we mitigated noise through careful electrode placement and shielding, while servo jitter was reduced by refining both the analog filtering and the classifier's

decision boundaries. Mechanical issues, such as tendon friction or servo misalignment, were solved through adjustments to the CAD and fishing line material.

After several rounds of oscilloscope testing, troubleshooting, and experimenting, our bionic arm was able to perform two distinct movements (open and close) in response to live EMG signals.

Attached in the appendix are some images representing the model and our design process.

The short video attached on canvas shows the video of our bionic arm with a description of our processes.

TEAM MANAGEMENT

- Arav Bhargava: Conceptual planning, circuit assembly, EMG signal amplification, Binary classification, implementation, data collection, testing.
- Finn Seyffer: Circuit design and assembly, EMG signal processing
- Xavier Ayala-Vermont: 3D modeling, printing, mechanical assembly.
- Anthony Bynum: Circuit design, troubleshooting, documentation.

OUTLOOK AND POSSIBLE IMPROVEMENTS

To improve this project further, we would consider adding more hand movements to mimic our own, in addition to having the bionic arm respond faster by increasing accuracy and lowering thresholds. These two improvements will come with eliminating noise from the system, using an instrumentation amplifier, and implementing a kNN algorithm, which is one of the best classifiers for myoelectric prosthetics.

ACKNOWLEDGEMENTS:

We would like to thank our TFs, professors, Champa, and Leo for their guidance and support throughout the project!

DISCLAIMER

It is acceptable to share our code and project with others.

REFERENCES:

Designing a New Bionic Hand (Which Prints Pre-Assembled)

Understanding EMG Resource

APPENDIX

Figure 1: Concept drawing

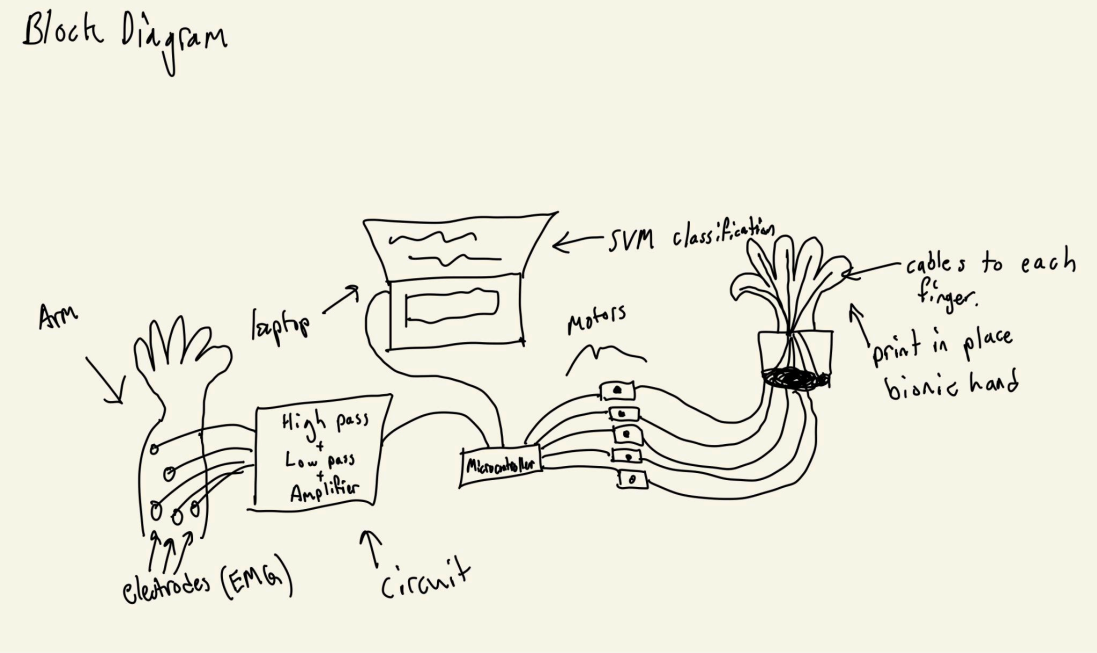

Figure 2: Initial circuit schematic

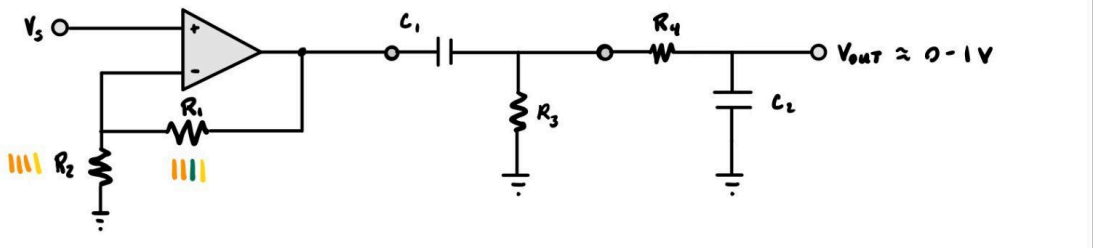

Figure 3: Testing the initial circuit design using oscilloscope probes

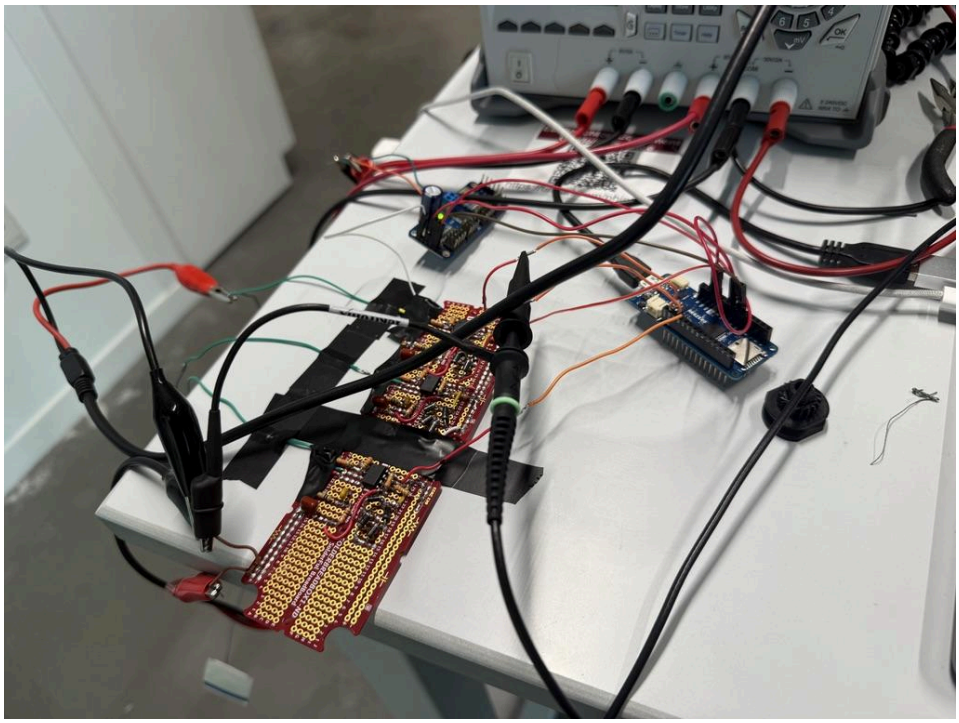

Figure 4: Initial design of dual amplifier and filtering implemented before the differential amplifier in the circuit

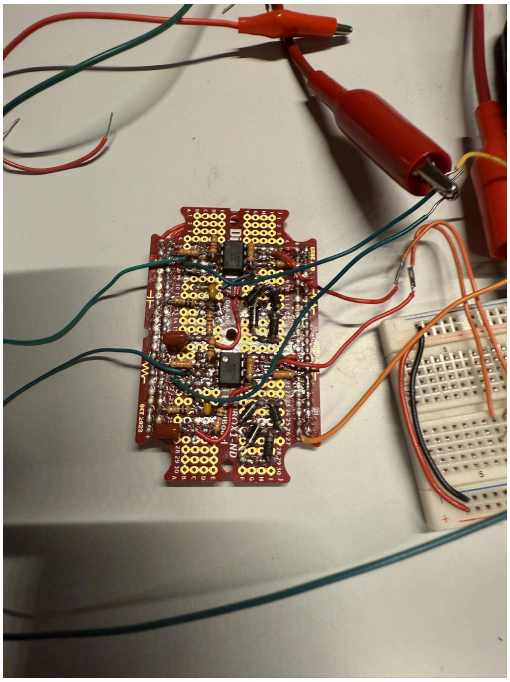

Figure 5: Assembled bionic hand

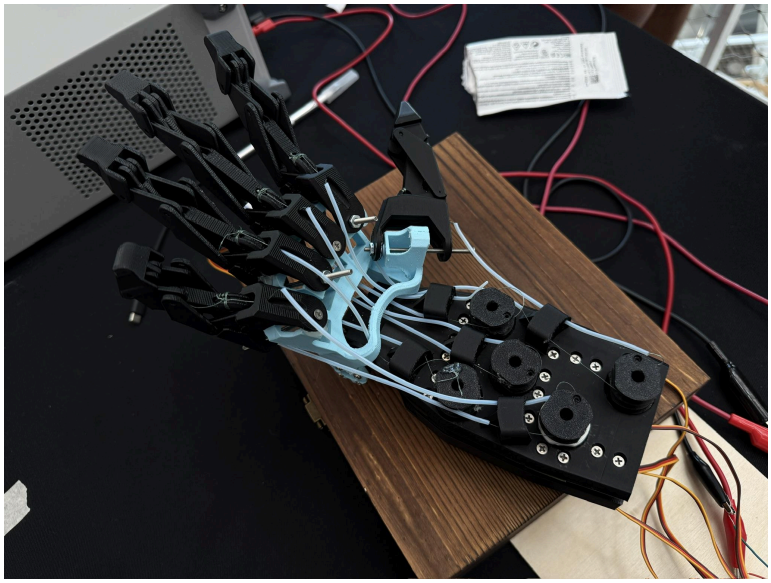

Figure 6: Full setup at demo

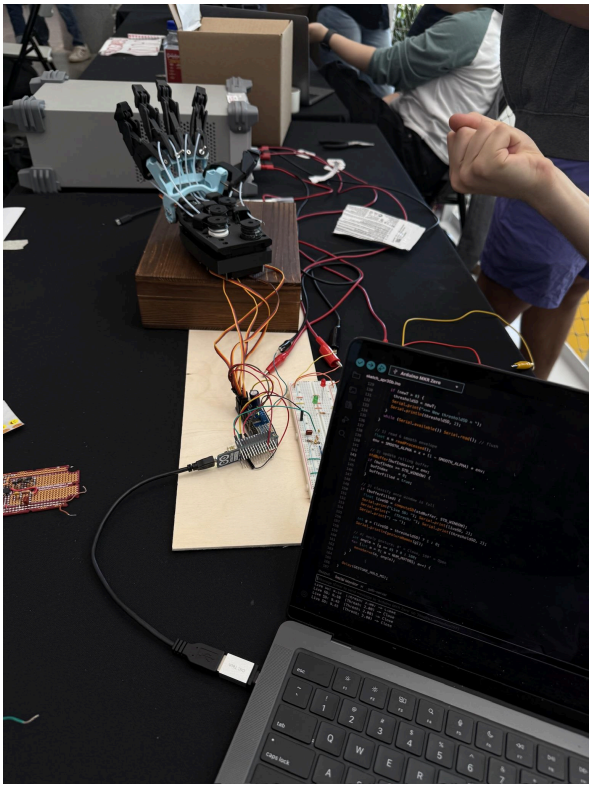

Figure 7: Full binary classification real-time trainable code

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// Only one EMG channel now: A0
#define NUM_CHANNELS     1
// Only two gestures: Open and Close
#define NUM_GESTURES     2
#define NUM_MOTORS       5

// Calibration params
#define BASELINE_SAMPLES 200    // samples for baseline
#define CAL_STD_SAMPLES  200    // samples per gesture to calibrate SD
#define STD_WINDOW       30     // rolling window size for live SD

const unsigned long GESTURE_HOLD_MS = 300;
const unsigned int  ADC_SETTLE_US    = 10;
const float SMOOTH_ALPHA            = 0.1;
```

```
// EMG pin
const int emgPin = A0;

// Servo driver and channels
Adafruit_PWMServoDriver pwm;
#define SERVOMIN 150
#define SERVOMAX 600
const float angleOpen   =  30.0;   // degrees for "Open"
const float angleClosed = 150.0;   // degrees for "Close"

// Gesture names
const char* gestureNames[NUM_GESTURES] = { "Open", "Close" };

// For baseline & envelope
float baseline;
float env;

// Calibration storage
float gestureStd[NUM_GESTURES];
float thresholdSD;

// Rolling buffer for live SD
float stdBuffer[STD_WINDOW];
int   bufIndex     = 0;
bool  bufferFilled = false;

//—— Read & process single EMG channel ——
float readProcessed() {
 // throw away one reading for settling
 analogRead(emgPin);
 delayMicroseconds(ADC_SETTLE_US);
 float raw = analogRead(emgPin);
 float corr = raw - baseline;
 return corr > 0 ? corr : 0;  // rectify
}

//—— Send angle to all servos ——
void applyGesture(int g) {
 float target = (g == 0) ? angleOpen : angleClosed;
 uint16_t span = SERVOMAX - SERVOMIN;
 uint16_t pulse = SERVOMIN + uint16_t(target/180.0 * span);
```

```
 for (int m = 0; m < NUM_MOTORS; m++) {
   pwm.setPWM(m, 0, pulse);
 }
}

//—— Compute SD of an array ——
float computeSD(const float* data, int n) {
 float sum = 0, sumSq = 0;
 for (int i = 0; i < n; i++) {
   sum   += data[i];
   sumSq += data[i]*data[i];
 }
 float mean = sum / n;
 float var  = (sumSq / n) - (mean * mean);
 return sqrt(var);
}

//—— Baseline calibration ——
void calibrateBaseline() {
 Serial.println("Calibrating baseline—relax...");
 long acc = 0;
 for (int i = 0; i < BASELINE_SAMPLES; i++) {
   analogRead(emgPin);
   delayMicroseconds(ADC_SETTLE_US);
   acc += analogRead(emgPin);
   delay(5);
 }
 baseline = float(acc) / BASELINE_SAMPLES;
 Serial.print("Baseline = ");
 Serial.println(baseline, 1);
}

//—— SD calibration for Open & Close ——
void calibrateStdDev() {
 Serial.println("=== SD Calibration ===");
 for (int g = 0; g < NUM_GESTURES; g++) {
   Serial.print("Hold \"");
   Serial.print(gestureNames[g]);
   Serial.println("" firmly until done...");
   // wait for user
   while (!Serial.available()) delay(10);
   Serial.read(); // clear keypress
```

```
    // temporary buffer
    float tmp[CAL_STD_SAMPLES];
    env = 0;
    for (int i = 0; i < CAL_STD_SAMPLES; i++) {
      tmp[i] = readProcessed();
      delay(10);
      Serial.print('.');
    }
    Serial.println(" done");

    gestureStd[g] = computeSD(tmp, CAL_STD_SAMPLES);
    Serial.print(" SD "); Serial.print(gestureNames[g]);
    Serial.print(" = "); Serial.println(gestureStd[g], 2);
  }
  // set threshold at midpoint
  thresholdSD = (gestureStd[0] + gestureStd[1]) / 2.0;
  Serial.print("Threshold SD = "); Serial.println(thresholdSD, 2);
  Serial.println("Calibration complete!\n");
}

void setup() {
  Serial.begin(115200);
  while (!Serial) { }
  pwm.begin();
  pwm.setPWMFreq(60);

  // start open
  applyGesture(0);

  calibrateBaseline();
  env = 0.0;
  calibrateStdDev();
}

void loop() {
  // —— allow dynamic threshold setting ——
  if (Serial.available()) {
    float newT = Serial.parseFloat();
    if (newT > 0) {
      thresholdSD = newT;
      Serial.print(">>> New thresholdSD = ");
```

```
    Serial.println(thresholdSD, 2);
  }
  while (Serial.available()) Serial.read(); // flush any extra chars
}

// read & envelope-smooth
float x = readProcessed();
env = SMOOTH_ALPHA * x + (1 - SMOOTH_ALPHA) * env;
 // insert into rolling buffer
stdBuffer[bufIndex++] = env;
if (bufIndex >= STD_WINDOW) {
  bufIndex = 0;
  bufferFilled = true;
}
 // once buffer filled, compute live SD and classify
if (bufferFilled) {
  float liveSD = computeSD(stdBuffer, STD_WINDOW);
  Serial.print("Live SD: "); Serial.print(liveSD, 2);
  Serial.print("  (Thresh: "); Serial.print(thresholdSD, 2);
  Serial.print(") -> ");

  int g = (liveSD > thresholdSD) ? 1 : 0;
  Serial.println(gestureNames[g]);
  applyGesture(g);
}

delay(GESTURE_HOLD_MS);
}
```