Processing for Visualisation

DBB170 Sensors for Physiology

Loe Feijs

2017

Today's goals

- refresh DBB100 Processing competence
- access PPG data from your Arduino
- do calculations based on heart beats
- make drawings based on heart beats
- real-time

Working with serial

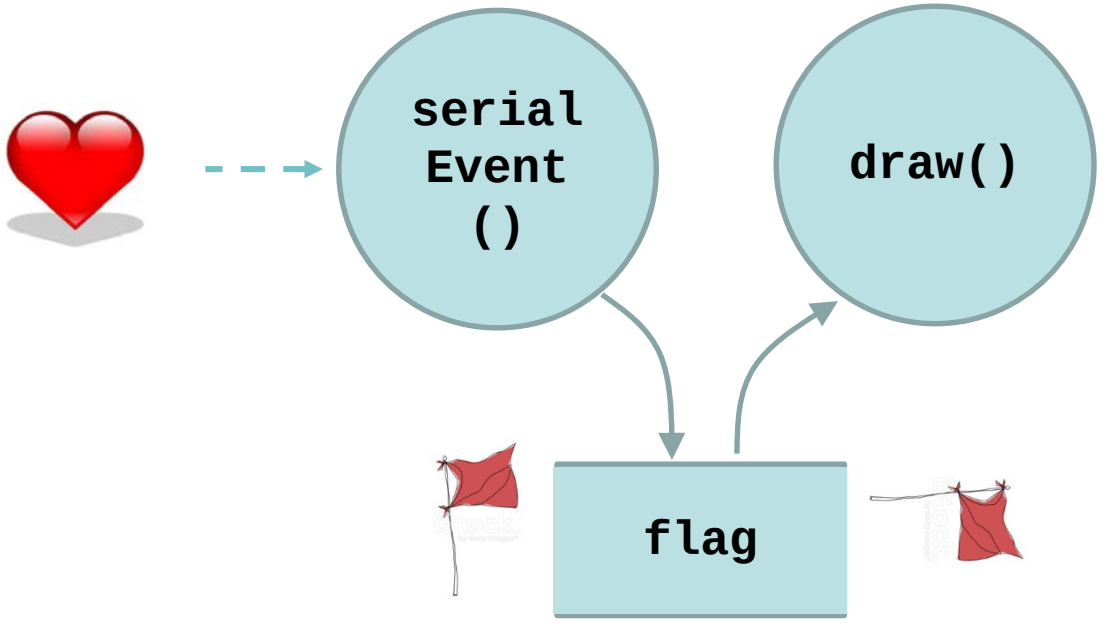
```
import processing.serial.*;

Serial myPort;
void setup() {
     //if [0] does't work, try [1] or [2]:
     myPort = new Serial(this, Serial.list()[2], 19200);
}

boolean flag = false;
void draw(){
     if (flag) {
          flag = false;
          print("3>  ");
     }
}

void serialEvent(Serial p) {
     if (p.available() > 0) {
          String val = myPort.readStringUntil('\n');
          flag = true;
     }
}
```

Working with serial
(dataflow diagram)



Working with hex

Decimal	Hexadecimal	Binary
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Dec	Hx	Oct	Char	
0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)
3	3	003	ETX	(end of text)
4	4	004	EOT	(end of transmission)
5	5	005	ENQ	(enquiry)
6	6	006	ACK	(acknowledge)
7	7	007	BEL	(bell)
8	8	010	BS	(backspace)
9	9	011	TAB	(horizontal tab)
10	A	012	LF	(NL line feed, new line)
11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)
14	E	016	SO	(shift out)
15	F	017	SI	(shift in)
16	10	020	DLE	(data link escape)
17	11	021	DC1	(device control 1)
18	12	022	DC2	(device control 2)
19	13	023	DC3	(device control 3)
20	14	024	DC4	(device control 4)
21	15	025	NAK	(negative acknowledge)
22	16	026	SYN	(synchronous idle)
23	17	027	ETB	(end of trans. block)
24	18	030	CAN	(cancel)
25	19	031	EM	(end of medium)
26	1A	032	SUB	(substitute)
27	1B	033	ESC	(escape)
28	1C	034	FS	(file separator)
29	1D	035	GS	(group separator)
30	1E	036	RS	(record separator)
31	1F	037	US	(unit separator)

Source: http://www.matrixlab-examples.com/hex-to-binary.html

http://www.asciitable.com/index/asciifull.gif

Working with hex

```
int RR = 1000;                  //beat to beat interval

void serialEvent(Serial p) {
     if (p.available() > 0) {
          String val = myPort.readStringUntil('\n');
          if (val != null){
               RR = unhex(trim(val)) / 1000;
               print("RR = " + RR + ", ");
          }
     }
}
```

This reference is for Processing 3.0+. If you have a previous version, use the reference included with your software in the Help menu. If you see any errors or have suggestions, please let us know. If you prefer a more technical reference, visit the Processing Core Javadoc and Libraries Javadoc.

Name

trim()

Examples

```
String s1 = "    Somerville MA ";
println(s1);  // Prints "    Somerville MA "
String s2 = trim(s1);
println(s2);  // Prints "Somerville MA"
```

Description

Removes whitespace characters from the beginning and end of a String. In addition to standard whitespace characters such as space, carriage return, and tab, this function also removes the Unicode "nbsp" character.

Syntax

```
trim(str)
trim(array)
```

Parameters

str	String: any string
array	String[]: a String array

Working with the screen

```
void setup() {
     size(900,900);
     background(0);
}

int RR = 1000;

void draw(){
     //fancy drawings based on RR
}

void serialEvent(Serial p) {
     if (p.available() > 0) {
          //set flag
          //fill RR

          //BUT DO NOT DRAW HERE

     }
}
```

Working with the screen

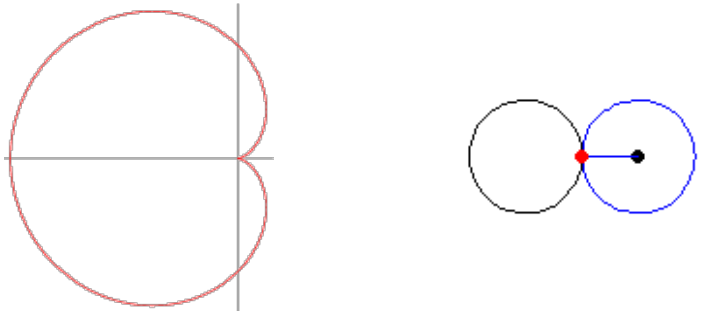
- `translate`
- `rotate`
- `pushMatrix`
- `fill`
- `stroke`
- `ellipse`
- `line`
- `rect`

Working with the screen

```
void blueEllipses(){
    //draw a bunch of blue ellipses in the centre of the window
    //have them 45 degrees rotated
    pushMatrix();
    translate(width/2,height/2);
    rotate(PI/4);
    stroke(0,0,225);
    strokeWeight(1);
    noFill();
    ellipse(0,0, 25, 50);
    ellipse(0,0, 50,100);
    ellipse(0,0,100,200);
    ellipse(0,0,200,400);
    ellipse(0,0,300,600);
    ellipse(0,0,400,800);
    popMatrix();
}
```

Working with the screen

Drawing a cardioid



The curve given by the polar equation

$$r = a\,(1 - \cos\theta),$$

The cardioid has Cartesian equation

$$\left(x^2 + y^2 + a\,x\right)^2 = a^2\left(x^2 + y^2\right),$$

and the parametric equations

$$x = a\cos t\,(1 - \cos t)$$
$$y = a\sin t\,(1 - \cos t).$$

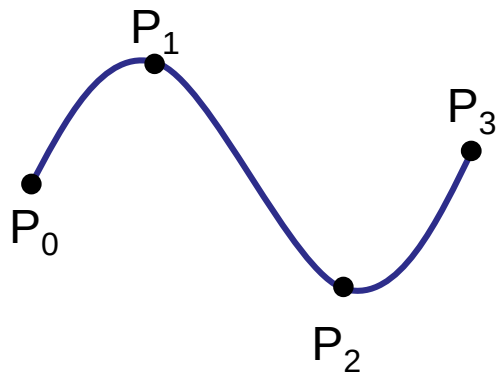
http://mathworld.wolfram.com/Cardioid.html

Drawing a cardioid

```
void cardioid(float x0, float y0, float a){
      smooth();
      beginShape();
      stroke(255,0,0);
      strokeWeight(1);
      fill(200,0,0);
      for (float theta=0; theta <= 2*PI; theta+=.1){
            float r = a * (1 - cos(theta));
            vertex(x0 + r*sin(theta),y0 - r*cos(theta) - 2*a);
      }
      endShape();
}
```

Drawing smooth curves

- four control points define one segment
- between P_1 and P_2 parameter t runs from 0 to 1
- the point P(t) is a weighted average of P_0,P_1,P_2,P_3
- $P = ½ (-t+2t^2-t^3)P_0 + ½ (2-5t^2+3t^3)P_1 + ½ (t+4t^2-3t^3)P_2 + ½(-t^2+t^3)P_3$



- the line goes *through* P_1 and P_2
- the tangent at P_1 parallels the line P_0- P_2
- the tangent at P_2 parallels the line P_1- P_3

www.mvps.org/directx/articles/catmull/

Drawing smooth curves

```
float catmullromPolynomial(float x0, float x1, float x2, float x3, float t){
      //weigthed version of x0,x1,x2,x3 using Catmull-Rom's polynomial
      return 0.5 * (-t + 2*t*t - t*t*t)  * x0
           + 0.5 * (2 - 5*t*t + 3*t*t*t) * x1
           + 0.5 * (t + 4*t*t - 3*t*t*t) * x2
           + 0.5 * (-t*t + t*t*t) * x3;
}
```

$$P = \tfrac{1}{2}(-t+2t^2-t^3)P_0 + \tfrac{1}{2}(2-5t^2+3t^3)P_1 + \tfrac{1}{2}(t+4t^2-3t^3)P_2 + \tfrac{1}{2}(-t^2+t^3)P_3$$

Drawing smooth curves

```
float[][] myPoints = new float[4][2]; //four points


void myCurve(float[][] p){
     //draw the curve as connected line segments
     //p must contains four points as x and y
     float prev_x = p[1][0];
     float prev_y = p[1][1];
     strokeWeight(.5);
     for (int i=0; i <= 100; i++){
          float t = i * 0.01;
          float x = catmullromPolynomial(p[0][0],p[1][0],p[2][0],p[3][0],t);
          float y = catmullromPolynomial(p[0][1],p[1][1],p[2][1],p[3][1],t);
          stroke(0,255,0); line(prev_x,prev_y,x,y); stroke(0);
          prev_x = x;
          prev_y = y;
     }
}
```

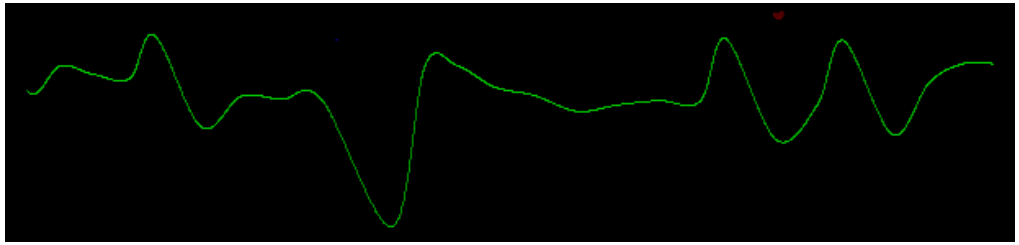
Drawing smooth curves

```
void addPoint(float newx, float newy){
      myPoints[0][0] = myPoints[1][0];
      myPoints[0][1] = myPoints[1][1];
      myPoints[1][0] = myPoints[2][0];
      myPoints[1][1] = myPoints[2][1];
      myPoints[2][0] = myPoints[3][0];
      myPoints[2][1] = myPoints[3][1];
      myPoints[3][0] = newx;
      myPoints[3][1] = newy;
}
```

```
float x = t10ms/2.5;//x scaled time
x = x % width;
float y = (2000-RR) / 10;//y scaled by 10
addPoint(x, height - y);

if (beats > 3) myCurve(myPoints);
```

Drawing smooth curves



Averaging

The simple moving average (SMA)

Intuitively, the simplest way to smooth a time series is to calculate a simple, or unweighted, moving average. This is known as using a rectangular or "boxcar" window function. The smoothed statistic s_t is then just the mean of the last k observations:

$$s_t = \frac{1}{k}\sum_{n=0}^{k-1} x_{t-n} = \frac{x_t + x_{t-1} + x_{t-2} + \cdots + x_{t-k+1}}{k} = s_{t-1} + \frac{x_t - x_{t-k}}{k},$$

where the choice of an integer $k > 1$ is arbitrary. A small value of k will have less of a smoothing effect and be more responsive to recent changes in the data, while a larger k will have a greater smoothing effect, and produce a more pronounced lag

en.wikipedia.org/wiki/Exponential_smoothing

Averaging

(simple moving average, the Wikipedia way)

```
int[] x = {1,2,3,4,5,6,7,8,9,10};

int k = 2;//size of window
int t = 9; //last data item
int s = 0; //sum

for (int n = 0; n < k; n++)
      s = s + x[t - n];

println(s);
```

en.wikipedia.org/wiki/Exponential_smoothing

Averaging

(simple moving average, real-time)

```
int WINDOWSIZE = 5;
int[] RRwindow = new int[WINDOWSIZE]; //oldest at [0] latest at end

void addRR(int r){
      for (int i = 0; i < WINDOWSIZE-1; i++)
            RRwindow[i] = RRwindow[i + 1];
      RRwindow[WINDOWSIZE - 1] = r;
}    //PS smarter way: use circular


float RRavg(){
      float sum = 0;
      for (int i = 0; i < WINDOWSIZE; i++)
            sum = sum + RRwindow[i];
      return sum / WINDOWSIZE;
}
```

Averaging
(simple moving average, real-time)

```
int beats = 0;
void serialEvent(Serial p) {
     if (p.available() > 0) {
          String val = myPort.readStringUntil('\n');
          if (val != null){
               print("3> ");
               int RR = unhex(trim(val)) / 1000;
               print("RR = " + RR + ", ");
               addRR(RR);
               if (++beats >= WINDOWSIZE)
                    println("HR = " + ceil(60.0/(RRavg()/1000)) + ", ");
          }
     }
}
```

en.wikipedia.org/wiki/Exponential_smoothing

Averaging
(exponential smoothing)

The simplest form of exponential smoothing is given by the formula:

$$s_t = \alpha \cdot x_t + (1 - \alpha) \cdot s_{t-1} \, .$$

where α is the *smoothing factor*, and $0 < \alpha < 1$. In other words, the smoothed statistic s_t is a simple weighted average of the current observation x_t and the previous smoothed statistic s_{t-1}.

en.wikipedia.org/wiki/Exponential_smoothing

Averaging
(exponential smoothing)

```
float RRavg = 1000;
```

initially

each beat

```
float tau = 5;//beats
float alpha = 1/tau;//weighing factor for smoothing

RRavg = alpha * RR + (1 - alpha) * RRavg;//exponential smoothing
```

Real time Poincaré plots

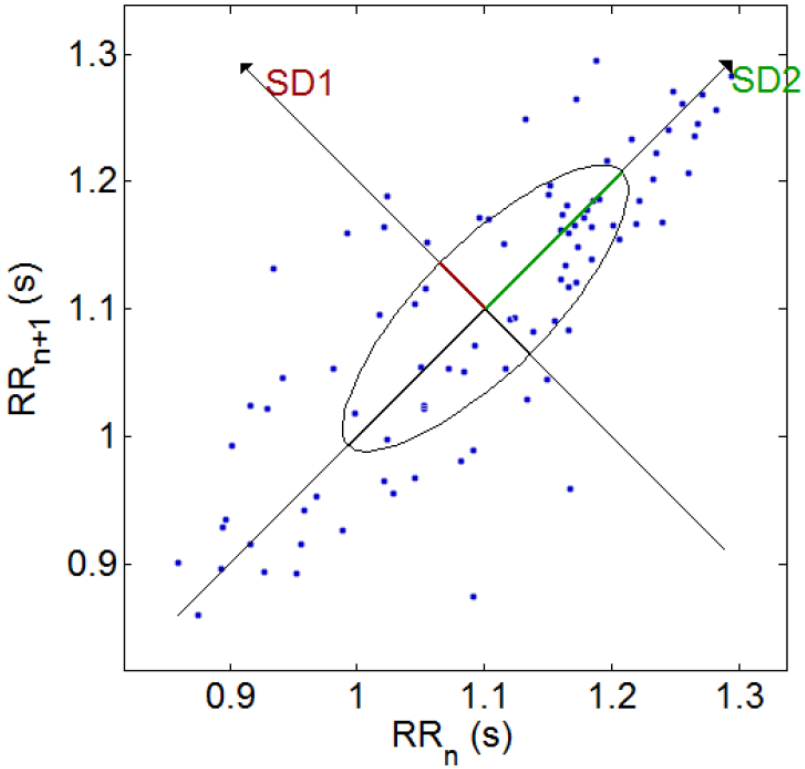


Fig. 9. Traditional usage of Poincaré Plot for off-line analysis of Heart Rate Variability. The graph is generated by Kubios Version 2.0. The RR interval (= beat to beat interval) is plotted vertically, against the previous RR interval, which defines the position on the horizontal axis.

Real time Poincaré plots

global variable

global variable

in **serialEvent()**

```
RRprev = RR;
RR = unhex(trim(val)) / 1000;
```

in **draw()**, each beat

```
RRavg = alpha * RR + (1 - alpha) * RRavg;

//poincaré.classic:
stroke(255,0,0); fill(255,0,0);
ellipse(width/2 + (RRprev - RRavg),height/2 - (RR - RRavg),5,5);
```

Your homework

- read PPG data from your Arduino
- make them visible in Processing
- create something useful or novel
- make a proper description of it
- don’t forget to add references
- work in same groups of 2