```
// Eye Blink Detection (Experimental!) - 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.

#include <math.h>

#define SAMPLE_RATE 75
#define BAUD_RATE 115200
#define INPUT_PIN A0
#define OUTPUT_PIN 13
#define DATA_LENGTH 10

int data_index = 0;
bool peak = false;


void setup() {
        // Serial connection begin
        Serial.begin(BAUD_RATE);
        // Setup Input & Output pin
        pinMode(INPUT_PIN, INPUT);
        pinMode(OUTPUT_PIN, OUTPUT);
}

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
        if(timer < 0){
                timer += 1000000 / SAMPLE_RATE;
        // Sample and Nomalize input data (-1 to 1)
                float sensor_value = analogRead(INPUT_PIN);
                float signal = EOGFilter(sensor_value)/512;
        // Get peak
        peak = Getpeak(signal);
        // Print sensor_value and peak
        Serial.print(signal);
        Serial.print(",");
        Serial.println(peak);
        // Blink LED on peak
        digitalWrite(OUTPUT_PIN, peak);
        }
}

bool Getpeak(float new_sample) {
        // Buffers for data, mean, and standard deviation
        static float data_buffer[DATA_LENGTH];
        static float mean_buffer[DATA_LENGTH];
        static float standard_deviation_buffer[DATA_LENGTH];

        // Check for peak
        if (new_sample - mean_buffer[data_index] > (DATA_LENGTH*1.2) *
standard_deviation_buffer[data_index]) {
                data_buffer[data_index] = new_sample + data_buffer[data_index];
                peak = true;
        } else {
                data_buffer[data_index] = new_sample;
                peak = false;
        }

        // Calculate mean
        float sum = 0.0, mean, standard_deviation = 0.0;
        for (int i = 0; i < DATA_LENGTH; ++i){
                sum += data_buffer[(data_index + i) % DATA_LENGTH];
        }
        mean = sum/DATA_LENGTH;

        // Calculate standard deviation
        for (int i = 0; i < DATA_LENGTH; ++i){
```

```
                standard_deviation += pow(data_buffer[(i) % DATA_LENGTH] - mean, 
2);
        }

        // Update mean buffer
        mean_buffer[data_index] = mean;

        // Update standard deviation buffer
        standard_deviation_buffer[data_index] = 
sqrt(standard_deviation/DATA_LENGTH);

        // Update data_index
        data_index = (data_index+1)%DATA_LENGTH;

        // Return peak
        return peak;
}

// Band-Pass Butterworth IIR digital filter, generated using filter_gen.py.
// Sampling rate: 75.0 Hz, frequency: [0.5, 19.5] 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 EOGFilter(float input)
{
  float output = input;
  {
    static float z1, z2; // filter section state
    float x = output - 0.02977423*z1 - 0.04296318*z2;
    output = 0.09797471*x + 0.19594942*z1 + 0.09797471*z2;
    z2 = z1;
    z1 = x;
  }
  {
    static float z1, z2; // filter section state
    float x = output - 0.08383952*z1 - 0.46067709*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 - -1.92167271*z1 - 0.92347975*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 - -1.96758891*z1 - 0.96933514*z2;
    output = 1.00000000*x + -2.00000000*z1 + 1.00000000*z2;
    z2 = z1;
    z1 = x;
  }
  return output;
}
```