```
//created by Stefan Ivanov
//RWTH-Aachen University
//i10 - Media Computing Group
//
//for contact with me send your emails to:
//stefan.ivanov@rwth-aachen.de
//
//The following code is used to filter a data flow
//it is based on the C++ code here:
//http://www.lifl.fr/~casiez/1euro/OneEuroFilter.cc
//
//  LowPassFilter.h
//
//  Created by Stefan Ivanov on 5/18/12.
//  Copyright (c) 2012 __RWTH-Aachen__. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface LowPassFilter : NSObject{

    double y;
    double a;
    double s;
    Boolean initialized;

}

@property (assign) double y;
@property (assign) double a;
@property (assign) double s;
@property (assign) Boolean initialized;

- (void)setAlpha:(double)alpha;
- (id)initWithAlpha:(double)alpha initval:(double)initval;
+ (id)lowPassFilterWithAlpha:(double)alpha
initval:(double)initval;
- (double)filterValue:(double)value;
- (double)filterValue:(double)value withAlpha:(double)alpha;
- (BOOL)hasLastRawValue;
- (double)lastRawValue;

@end


//
//  LowPassFilter.m
//
//  Created by Stefan Ivanov on 5/18/12.
//  Copyright (c) 2012 __RWTH-Aachen__. All rights reserved.
```

```
//

#import "LowPassFilter.h"

@implementation LowPassFilter
@synthesize y;
@synthesize a;
@synthesize s;
@synthesize initialized;

- (void)setAlpha:(double)alpha
{
    if ((alpha <= 0.0) || (alpha > 1.0)) {
        NSLog(@"Alpha should be in range [0.0..1.0]!
alpha=%lf",alpha);
    } else {
        self.a = alpha;
    }
}

- (id)initWithAlpha:(double)alpha initval:(double)initval
{
    // Call appropriate super class initializer
    if (self = [super init]) {
        // Initialize instance
        self.y = self.s = initval;
        [self setAlpha:alpha];
        self.initialized = NO;
    }
    return self;
}

+ (id)lowPassFilterWithAlpha:(double)alpha
initval:(double)initval
{
    return [[self alloc] initWithAlpha:alpha initval:initval];
}

- (double)filterValue:(double)value
{
    double result;
    if (self.initialized) {
        result = self.a * value + (1.0 - self.a) * self.s;
    }else{
        result = value;
        self.initialized = YES;
    }
    self.y = value;
    self.s = result;
    return result;
}

- (double)filterValue:(double)value withAlpha:(double)alpha
{
    [self setAlpha:alpha];
    return [self filterValue:value];
}

- (BOOL)hasLastRawValue
{
    return self.initialized;
}

- (double)lastRawValue
{
    return self.y;
}

@end


//
//  OneEuroFilter.h
//
//  Created by Stefan Ivanov on 5/18/12.
//  Copyright (c) 2012 __RWTH-Aachen__. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "LowPassFilter.h"

@interface OneEuroFilter : NSObject{

    double freq;
    double mincutoff;
    double beta;
    double dcutoff;

    LowPassFilter *x;
    LowPassFilter *dx;

    NSDate *lasttime;

}

@property (assign) double freq;
@property (assign) double mincutoff;
@property (assign) double beta;
@property (assign) double dcutoff;
```

```
@property (retain) LowPassFilter *x;
@property (retain) LowPassFilter *dx;

@property (retain) NSDate *lasttime;

- (double)alpha:(double)cutoff;
- (void)setFrequency:(double)f;
- (void)setMinCutoff:(double)mc;
- (void)setDerivativeCutoff:(double)dc;

- (id)initWithFrequency:(double)f mincutoff:(double)m
beta:(double)b dcutoff:(double)d;
+ (id)oneEuroFilterWithFrequency:(double)f mincutoff:(double)m
beta:(double)b dcutoff:(double)d;
- (double) filterValue:(double)value;

@end


//
//  OneEuroFilter.m
//
//  Created by Stefan Ivanov on 5/18/12.
//  Copyright (c) 2012 __RWTH-Aachen__. All rights reserved.
//

#import "OneEuroFilter.h"

@implementation OneEuroFilter
@synthesize freq;
@synthesize mincutoff;
@synthesize beta;
@synthesize dcutoff;
@synthesize x, dx;
@synthesize lasttime;

- (double) alpha:(double)cutoff
{
    double te = 1.0 / self.freq;
    double tau = 1.0 / (2 * M_PI *cutoff);
    return 1.0 / (1.0 + tau / te);
}

- (void) setFrequency:(double)f
{
    if (f < 0) {
        NSLog(@"Frequency should be above zero! frequency=%lf",
f);
    } else {
        self.freq = f;
    }
}

- (void)setMinCutoff:(double)mc
{
    if (mc <= 0) {
        NSLog(@"Min cutoff should be above zero! min cutoff=%lf",
mc);
    } else {
        self.mincutoff = mc;
    }
}

- (void)setDerivativeCutoff:(double)dc
{
    if (dc <= 0) {
        NSLog(@"D cutoff should be above zero! d cutoff=%lf",
dc);
    } else {
        self.dcutoff = dc;
    }
}

- (id)initWithFrequency:(double)f mincutoff:(double)m
beta:(double)b dcutoff:(double)d
{
    // Call appropriate super class initializer
    if (self = [super init]) {
        // Initialize instance
        [self setFrequency:f];
        [self setMinCutoff:m];
        [self setBeta:b];
        [self setDerivativeCutoff:d];
        self.x = [LowPassFilter lowPassFilterWithAlpha:[self
alpha:m] initval:0.0];
        self.dx = [LowPassFilter lowPassFilterWithAlpha:[self
alpha:d] initval:0.0];
        self.lasttime = [NSDate date];
    }
    return self;
}

+ (id)oneEuroFilterWithFrequency:(double)f mincutoff:(double)m
beta:(double)b dcutoff:(double)d
{
    return [[self alloc] initWithFrequency:f mincutoff:m beta:b
dcutoff:d];
}

- (double) filterValue:(double)value
```

```
{
    NSDate *now = [NSDate date];
    self.freq = 1.0 / [now timeIntervalSinceDate:self.lasttime];
    self.lasttime = now;

    //estimate the current variation per second
    double dvalue = [self.x hasLastRawValue] ? (value - [self.x
lastRawValue]) * self.freq : 0.0;
    double edvalue = [self.dx filterValue:dvalue withAlpha:[self
alpha:self.dcutoff]];

    //use it to update the cutoff frequency
    double cutoff = self.mincutoff + self.beta * fabs(edvalue);

    //filter the given value
    return [self.x filterValue:value withAlpha:[self
alpha:cutoff]];
}


@end




//FOR USING THE FILTER DO THE FOLLOWING

//initialize the filters in the beginning
//in the init method of the class you use for reading values
//or in the method opening connection to your board (Arduino in
//my case)
    self.oneEuroFilterX = [OneEuroFilter
oneEuroFilterWithFrequency:120 mincutoff:1.0 beta:1.0
dcutoff:1.0];
    self.oneEuroFilterY = [OneEuroFilter
oneEuroFilterWithFrequency:120 mincutoff:1.0 beta:1.0
dcutoff:1.0];


//filter the X and Y values with their respective filters every
//time that you read them
    tl.Xcoord = [self.oneEuroFilterX filterValue:tl.Xcoord];
    tl.Ycoord = [self.oneEuroFilterY filterValue:tl.Ycoord];
```