```
//created by Stefan Ivanov
//RWTH-Aachen University
//i10 - Media Computing Group

//for contacts email me at stefan.ivanov@rwth-aachen.de

//Implementation of 1 euro filter for the processing environment

//The following code is used to filter two data flows.
//It is based on the c++ code here: http://www.lifl.fr/~casiez/1euro/OneEuroFilter.cc
//In my implementation I use the filter for two continuosly read sets of values
//representing X coordinate and Y coordinate of a point coming on the serial port
//from an Arduino Duelmilanove board.
//Due to the fact that processing is not object-oriented there is a large number of
//variables declared. Methods also duplicate each other for X and Y values but
//since they work on different variables they have a slight difference in the name.
//I use the following naming convention for methods and variables: all that is prefixed
//with x or dx refers to the set of X coordinates that I filter and all that is
//prefixed with y or dy refers to the set of Y coordinates that I filter. Additionally
//all methods, containing LPF as second prefix are part of the low-pass filtering and
//all methods, containing OEF as second prefix are part of the one euro filtering itself

//This filter worked significantly better than the implementation of Kalman filter
//that I was using before and the code executes much faster

import processing.serial.*;

//vars used in the low-pass filtering of the incoming X coordinate
float xLPFy;
float xLPFa;
float xLPFs;
boolean xLPFinitialized;
float dxLPFy;
float dxLPFa;
float dxLPFs;
boolean dxLPFinitialized;

//vars used in the low-pass filtering of the incoming Y coordinate
float yLPFy;
float yLPFa;
float yLPFs;
boolean yLPFinitialized;
```

```
float dyLPFy;
float dyLPFa;
float dyLPFs;
boolean dyLPFinitialized;

//vars used for the 1 euro filter parameters for the X coordinate
float xOEFfreq;
float xOEFmincutoff;
float xOEFbeta;
float xOEFdcutoff;
float xOEFoldTime = 0.0;
float xOEFnewTime = 0.0;

//vars used for the 1 euro filter parameters for the Y coordinate
float yOEFfreq;
float yOEFmincutoff;
float yOEFbeta;
float yOEFdcutoff;
float yOEFoldTime = 0.0;
float yOEFnewTime = 0.0;

Serial myPort;

void setup()
{
 //other setup code here

 //set up the parameters of the 1 euro filter
 xOneEuroFilter(120.0,1.0,1.0,1.0);
 yOneEuroFilter(120.0,1.0,1.0,1.0);

}

//low-pass filtering methods

void xLPFsetAlfa(float alfa)
{
 if(alfa < 0.0){
   xLPFa = 0.0;
 }else if(alfa > 1.0){
   xLPFa = 1.0;
 }else {
```

```
    xLPFa = alfa;
  }
}

void yLPFsetAlfa(float alfa)
{
  if(alfa < 0.0){
    yLPFa = 0.0;
  }else if(alfa > 1.0){
    yLPFa = 1.0;
  }else {
    yLPFa = alfa;
  }
}

void dxLPFsetAlfa(float alfa)
{
  if(alfa < 0.0){
    dxLPFa = 0.0;
  }else if(alfa > 1.0){
    dxLPFa = 1.0;
  }else {
    dxLPFa = alfa;
  }
}

void dyLPFsetAlfa(float alfa)
{
  if(alfa < 0.0){
    dyLPFa = 0.0;
  }else if(alfa > 1.0){
    dyLPFa = 1.0;
  }else {
    dyLPFa = alfa;
  }
}

void xLowPassFilter(float alfa, float initval)
{
  xLPFy = initval;
  xLPFs = initval;
  xLPFsetAlfa(alfa);
```

```
  xLPFinitialized = false;
}

void yLowPassFilter(float alfa, float initval)
{
  yLPFy = initval;
  yLPFs = initval;
  yLPFsetAlfa(alfa);
  yLPFinitialized = false;
}

void dxLowPassFilter(float alfa, float initval)
{
  dxLPFy = initval;
  dxLPFs = initval;
  dxLPFsetAlfa(alfa);
  dxLPFinitialized = false;
}

void dyLowPassFilter(float alfa, float initval)
{
  dyLPFy = initval;
  dyLPFs = initval;
  dyLPFsetAlfa(alfa);
  dyLPFinitialized = false;
}

float xLPFfilter(float value)
{
  float result;
  if(xLPFinitialized){
    result = xLPFa * value + (1.0 - xLPFa) * xLPFs;
  } else{
    result = value;
    xLPFinitialized = true;
  }
  xLPFy = value;
  xLPFs = result;
  return result;
}

float yLPFfilter(float value)
```

```
{
  float result;
  if(yLPFinitialized){
    result = yLPFa * value + (1.0 - yLPFa) * yLPFs;
  } else{
    result = value;
    yLPFinitialized = true;
  }
  yLPFy = value;
  yLPFs = result;
  return result;
}

float dxLPFfilter(float value)
{
  float result;
  if(dxLPFinitialized){
    result = dxLPFa * value + (1.0 - dxLPFa) * dxLPFs;
  } else{
    result = value;
    dxLPFinitialized = true;
  }
  dxLPFy = value;
  dxLPFs = result;
  return result;
}

float dyLPFfilter(float value)
{
  float result;
  if(dyLPFinitialized){
    result = dyLPFa * value + (1.0 - dyLPFa) * dyLPFs;
  } else{
    result = value;
    dyLPFinitialized = true;
  }
  dyLPFy = value;
  dyLPFs = result;
  return result;
}

float xLPFfilterWithAlfa(float value, float alfa)
```

```
{
  xLPFsetAlfa(alfa);
  return xLPFfilter(value);
}

float yLPFfilterWithAlfa(float value, float alfa)
{
  yLPFsetAlfa(alfa);
  return yLPFfilter(value);
}

float dxLPFfilterWithAlfa(float value, float alfa)
{
  dxLPFsetAlfa(alfa);
  return dxLPFfilter(value);
}

float dyLPFfilterWithAlfa(float value, float alfa)
{
  dyLPFsetAlfa(alfa);
  return dyLPFfilter(value);
}



//one euro filtering methods

float xOEFalfa(float cutoff, float frequency)
{
  float te = 1.0/frequency;
  float tau = 1.0/(2*PI*cutoff);
  return 1.0/(1.0+tau/te);
}

float yOEFalfa(float cutoff, float frequency)
{
  float te = 1.0/frequency;
  float tau = 1.0/(2*PI*cutoff);
  return 1.0/(1.0+tau/te);
}

void xOneEuroFilter(float freq, float mincutoff, float beta, float dcutoff)
```

```
{
  xOEFfreq = freq;
  xOEFmincutoff = mincutoff;
  xOEFbeta = beta;
  xOEFdcutoff = dcutoff;
  xLowPassFilter(xOEFalfa(mincutoff, xOEFfreq),0.0);
  dxLowPassFilter(xOEFalfa(dcutoff, xOEFfreq), 0.0);
}

void yOneEuroFilter(float freq, float mincutoff, float beta, float dcutoff)
{
  yOEFfreq = freq;
  yOEFmincutoff = mincutoff;
  yOEFbeta = beta;
  yOEFdcutoff = dcutoff;
  yLowPassFilter(yOEFalfa(mincutoff, yOEFfreq),0.0);
  dyLowPassFilter(yOEFalfa(dcutoff, yOEFfreq), 0.0);
}

float xOEFfilter(float value)
{
  xOEFoldTime = xOEFnewTime;
  xOEFnewTime = millis();
  xOEFfreq = 1.0 / ((xOEFnewTime - xOEFoldTime) * 1000);
  float dvalue;
  if(xLPFinitialized){
    dvalue = (value - xLPFy) * xOEFfreq;
  }else{
    dvalue = 0.0;
  }
  float edvalue = dxLPFfilterWithAlfa(dvalue, xOEFalfa(xOEFdcutoff, xOEFfreq));
  float cutoff = xOEFmincutoff + xOEFbeta * abs(edvalue);

  return xLPFfilterWithAlfa(value, xOEFalfa(cutoff, xOEFfreq));
}

float yOEFfilter(float value)
{
  yOEFoldTime = yOEFnewTime;
  yOEFnewTime = millis();
  yOEFfreq = 1.0 / ((yOEFnewTime - yOEFoldTime) * 1000);
  float dvalue;
```

```
  if(yLPFinitialized){
    dvalue = (value - yLPFy) * yOEFfreq;
  }else{
    dvalue = 0.0;
  }
  float edvalue = dyLPFfilterWithAlfa(dvalue, yOEFalfa(yOEFdcutoff, yOEFfreq));
  float cutoff = yOEFmincutoff + yOEFbeta * abs(edvalue);

  return yLPFfilterWithAlfa(value, yOEFalfa(cutoff, yOEFfreq));
}

void draw()
{

      //perform the 1 euro filtering

      //filteredValue = xOEFfilter(unfilteredValue);
      //filteredValue = yOEFfilter(unfilteredValue);

}
```