```
// Servo Control - BioAmp EXG Pill
// https://github.com/upsidedownlabs/BioAmp-EXG-Pill

// Upside Down Labs invests time and resources providing this open source code,
// please support Upside Down Labs and open-source hardware by purchasing
// products from Upside Down Labs!

// Copyright (c) 2021 Upside Down Labs - contact@upsidedownlabs.tech

// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:

// The above copyright notice and this permission notice shall be included in all
// copies or substantial portions of the Software.

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#if defined(ESP32)
  #include <ESP32Servo.h>
#else
  #include <Servo.h>
#endif

#define SAMPLE_RATE 500
#define BAUD_RATE 115200
#define INPUT_PIN A0
#define BUFFER_SIZE 500
#define SERVO_PIN 10
#define EMG_MIN 2
#define EMG_MAX 10

int circular_buffer[BUFFER_SIZE];
int data_index, sum;

Servo servo;

void setup() {
        // Serial connection begin
        Serial.begin(BAUD_RATE);
  // Attach servo
```

```
  servo.attach(SERVO_PIN);
}

void loop() {
        // Calculate elapsed time
        static unsigned long past = 0;
        unsigned long present = micros();
        unsigned long interval = present - past;
        past = present;

        // Run timer
        static long timer = 0;
        timer -= interval;

        // Sample and get envelop
        if(timer < 0) {
                timer += 1000000 / SAMPLE_RATE;
                int sensor_value = analogRead(INPUT_PIN);
                int signal = EMGFilter(sensor_value);
                int envelop = getEnvelop(abs(signal));
    int servo_position = map(envelop, EMG_MIN, EMG_MAX, 0, 180);
    servo.write(servo_position);
                Serial.print(signal);
                Serial.print(",");
                Serial.println(servo_position);
        }
}

// Envelop detection algorithm
int getEnvelop(int abs_emg){
        sum -= circular_buffer[data_index];
        sum += abs_emg;
        circular_buffer[data_index] = abs_emg;
        data_index = (data_index + 1) % BUFFER_SIZE;
        return (sum/BUFFER_SIZE) * 2;
}

// Band-Pass Butterworth IIR digital filter, generated using filter_gen.py.
// Sampling rate: 500.0 Hz, frequency: [74.5, 149.5] Hz.
// Filter is order 4, implemented as second-order sections (biquads).
// Reference:
// https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html
// https://courses.ideate.cmu.edu/16-223/f2020/Arduino/FilterDemos/filter_gen.py
float EMGFilter(float input)
{
  float output = input;
  {
    static float z1, z2; // filter section state
    float x = output - 0.05159732*z1 - 0.36347401*z2;
    output = 0.01856301*x + 0.03712602*z1 + 0.01856301*z2;
```

```
    z2 = z1;
    z1 = x;
  }
  {
    static float z1, z2; // filter section state
    float x = output - -0.53945795*z1 - 0.39764934*z2;
    output = 1.00000000*x + -2.00000000*z1 + 1.00000000*z2;
    z2 = z1;
    z1 = x;
  }
  {
    static float z1, z2; // filter section state
    float x = output - 0.47319594*z1 - 0.70744137*z2;
    output = 1.00000000*x + 2.00000000*z1 + 1.00000000*z2;
    z2 = z1;
    z1 = x;
  }
  {
    static float z1, z2; // filter section state
    float x = output - -1.00211112*z1 - 0.74520226*z2;
    output = 1.00000000*x + -2.00000000*z1 + 1.00000000*z2;
    z2 = z1;
    z1 = x;
  }
  return output;
}
```