Robotic Hand

Aaron Feldman, Chance Martin, Shang-Li Wu, Sami Malek
Professors: Edward A. Lee, Alberto Sangiovanni-Vincentelli
Graduate Student Instructors: Antonio Iannopollo, Ben Zhang, John Finn

University of California, Berkeley

Abstract

The aim of this project is to designing and implementing a robotic hand that is controlled by a combination of sensor inputs including electromyography and voice recognition. The goals will be accurate detection of voice commands, accurate detection of hand gestures via an electromyographic device, and accurate response of the robotic hand to these inputs.

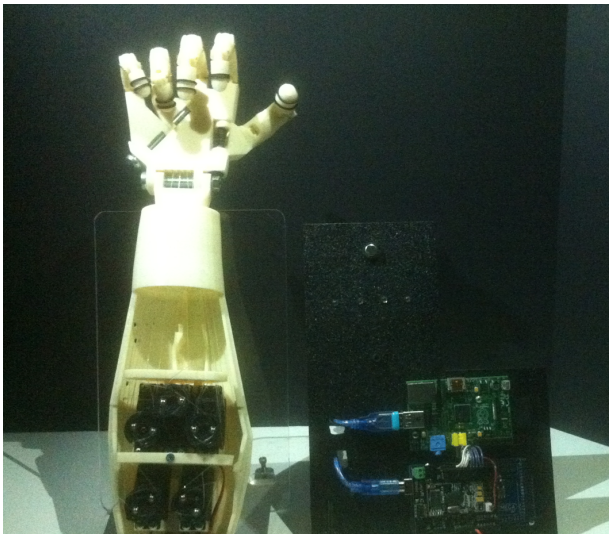

Figure 1: 3D-Printed Robotic Hand Full System

I. System Model

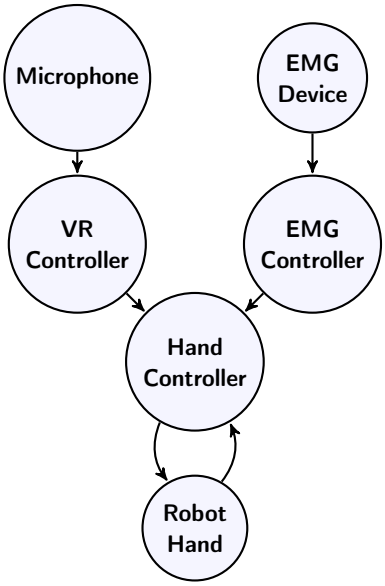

Figure 2: Robotic Hand System Model

Figure 2 contains the course grain overview of the system model. We can see that there are three main inputs to the hand controller: feedback from the hand, input from the EMG controller, and input from the voice recognition controller. This system model allowed us to split the design and implementation into three parts that we were able to complete in parallel.

II. Hardware

Robotic Hand

The robotics hand in our model, is derived from "InMoov," an open source 3D printed life size robot. Parts for downloads are licensed under the attribution - Non - Commercial - Creative Commons license (CC-BY-NC 3.0), in which we are free to share, adapt under the attribution and non-commercial term. The parts we use are the STL file (stereo lithography file format) of hands and forearm.

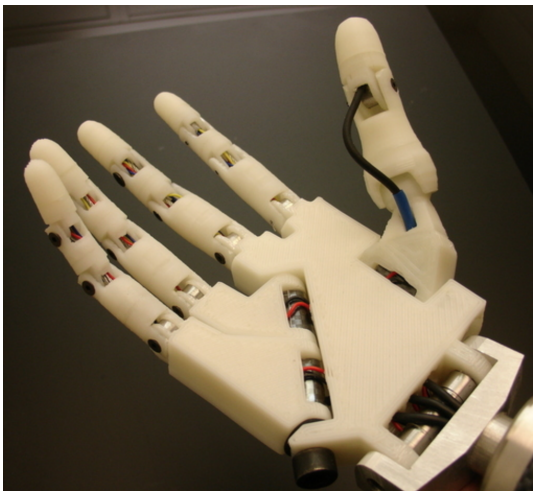

Figure 3: Robotic Hand Close Up

We contracted to the ME machine shop to 3D print the plastic parts for both the hand and forearm. They have a newer version of model material called ABS+, according to the manufacturer this is a stronger, more machinable ABS. They have 3 print options of density: low density, high density, and solid. We choose high density for our forearm, and solid for the other parts in order to get enough structure strength. The parts are printed in SST 1200ES with soluble support material. The soluble support is more suitable for our design since there are small holes in the design and the soluble support material helps to preserve the holes during the printing process.

Figure 4: TowerPro MG995 servomotor

Motors

Motors for robotics come in three main types: standard DC motors, servo motors, and stepper motors. Because the hand needs to be able to correctly position the fingers, it is desirable to have motors with position control. This limits the options to servo motors and stepper motors. However the downside to stepper motors is that if the motor controller is unpowered, it can be difficult to maintain knowledge of the motors current angle. In contrast a R/C servo motor determines its position via a built in potentiometer and controller, thus a R/C servo can determine its position even after a power off. Since we were already planning on running an mbed or arduino embedded system in the hand and thus could handle the pulse width modulation (pwm) signals necessary to drive a servo, we chose this option for the motors. Considerations for choosing the hand servos were price, footprint, torque, range of degrees, and power consumption. We chose to go with the TowerPro MG995 servos as these servos had 10.00 kg-cm at 4.8V of torque, had a small footprint, operated at were low priced, had 180° range, operated between 4V-6V, averaged 500mA, and were available on Amazon, so we could get them quickly and start building and testing.

Controllers

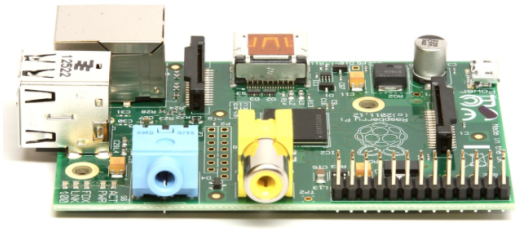

Figure 5: Raspberry Pi Model B

For the EMG controller we chose the Raspberry Pi Model B. It has a 700 MHz ARM Core Processor with 512 MB SDRAM. This would allow us enough flexibility to run an operating system with a file system and expanded ability to run different scripting languages including Python. Furthermore, the Raspberry Pi has two USB ports, necessary to connect a USB bluetooth low energy (BLE) controller dongle to communicate with the EMG sensor. The other being an easy way to communicate with the robotic hand controller serially.

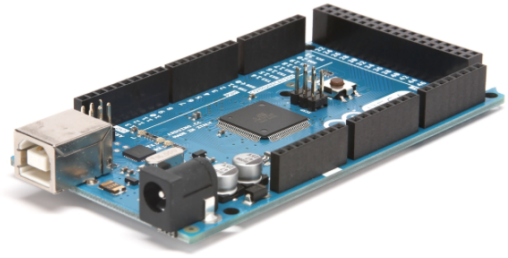

Figure 6: Arduino Mega 2560 R3 (ATmega2560)

For the hand controller we chose the Arduino ATmega2560. Timing is a critical issue with the servos in the robotic hand. The signals are PWM and need accurate hardware timing to provide the correct width of high voltages. While PWM can be replicated by "bit-banging" a GPIO pin, this does not guarantee accurate timing and thus widths of the PWM pulses as a simple print statement can tie up the processor longer than a PWM pulse. The Arduino Mega provides 12 PWM pins with hardware timing. This is more than the 5 we need for the hand servos and also allows us the opportunity

to use a few for LEDs.

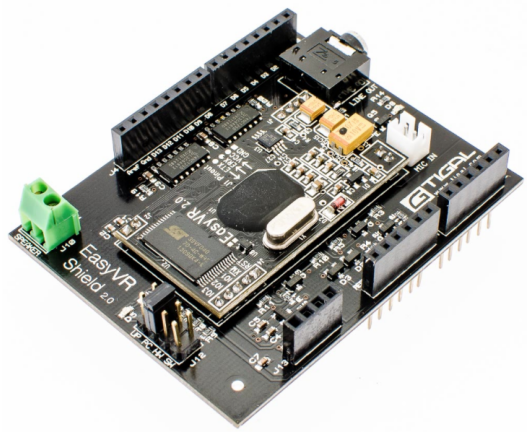

Figure 7: TIGAL EasyVR Arduino Shield 2.0

For the voice recognition controller we chose the TIGAL EasyVR Arduino Shield 2.0. The The EasyVR module can be used with any host with an UART interface powered at 3.3V âĂŞ 5V. This allowed us to easily integrate the system with the Arduino board. This system also comes with command training software, allowing us to use the voice commands module as a black box.

III. Power

A large challenge in powering the hand is maintaining the correct voltage for each of the electrical components. The Arduino can take anywhere from 5V - 20V so it is the most flexible. The servos needed between 4V - 6V and the Raspberry Pi needed 5V.

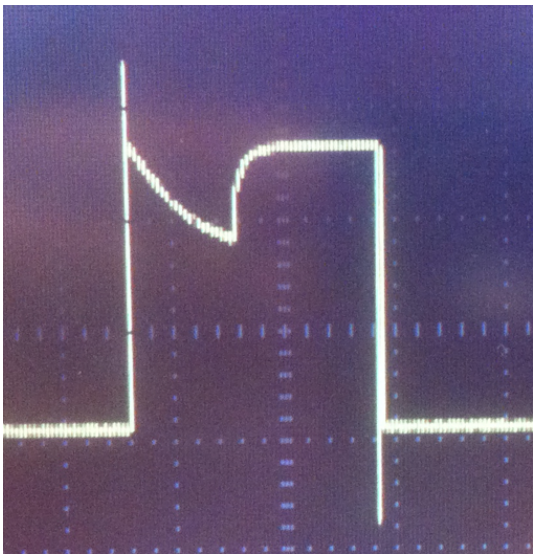

Figure 8: Faulty PWM Signal

However with we found that more than one servo won't work off the 5V power output from an Arduino board as they could take up to 1.1A of current and that was more than the Arduino could supply. When the servos consumed more amperage than the Arduino could supply, it produced erratic PWM signals which in kind produced erratic servo behavior.

In order to solve this problem, we implemented a power circuit to provide power to the servos with a battery supply. But due to overcharging the battery we supplied 7V to the servos and after burning a few of their controller boards, we discovered they do not operate very far outside of their rated voltages as can be seen in the following figure. For future iterations, we suggest more robust voltage regulation circuitry to prevent overpowering of the Raspberry Pi and servos.

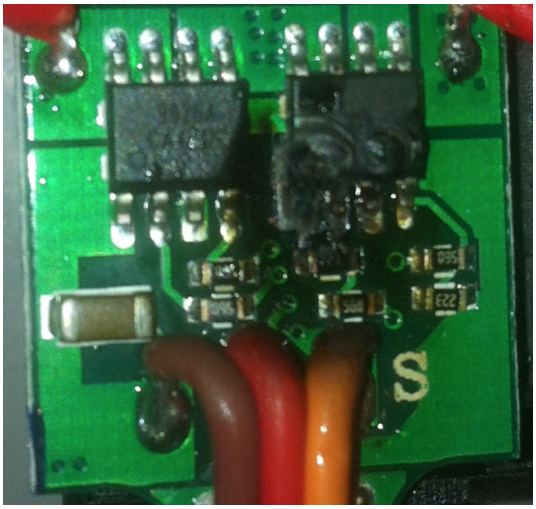

Figure 9: Burnt Servo Board

IV. Voice Recognition

We considered using a Raspberry Pi as the voice recognition controller. On the Raspberry Pi there are three leading ways to perform voice recognition: Jasper âĂŞ Voice Recognition Software, Raspberry Pi Voice Recognition by Oscar Liang, Raspberry Pi Voice Control by Steven Hickson. The voice control software packages by Liang and Hickson both use Google APIs and as such require the Raspberry Pi to be connected to the internet. While this was within the capability of the Raspberry Pi via a wireless controller, connecting to the internet was something we didn't want to do. We wanted to create a self contained unit that could be used remotely if possible and absent the internet. This left the Raspberry Pi with the option of Jasper, which we implemented and tested. We found however that Jasper wasn't

as responsive on the Raspberry Pi as we were hoping and it also consumed a lot of resources on the controller.

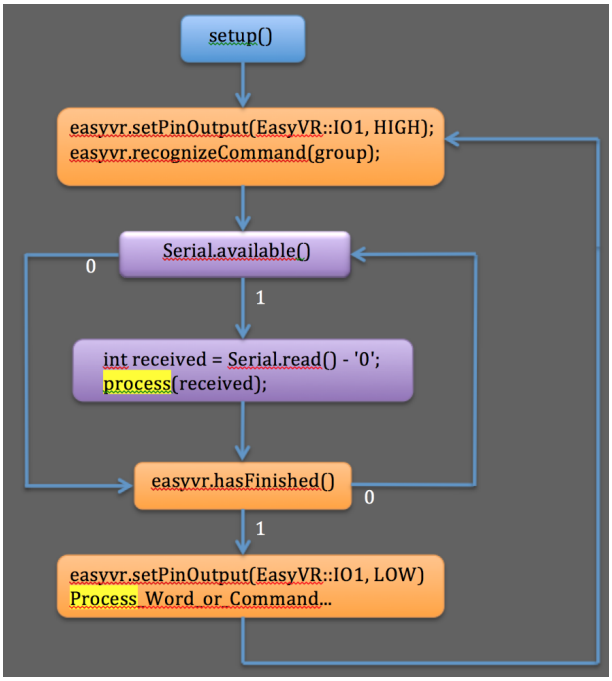

Figure 10: Software Flow Chart

This then led us to consider using alternative controllers. We found a video by Patrick S where he used a TIGAL EasyVR shield attached to an Arduino to perform voice recognition: https://www.youtube.com/watch?v=zz6hT3kDhOc. He used a robotic hand similar to the one we were using and provided an instructions on his implementation. The EasyVR shield allowed us to process the voice recognition separately from the Arduino, freeing up its processing power so we could drive the servo motors. However in testing, we found that this voice recognition shield only provided about 40% accuracy in recognizing words under good conditions, and worse under adverse conditions.

For the next iteration of the voice controller, we suggest a more controller with a more powerful processor, perhaps a PCDuino which runs in the 1GHz range with either Jasper or custom software. There are tutorials online that suggest closer to a 80% accuracy range that could be implemented.

V. Electromyography

We considered multiple ways of implementing an electromyograph including building on from scratch or the Arduino SHIELD-EKG-EMG which was inexpensive but required placing electrodes on the arm and buying disposable electrode stickies. Instead we found a group on campus at UC Berkeley that had a beta developer version of the Myo Armband. This armband could be placed on the arm easily with no disposables, and could easily be placed into the same position multiple times.

However one problem was that we wanted access to the raw data. The Myo firmware processed all the data on the armband itself and then sent out preprocessed data. Fortunately we were able to locate an older firmware version (0.8.8.12) that allowed us access to the raw data. With this we were able to find an individual who had created a library to connect to the Myo armband and process the data in Python: https://github.com/jwcrawley/myo-raw. He provides this under the MIT license. However we found that this package wouldn't run on the Raspberry Pi. So in turn we optimized the code and also disabled unnessary things on the Raspberry Pi like the accelerometer that were taking up processing power.

We trained 1000 samples for each of 8 gestures in Python using the sklearn library with a kd-nearest neighbor maching learning algorithm. This plotted each gesture on an x-y graph. We then took 15 samples of the gesture we were trying to determine, plotted it, found the 25 closest training points and took a majority vote to determine the gesture the Myo armband was reading. We found in testing that we had over 85% accuracy rate in determining the correct gesture. The accuracy rate went up, closer to 100% with a limited set of gestures that were uniquely different.

VI. Testing

Motors

We found that more than one servo won't work off the 5V power output from an Arduino board, so we implemented a power circuit to provide power to the servos. We also found that even though the servos are rated to have a 180° rotational range with a pulse width between 1000-2000 μs. However we found that at that range it only had about 150° rotation. This worked out in our favor because we realized that it is possible to easily adjust the pulse width range in the Arduino sketch when attaching the servo: $myservo.attach(9, 544, 3550)$. We found that a pulse width range of 544-3550 μs provides about 190° of safe rotation for the servo. Greater rotation appeared to be detrimental to the servos. It is

our thought going forward that we will use this limit of ranging to control the basics of the finger range to keep them within a safe desired amount so there will be no damage to the robotic hands wires or hardware.

Hand Control

Using the Matlab support package for Arduino, we've been able to test the servos outside of the robotic hand, so that we don't damage the hand, and so that we were able to get a jump start while a few things were being finished on the hand. Since all of the basic movements seem to be ok, we're looking into adding the ability to not just select hand positions, but being able to grasp things. We're considering two further options, one being built in finger tip sensors, or possibly another in that we get feedback that the position of the servos don't change after some small amount of time, thus meaning that the servos encountered an obstruction (the object being grasped). Since we've used 40lb test fishing string, this will act as a possible way to grasp things.

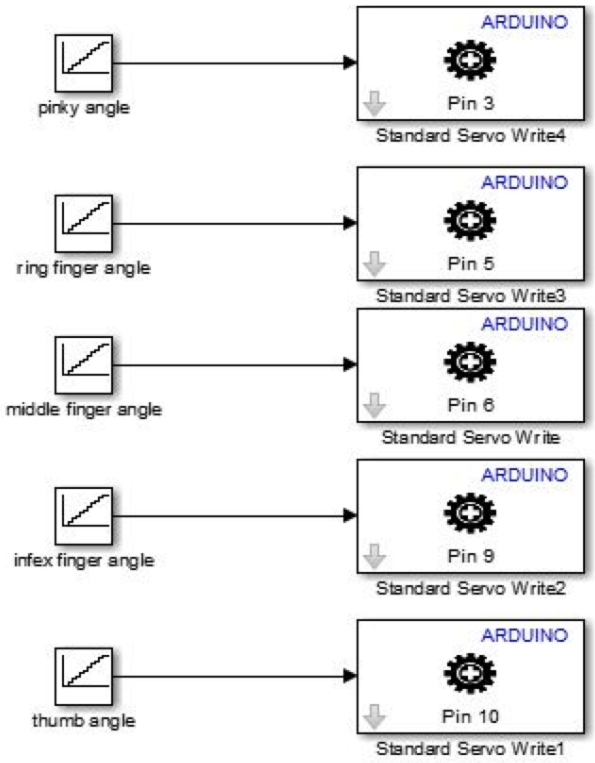

Figure 11: TowerPro MG995 servomotor

Communication between Controllers

We were able to test communication between the Raspberry Pi and the Arduino board over USB serial communication. We built a sending function in Python on the Raspberry Pi and a receiving function in C on the Arduino. This allowed us to turn on and off LEDs. This testing allowed us to determine that the Raspberry Pi and Arduino could send and receive bytes before the EMG controller software was finished. Similarly we tested commands on the voice recognition controller to the Arduino hand controller.

References

[1] http://it.wikipedia.org/wiki/InMoov

[2] http://www.inmoov.fr/hand-and-forarm/

[3] http://www.me.berkeley.edu/new/Shop/FDM_protocol.pdf

[4] https://github.com/jwcrawley/myo-raw

[5] http://www.protoparadigm.com/news-updates/the-difference-between-abs-and-pla-for-3d-printing/

[6] http://www.servodatabase.com/servo/towerpro/mg995

[7] http://www.amci.com/tutorials/tutorials-stepper-vs-servo.asp

[8] http://arduino.cc/en/Tutorial/SecretsOfArduinoPWM

CODE TITLE

Servo Drive Function

```
// Drive function to control servo speed and reduce current draw
void drive(Servo curr_servo, int target_pos, int s_speed) {
  int curr_pos = curr_servo.read();
  if (curr_pos < target_pos) {
    for (; curr_pos < target_pos;) {
      curr_servo.write(++curr_pos);
      delay(s_speed);
    }
    //  Backoff five to reduce servo current draw
    delay(500);
    curr_servo.write(curr_servo.read() - 5);
  } else {
      for (; curr_pos > target_pos;) {
      curr_servo.write(--curr_pos);
      delay(s_speed);
    }
    //  Backoff five to reduce servo current draw
    delay(500);
    curr_servo.write(curr_servo.read() + 5 );
  }
}
```

Arduino.ino

```
#if defined(ARDUINO) && ARDUINO ≥ 100
  #include "Arduino.h"
  #include "Platform.h"
  #include "SoftwareSerial.h"
  #include "Servo.h"
  #include "WProgram.h"
  #include "NewSoftSerial.h"
  NewSoftSerial port(12,13);
#endif

#include "EasyVR.h"
EasyVR easyvr(port);

// Groups

enum Groups {
  GROUP_0  = 0,
  GROUP_1  = 1,
};

enum Group0 {
  G0_HAND = 0,
};

enum Group1
{
  G1_GO_BEARS = 0,
  G1_STANFORD = 1,
  G1_HANG_LOOSE = 2,
  G1_PEACE = 3,
  G1_THUMBS_UP = 4,
  G1_RELAX = 5,
  G1_OPEN = 6,
  G1_ROCK_ON = 7,
  G1_POINT = 8,
  G1_WAVE = 9,
  G1_CLOSE = 10,
  G1_COME_HERE = 11,
};
```

```
40
41 int8_t group, idx;
42
43 Servo myservo1, myservo2, myservo3, myservo4, myservo5;
44
45 //Myo mappings
46 /*
47 0 --> rest
48 1 --> fist (not contracted)
49 2 --> pinky to thumb    (CONTRACT)
50 3 --> point (not contracted)
51 4 --> thumbs up (not contracted
52 5 --> peace  (palm towards out)
53 6X -->
54 7X-->
55 8 --> open wide (don't contract)
56 */
57
58 //Myo_funcs
59 void rest12() {
60         myservo1.write(100);
61         delay(50);
62         myservo2.write(80);
63 }
64
65 void zero() {  //rest
66         myservo1.write(100);
67         delay(50);
68         myservo5.write(100);
69         delay(50);
70         myservo4.write(50);
71         delay(50);
72         myservo2.write(80);
73         myservo3.write(80);
74         delay(50);
75         myservo1.write(130);
76 }
77 void one() {  //fist
78         myservo1.write(100);
79         delay(100);
80         myservo2.write(150);
81         myservo3.write(150);
82         myservo4.write(150);
83         delay(100);
84         myservo5.write(150);
85         delay(100);
86         myservo1.write(150);
87 }
88 void two() {  //pinky_to_thumb
89         myservo5.write(100);
90         myservo1.write(100);
91         delay(100);
92         myservo2.write(0);
93         myservo3.write(0);
94         myservo4.write(0);
95         delay(200);
96         myservo5.write(150);
97         myservo1.write(150);
98 }
99 void three() {  //point
100         myservo1.write(100);
101         myservo3.write(150);
102         delay(100);
103         myservo2.write(0);
104         delay(50);
105         myservo4.write(150);
106         delay(50);
```

```
107         myservo5.write(150);
108         delay(50);
109         myservo1.write(150);
110 }
111 void four() {  //thumbs_up
112         myservo1.write(0);
113         myservo2.write(145);
114         myservo3.write(145);
115         myservo4.write(145);
116         myservo5.write(145);
117 }
118 void five() {  //peace
119         rest12();
120         delay(100);
121         myservo1.write(145);
122         myservo2.write(0);
123         myservo3.write(0);
124         myservo4.write(145);
125         myservo5.write(145);
126 }
127
128
129 void six() {  //middle finger
130         rest12();
131         delay(100);
132         myservo1.write(100);
133         delay(100);
134         myservo2.write(150);
135         myservo3.write(0);
136         myservo4.write(150);
137         delay(100);
138         myservo5.write(150);
139         delay(100);
140         myservo1.write(150);
141 }
142 void seven() {
143 }
144 void eight() {  //open_wide
145         myservo1.write(0);
146         myservo2.write(0);
147         myservo3.write(0);
148         myservo4.write(0);
149         myservo5.write(0);
150 }
151 void nine(){
152 }
153
154 void process(int received) {
155
156        if( received > 4){
157        if(received > 6){  // >6
158          if(received > 8){    // 9
159            Serial.println(9);
160            nine();
161          }
162          else{  // > 6  ≤8
163            if(received>7){  // 8
164              Serial.println(8);
165                eight();
166            }
167            else{  // 7
168              Serial.println(7);
169              seven();
170            }
171          }
172        }
173        else{  //  4 < received  ≤ 6
```

```
174           if(received > 5){
175             Serial.println(6);
176             six();
177           }
178           else{
179             Serial.println(5);
180             //5
181             five();
182           }
183         }
184       }
185       else{  // received ≤4
186         if( received ≤2){
187           if(received ≤1){ // 1 or 0
188             if(received > 0){
189                 // 1
190                 Serial.println(1);
191                 one();
192             }
193             else{
194                 //0
195                 Serial.println(0);
196                 zero();
197             }
198           }
199           else{  // > 1  ≤2
200               // 2
201               Serial.println(2);
202               two();
203           }
204         }
205         else{  //  2<  ≤4
206           if(received > 3){
207             Serial.println(4);
208             four();
209             // 4
210           }
211           else{
212             // 3
213             Serial.println(3);
214             three();
215           }
216         }
217       }
218 }
219
220
221 void setup() {
222     Serial.begin(9600);
223     myservo1.attach(3); //thumb
224     myservo2.attach(4); //index
225     myservo3.attach(5); //middle
226     myservo4.attach(6); //ring
227     myservo5.attach(7); //pinky
228     pinMode(8,OUTPUT);
229     pinMode(11,OUTPUT);
230
231   port.begin(9600);
232
233   while (!easyvr.detect()) {
234     Serial.println("EasyVR not detected!");
235     delay(1000);
236   }
237
238   easyvr.setPinOutput(EasyVR::IO1, LOW);
239   Serial.println("EasyVR detected!");
240   easyvr.setTimeout(5);
```

```
241   easyvr.setLanguage(0);
242   group = EasyVR::TRIGGER; //<-- start group (customize)
243
244 }
245
246 void action();
247
248 void loop() {
249   if (Serial.available())  {
250      //light(Serial.read() - '0');
251      int received = Serial.read() - '0';
252      process(received);
253   }
254   else {
255     easyvr.setPinOutput(EasyVR::IO1, HIGH); // LED on (listening)
256     Serial.print("Say a command in Group ");
257     Serial.println(group);
258     easyvr.recognizeCommand(group);
259
260     while (!easyvr.hasFinished());
261     easyvr.setPinOutput(EasyVR::IO1, LOW); // LED off
262     idx = easyvr.getWord();
263
264     if (idx ≥ 0) {
265
266       // built-in trigger (ROBOT)
267       group = GROUP_1;
268
269       return;
270
271     }
272
273     idx = easyvr.getCommand();
274
275     if (idx ≥ 0) {
276
277       // print debug message
278
279       uint8_t train = 0;
280
281       char name[32];
282
283       Serial.print("Command: ");
284
285       Serial.print(idx);
286
287       if (easyvr.dumpCommand(group, idx, name, train))
288
289       {
290
291         Serial.print(" = ");
292
293         Serial.println(name);
294
295       }
296
297       else
298
299         Serial.println();
300
301       easyvr.playSound(0, EasyVR::VOL_FULL);
302
303       // perform some action
304
305       action();
306
307     }
```

```
308
309     else // errors or timeout
310
311     {
312
313       if (easyvr.isTimeout())
314
315         Serial.println("Timed out, try again...");
316
317       int16_t err = easyvr.getError();
318
319       if (err ≥ 0)
320
321       {
322         digitalWrite(11,HIGH);
323         delay(1000);
324         digitalWrite(11,LOW);
325
326         Serial.print("Error ");
327
328         Serial.println(err, HEX);
329
330       }
331
332     }
333
334   }
335 }
336
337 void action() {
338
339     switch (group) {
340
341     case GROUP_0:
342       switch (idx)
343       {
344       case G0_HAND:
345
346         digitalWrite(8,HIGH);
347         group =GROUP_1;
348
349         break;
350       }
351       break;
352     case GROUP_1:
353       switch (idx)
354       {
355       case G1_GO_BEARS:
356
357         myservo2.write(150);
358         myservo3.write(150);
359         myservo4.write(150);
360         myservo5.write(150);
361         myservo1.write(150);
362         delay(3);
363         myservo1.write(145);
364         myservo2.write(145);
365         myservo3.write(145);
366         myservo4.write(145);
367         myservo5.write(145);
368
369         break;
370       case G1_STANFORD:
371
372         rest12();
373         delay(100);
374         myservo1.write(145);
```

```
375           myservo2.write(145);
376           myservo3.write(15);
377           myservo4.write(145);
378           myservo5.write(145);
379
380           break;
381         case G1_HANG_LOOSE:
382
383           myservo1.write(0);
384           myservo2.write(145);
385           myservo3.write(145);
386           myservo4.write(145);
387           myservo5.write(0);
388
389           break;
390         case G1_PEACE:
391
392           rest12();
393           delay(100);
394           myservo1.write(145);
395           myservo2.write(0);
396           myservo3.write(0);
397           myservo4.write(145);
398           myservo5.write(145);
399
400           break;
401         case G1_THUMBS_UP:
402
403           rest12();
404           delay(100);
405           myservo1.write(0);
406           myservo2.write(145);
407           myservo3.write(145);
408           myservo4.write(145);
409           myservo5.write(145);
410
411           break;
412         case G1_RELAX:
413
414           zero();
415
416           break;
417         case G1_OPEN:
418
419           myservo1.write(0);
420           myservo2.write(0);
421           myservo3.write(0);
422           myservo4.write(0);
423           myservo5.write(0);
424
425           break;
426         case G1_ROCK_ON:
427
428           rest12();
429           delay(100);
430           myservo1.write(150);
431           myservo2.write(0);
432           myservo3.write(150);
433           myservo4.write(150);
434           myservo5.write(0);
435
436           break;
437         case G1_POINT:
438
439           myservo1.write(150);
440           myservo2.write(0);
441           myservo3.write(150);
```

```
442           myservo4.write(150);
443           myservo5.write(150);
444 
445           break;
446         case G1_WAVE:
447           myservo1.write(0);
448           myservo2.write(0);
449           myservo3.write(0);
450           myservo4.write(25);
451           myservo5.write(0);
452 
453           break;
454         case G1_CLOSE:
455 
456           myservo1.write(150);
457           myservo2.write(150);
458           myservo3.write(150);
459           myservo4.write(150);
460           myservo5.write(150);
461 
462           break;
463         case G1_COME_HERE:
464 
465           rest12();
466           delay(100);
467           myservo1.write(150);
468           myservo2.write(0);
469           myservo3.write(150);
470           myservo4.write(150);
471           myservo5.write(150);
472           drive(myservo2,70,4);
473           drive(myservo2,0,4);
474           drive(myservo2,70,4);
475           drive(myservo2,0,4);
476           drive(myservo2,70,4);
477           drive(myservo2,0,4);
478           drive(myservo2,70,4);
479           drive(myservo2,0,4);
480 
481           break;
482         }
483         break;
484       }
485 
486 }
```

Train Gestures

```
1  from __future__ import print_function
2 
3  from collections import Counter
4  import struct
5  import sys
6  import time
7  import numpy as np
8  FIST = 5
9  PINKY = 4
10 SPREAD = 3
11 FLEX = 2
12 EXTEND = 1
13 try:
14     from sklearn import neighbors, svm
15     HAVE_SK = True
16 except ImportError:
17     HAVE_SK = False
18 
19 try:
```

```
    import pygame
    from pygame.locals import *
    HAVE_PYGAME = True
except ImportError:
    HAVE_PYGAME = False

from common import *
import myo

class EMGHandler(object):
    def __init__(self, m):
        self.recording = -1
        self.m = m
        self.emg = (0,) * 8

    def __call__(self, emg, moving):
        self.emg = emg
        if self.recording >= 0:
            self.m.cls.store_data(self.recording, emg)

if __name__ == '__main__':
    if HAVE_PYGAME:
        pygame.init()
        w, h = 800, 320
        scr = pygame.display.set_mode((w, h))
        font = pygame.font.Font(None, 30)

    m = myo.Myo(myo.NNClassifier(), sys.argv[1] if len(sys.argv) >= 2 else None)
    hnd = EMGHandler(m)
    m.add_emg_handler(hnd)
    m.connect()
    pressedr = None
    rcounter = 0
    rtime = 10
    rcutoff = 23
    pressedkeys = [None]
    try:
        while True:
            m.run()

            r = m.history_cnt.most_common(1)[0][0]
            if m.history_cnt[r] > rcutoff:
                if pressedr != r:
                    pressedr = r
                    rcounter = 0
                else:
                    rcounter += 1
                    if rcounter >= rtime:
                        if pressedkeys[-1] != r:
                            print('pressed', r)
                            pressedkeys.append(r)
                            if len(pressedkeys) > 20:
                                pressedkeys.pop(0)
            if HAVE_PYGAME:
                for ev in pygame.event.get():
                    if ev.type == QUIT or (ev.type == KEYDOWN and ev.unicode == 'q'):
                        raise KeyboardInterrupt()
                    elif ev.type == KEYDOWN:
                        if K_0 <= ev.key <= K_9:
                            hnd.recording = ev.key - K_0
                        elif K_KP0 <= ev.key <= K_KP9:
                            hnd.recording = ev.key - K_Kp0
                        elif ev.unicode == 'r':
                            hnd.cl.read_data()
                    elif ev.type == KEYUP:
                        if K_0 <= ev.key <= K_9 or K_KP0 <= ev.key <= K_KP9:
                            hnd.recording = -1
```

```
87
88                  scr.fill((0, 0, 0), (0, 0, w, h))
89
90                  for i in range(10):
91                      x = 0
92                      y = 0 + 30 * i
93
94                      clr = (0,200,0) if i == r else (255,255,255)
95
96                      txt = font.render('%5d' % (m.cls.Y == i).sum(), True, (255,255,255))
97                      scr.blit(txt, (x + 20, y))
98
99                      txt = font.render('%d' % i, True, clr)
100                     scr.blit(txt, (x + 110, y))
101
102
103                     scr.fill((0,0,0), (x+130, y + txt.get_height() / 2 - 10, len(m.history) * 20, ...
                            20))
104                     scr.fill(clr, (x+130, y + txt.get_height() / 2 - 10, m.history_cnt[i] * 20, 20))
105
106                 if HAVE_SK and m.cls.nn != None:
107
108                     dists, inds = m.cls.nn.kneighbors(hnd.emg)
109                     for i, (d, ind) in enumerate(zip(dists[0], inds[0])):
110                         y = m.cls.Y[myo.NUMSAMPLES*ind]
111                         text(scr, font, '%d %6d' % (y, d), (650, 20 * i))
112
113                 pygame.display.flip()
114             else:
115                 for i in range(10):
116                     if i == r: sys.stdout.write('\x1b[32m')
117                     print(i, '-' * m.history_cnt[i], '\x1b[K')
118                     if i == r: sys.stdout.write('\x1b[m')
119                 sys.stdout.write('\x1b[11A')
120                 print()
121
122     except KeyboardInterrupt:
123         pass
124     finally:
125         m.disconnect()
126         print()
127
128     if HAVE_PYGAME:
129         pygame.quit()
```

Classify Gesture

```
1  from __future__ import print_function
2
3  import re
4  import struct
5  import sys
6  import threading
7  import time
8
9  import serial
10 from serial.tools.list_ports import comports
11
12 from common import *
13
14 def multichr(ords):
15     if sys.version_info[0] ≥ 3:
16         return bytes(ords)
17     else:
18         return ''.join(map(chr, ords))
19
20 def multiord(b):
```

```
21      if sys.version_info[0] ≥ 3:
22          return list(b)
23      else:
24          return map(ord, b)
25
26
27  class Packet(object):
28      def __init__(self, ords):
29          self.typ = ords[0]
30          self.cls = ords[2]
31          self.cmd = ords[3]
32          self.payload = multichr(ords[4:])
33
34      def __repr__(self):
35          return 'Packet(%02X, %02X, %02X, [%s])' % \
36              (self.typ, self.cls, self.cmd,
37               ' '.join('%02X' % b for b in multiord(self.payload)))
38
39
40  class BT(object):
41      '''Implements the non-Myo-specific details of the Bluetooth protocol.'''
42      def __init__(self, tty):
43          self.ser = serial.Serial(port=tty, baudrate=9600, dsrdtr=1)
44          self.buf = []
45          self.lock = threading.Lock()
46          self.handlers = []
47
48      ## internal data-handling methods
49      def recv_packet(self, timeout=None):
50          t0 = time.time()
51          try:
52              self.ser.timeout = timeout
53          except:
54              print('recv timeout error')
55              pass
56          while timeout is None or time.time() < t0 + timeout:
57              if timeout is not None:
58                  self.ser.timeout = t0 + timeout - time.time()
59              c = self.ser.read()
60              if not c:
61                  print("found serial None")
62                  return None
63
64              ret = self.proc_byte(ord(c))
65              if ret:
66                  if ret.typ == 0x80:
67                      try:
68                          self.handle_event(ret)
69                      except:
70                          return
71                  return ret
72
73      def recv_packets(self, timeout=1.0):
74          res = []
75          t0 = time.time()
76          while time.time() < t0 + timeout:
77              p = self.recv_packet(t0 + timeout - time.time())
78              if not p: return res
79              res.append(p)
80          return res
81
82      def proc_byte(self, c):
83          if not self.buf:
84              if c in [0x00, 0x80, 0x08, 0x88]:
85                  self.buf.append(c)
86              return None
87          elif len(self.buf) == 1:
```

```
88              self.buf.append(c)
89              self.packet_len = 4 + (self.buf[0] & 0x07) + self.buf[1]
90              return None
91          else:
92              self.buf.append(c)
93
94          if self.packet_len and len(self.buf) == self.packet_len:
95              p = Packet(self.buf)
96              self.buf = []
97              return p
98          return None
99
100     def handle_event(self, p):
101         for h in self.handlers:
102             h(p)
103
104     def add_handler(self, h):
105         self.handlers.append(h)
106
107     def remove_handler(self, h):
108         try: self.handlers.remove(h)
109         except ValueError: pass
110
111     def wait_event(self, cls, cmd):
112         res = [None]
113         def h(p):
114             if p.cls == cls and p.cmd == cmd:
115                 res[0] = p
116         self.add_handler(h)
117         while res[0] is None:
118             self.recv_packet()
119         self.remove_handler(h)
120         return res[0]
121
122     ## specific BLE commands
123     def connect(self, addr):
124         return self.send_command(6, 3, pack('6sBHHHH', multichr(addr), 0, 6, 6, 64, 0))
125
126     def get_connections(self):
127         return self.send_command(0, 6)
128
129     def discover(self):
130         return self.send_command(6, 2, b'\x01')
131
132     def end_scan(self):
133         return self.send_command(6, 4)
134
135     def disconnect(self, h):
136         return self.send_command(3, 0, pack('B', h))
137
138     def read_attr(self, con, attr):
139         self.send_command(4, 4, pack('BH', con, attr))
140         return self.wait_event(4, 5)
141
142     def write_attr(self, con, attr, val):
143         self.send_command(4, 5, pack('BHB', con, attr, len(val)) + val)
144         return self.wait_event(4, 1)
145
146     def send_command(self, cls, cmd, payload=b'', wait_resp=True):
147         s = pack('4B', 0, len(payload), cls, cmd) + payload
148         self.ser.write(s)
149
150         while True:
151             p = self.recv_packet()
152
153             ## no timeout, so p won't be None
154             if p.typ == 0: return p
```

```
155
156             ## not a response: must be an event
157             self.handle_event(p)
158
159
160 class MyoRaw(object):
161     '''Implements the Myo-specific communication protocol.'''
162
163     def __init__(self, tty=None):
164         if tty is None:
165             tty = self.detect_tty()
166         if tty is None:
167             raise ValueError('Myo dongle not found!')
168
169         self.bt = BT(tty)
170         self.emg_handlers = []
171         self.imu_handlers = []
172
173     def detect_tty(self):
174         for p in comports():
175             if re.search(r'PID=2458:0*1', p[2]):
176                 print('using device:', p[0])
177                 return p[0]
178
179         return None
180
181     def run(self):
182         self.bt.recv_packet()
183
184     def connect(self):
185         ## stop everything from before
186         self.bt.end_scan()
187         self.bt.disconnect(0)
188         self.bt.disconnect(1)
189         self.bt.disconnect(2)
190
191         ## start scanning
192         print('scanning...')
193         self.bt.discover()
194         while True:
195             p = self.bt.recv_packet()
196             print('scan response:', p)
197             if p.payload[15:] == ...
                    b'\x06\x42\x48\x12\x4A\x7F\x2C\x48\x47\xB9\xDE\x04\xA9\x01\x00\x06\xD5':
198                 addr = list(multiord(p.payload[2:8]))
199                 break
200         self.bt.end_scan()
201
202         ## connect and wait for status event
203         conn_pkt = self.bt.connect(addr)
204         self.conn = multiord(conn_pkt.payload)[-1]
205         self.bt.wait_event(3, 0)
206
207         ## get firmware version
208         fw = self.bt.read_attr(self.conn, 0x17)
209         _, _, _, _, v0, v1, v2, v3 = unpack('BHBBHHHH', fw.payload)
210         print('firmware version: %d.%d.%d.%d' % (v0, v1, v2, v3))
211
212         ## don't know what these do; Myo Connect sends them, though we get data
213         ## fine without them
214         self.bt.write_attr(self.conn, 0x19, b'\x01\x02\x00\x00')
215         self.bt.write_attr(self.conn, 0x2f, b'\x01\x00')
216         self.bt.write_attr(self.conn, 0x2c, b'\x01\x00')
217         self.bt.write_attr(self.conn, 0x32, b'\x01\x00')
218         self.bt.write_attr(self.conn, 0x35, b'\x01\x00')
219
220         ## enable EMG data
```

```
221         self.bt.write_attr(self.conn, 0x28, b'\x01\x00')
222         ## enable IMU data
223         self.bt.write_attr(self.conn, 0x1d, b'\x01\x00')
224 
225         ## Sampling rate of the underlying EMG sensor, capped to 1000. If it's
226         ## less than 1000, emg_hz is correct. If it is greater, the actual
227         ## framerate starts dropping inversely. Also, if this is much less than
228         ## 1000, EMG data becomes slower to respond to changes. In conclusion,
229         ## 1000 is probably a good value.
230         C = 1000
231         emg_hz = 50
232         ## strength of low-pass filtering of EMG data
233         emg_smooth = 100
234 
235         imu_hz = 50
236 
237         ## send sensor parameters, or we don't get any data
238         self.bt.write_attr(self.conn, 0x19, pack('BBBBHBBBBB', 2, 9, 2, 1, C, emg_smooth, C // ...
                emg_hz, imu_hz, 0, 0))
239 
240         ## add data handlers
241         def handle_data(p):
242             # print(len(p.payload[:4]))
243             c, attr, typ = unpack('BHB', p.payload[:4])
244             pay = p.payload[5:]
245 
246             if attr == 0x27:
247                 vals = unpack('8HB', pay)
248                 ## not entirely sure what the last byte is, but it's a bitmask that
249                 ## seems to indicate which sensors think they're being moved around or
250                 ## something
251                 emg = vals[:8]
252                 moving = vals[8]
253                 self.proc_emg(emg, moving)
254             if attr == 0x1c:
255                 vals = unpack('10h', pay)
256                 quat = vals[:4]
257                 acc = vals[4:7]
258                 gyro = vals[7:10]
259                 self.proc_imu(quat, acc, gyro)
260 
261         self.bt.add_handler(handle_data)
262 
263 
264     def disconnect(self):
265         self.bt.disconnect(self.conn)
266 
267 
268     def vibrate(self, length):
269         if length in xrange(1, 4):
270             ## first byte tells it to vibrate; purpose of second byte is unknown
271             self.bt.write_attr(self.conn, 0x19, pack('3B', 3, 1, length))
272 
273 
274     def add_emg_handler(self, h):
275         self.emg_handlers.append(h)
276 
277     def add_imu_handler(self, h):
278         self.imu_handlers.append(h)
279 
280     def proc_emg(self, emg, moving):
281         for h in self.emg_handlers:
282             h(emg, moving)
283 
284     def proc_imu(self, quat, acc, gyro):
285         for h in self.imu_handlers:
286             h(quat, acc, gyro)
```

```
287
288
289 if __name__ == '__main__':
290     try:
291         import pygame
292         from pygame.locals import *
293         HAVE_PYGAME = True
294     except ImportError:
295         HAVE_PYGAME = False
296
297     if HAVE_PYGAME:
298         w, h = 1200, 400
299         scr = pygame.display.set_mode((w, h))
300
301     last_vals = None
302     def plot(scr, vals):
303         DRAW_LINES = False
304
305         global last_vals
306         if last_vals is None:
307             last_vals = vals
308             return
309
310         D = 5
311         scr.scroll(-D)
312         scr.fill((0,0,0), (w - D, 0, w, h))
313         for i, (u, v) in enumerate(zip(last_vals, vals)):
314             if DRAW_LINES:
315                 pygame.draw.line(scr, (0,255,0),
316                                  (w - D, int(h/8 * (i+1 - u))),
317                                  (w, int(h/8 * (i+1 - v))))
318                 pygame.draw.line(scr, (255,255,255),
319                                  (w - D, int(h/8 * (i+1))),
320                                  (w, int(h/8 * (i+1))))
321             else:
322                 c = int(255 * max(0, min(1, v)))
323                 scr.fill((c, c, c), (w - D, i * h / 8, D, (i + 1) * h / 8 - i * h / 8));
324
325         pygame.display.flip()
326         last_vals = vals
327
328     m = MyoRaw(sys.argv[1] if len(sys.argv) >= 2 else None)
329
330     def proc_emg(emg, moving, times=[]):
331         if HAVE_PYGAME:
332             ## update pygame display
333             plot(scr, [e / 2000. for e in emg])
334         else:
335             print(emg)
336
337         ## print framerate of received data
338         times.append(time.time())
339         if len(times) > 20:
340             #print((len(times) - 1) / (times[-1] - times[0]))
341             times.pop(0)
342
343     m.add_emg_handler(proc_emg)
344     m.connect()
345
346     try:
347         while True:
348             m.run()
349
350             if HAVE_PYGAME:
351                 for ev in pygame.event.get():
352                     if ev.type == QUIT or (ev.type == KEYDOWN and ev.unicode == 'q'):
353                         raise KeyboardInterrupt()
```

```
                    elif ev.type == KEYDOWN:
                        if K_1 ≤ ev.key ≤ K_3:
                            m.vibrate(ev.key − K_0)
                        if K_KP1 ≤ ev.key ≤ K_KP3:
                            m.vibrate(ev.key − K_KP0)

    except KeyboardInterrupt:
        pass
    finally:
        m.disconnect()
        print()
```

Matlab Test Script

```
%% RUN ONCE AT BEGINNING TO SETUP CONNECTION
clear all
clc

% set up arduino connection
comPort = '/dev/tty.usbmodem1411';
arduino=serial(comPort,'BaudRate',9600);

% connect to arduino
try
    fopen(arduino);
catch ME,
    disp(ME.message)
    delete(arduino);
    error(['Could not open port: ' comPort]);
end

%% RUN THIS PART FOR THE MATLAB GUI

% it takes several seconds before any operation could be attempted
fprintf(1,'Attempting connection .');
for i=1:2,
    pause(0.5);
    fprintf(1,'.');
end
fprintf(1,'\n');

% flush serial buffer before sending anything
val=-1;
if arduino.BytesAvailable > 0,
    val=fread(arduino,arduino.BytesAvailable);
end

% set command variables
mode = 1;
angle = 40;
speed = 10; % larger -> slower

% send commands to arduino
fwrite(arduino, mode, 'int8');
fwrite(arduino, angle, 'int8'); % limited to 128
fwrite(arduino, speed, 'int8');

%% RUN ONCE AT END TO CLOSE

fclose(arduino);
delete(instrfind('Type', 'serial'));
```

Arduino Test Script

```
#include <Servo.h>

int angle = 45;
```

```
int mode = 0;
int servo_speed = 15;
Servo myservoL;
Servo myservoR;

void setup() {
  Serial.begin(9600);
  myservoL.attach(9, 544, 3550);
  myservoR.attach(8, 544, 3550);
}

void loop() {
  // if there is data to read
  if (Serial.available() > 2) {
    mode = Serial.read(); // read data
    angle = Serial.read(); // read data
    servo_speed = Serial.read(); // read data
    switch(mode){
      case 1:
        test1(angle, servo_speed);
        break;
      case 2:
        test2(angle, servo_speed);
        break;
      case 3:
        test3(angle, servo_speed);
        break;
      default : /* Optional */
      ;
    }
  }
}

// Set the servo to the target angle
// TODO: set up if curr angle == last angle stop ... to reduce holding buzz.
void drive(Servo curr_servo, int target_pos, int s_speed) {
  int curr_pos = curr_servo.read();
  if (curr_pos < target_pos) {
    for (; curr_pos < target_pos;) {
      curr_servo.write(++curr_pos);
      delay(s_speed);
    }
  } else {
      for (; curr_pos > target_pos;) {
      curr_servo.write(--curr_pos);
      delay(s_speed);
    }
  }
}

void test1(int ang, int s_speed) {
//  myservoL.write(90);
  drive(myservoL, ang, s_speed);
}

void test2(int ang, int s_speed) {
  myservoL.write(90);
}

void test3(int ang, int s_speed) {
  myservoL.write(180);
}
```