Physical Computing based on Open Software and Hardware Platforms

Heart Beat Project

Group 12
Jan Tomsa
Mariantonietta Vigorito

Madrid, 18/11/2016

Table of contents

Focus

The heart rate is a very important parameter of human body. This document describes our project for measuring the heartbeat by light.
The system uses a LED for illuminating a finger, the light is detected by a LDR (light depending resistor) after crossing the finger. The system uses voltage divider to read the value of the LDR, the signal is converted by the ADC (analog to digital converter) of the Arduino. Thus the variations of the transmitted light (caused by different blood flows) are measured and recorded. After that, the signal is smoothed (average and running average) and the samples are sent to the PC by the USB-serial port of the Arduino. When received, a presentation program, developed in PROCESSING, shows a live chart of measured values. In this graph, the period of the signal, i.e. heartbeats period, can be calculated or estimated visually.

1.1 Material

The system adopts the following materials:

- One Arduino Nano;
- Two LED;
- One LDR;
- Small breadboard;
- Mini USB cable;
- Three resistor;
- Several connecting cables.

The heart beat project of reference is shown in the
Figure 1.1.1.

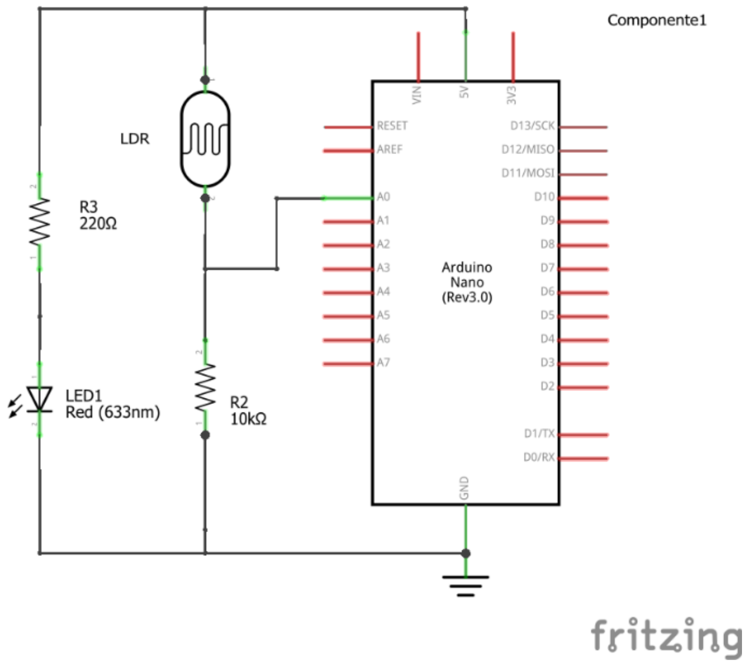
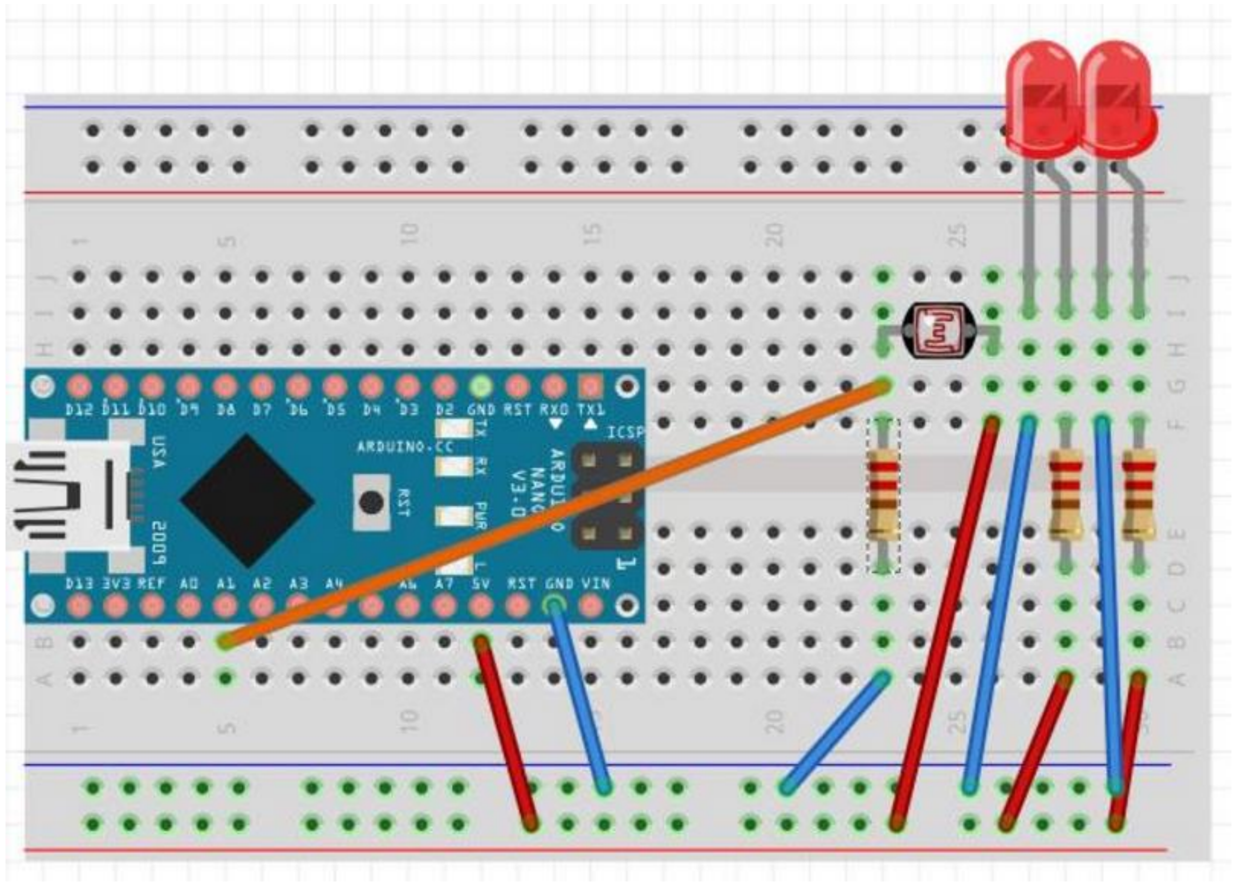


Figure 1.1.1 - Heart beat project of reference.

The LED has to be very powerful in order to obtain a realistic and accurate reproduction of the heart beat. For this reason an external powerful led, different from the project of reference, or more LEDs as the shows in Figure 1.1.2 can be used. This figure explains all necessary connections.



D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
D13 3V3 REF A0 A1 A2 A3 A4 A6 A7 5V RST GND VIN
ARDUINO.CC
ARDUINO NANO V3.0
ICSP
USA
2009
RST

Figure 1.1.2 – Heart beat project: breadboard layout.

1.2 Arduino section

The Arduino code is simple. The only task of the Arduino is to read values from it's analog input. The signal from the LDR is converted to digital values within the range of 0-1023.
The logic of the code is as follows. The sample rate is 50Hz (20ms between value readings). Once the 20ms interval passes, the Arduino reads a burst of 10 values and calculates the average. This value is then used to update the running average. (RA = 0.93*RA = 0.03*(average_of_ten_samples)). This newly calculated value is sent via the serial port to the computer. The precision of the number is set to 5 decimal numbers. For proper separation of the numbers sent via the serial port, each number is wrapped inside a pair of parenthesis (<, >). It helps parsing the output in the PROCESSING language.

1.3 Processing section

The code for the Processing is more complicated than the Arduino part. In the begining a few variables need to be initialized. The most important thing is to open Serial port to read values from the Arduino. The function for reading values from the serial port consists of a while loop that runs until we get a correct value. First we read everything up to the opening bracket (<), then we read the rest of the number (read to the closing bracket (>)) and if that is successful, the closing bracket is removed and the obtained string is converted to float. The important part is to call the clear() function after the successful reading of a value. It clears the queue in the Serial port so next time we read values we get the most recent one.

There is a list of float numbers that serves as a buffer for values. This buffer is periodically updated via the function "update_values". This function reads value from the serial port and appends this value to the end of the list. If the specified length of the buffer is exceeded, the oldest value in the buffer is discarded.

This program is able to display all the values in real-time. Every time a new value is inserted into the buffer, all point on the screen are redrawn and shifted accordingly to the time shift. New values emerge on the right side of the screen while the old disappear on the left side.

For better convenience, this program is equipped with auto-scale functionality. What it means is that the graph uses all available space to show the current range of values. The scale is not fixed, it changes according to the current data range.

The graph has two axes, the vertical axis shows data values, it’s labels change accordingly to the scale. The horizontal axis shows time, every line represents one second (60FPS, every value takes one pixel, thus 60pixels equals one second).

1.4 Results

The project implementation starts by designing the breadboard layout and by connecting all necessary parts. The layout is reported in Figure 1.4.1 and 1.1.2.

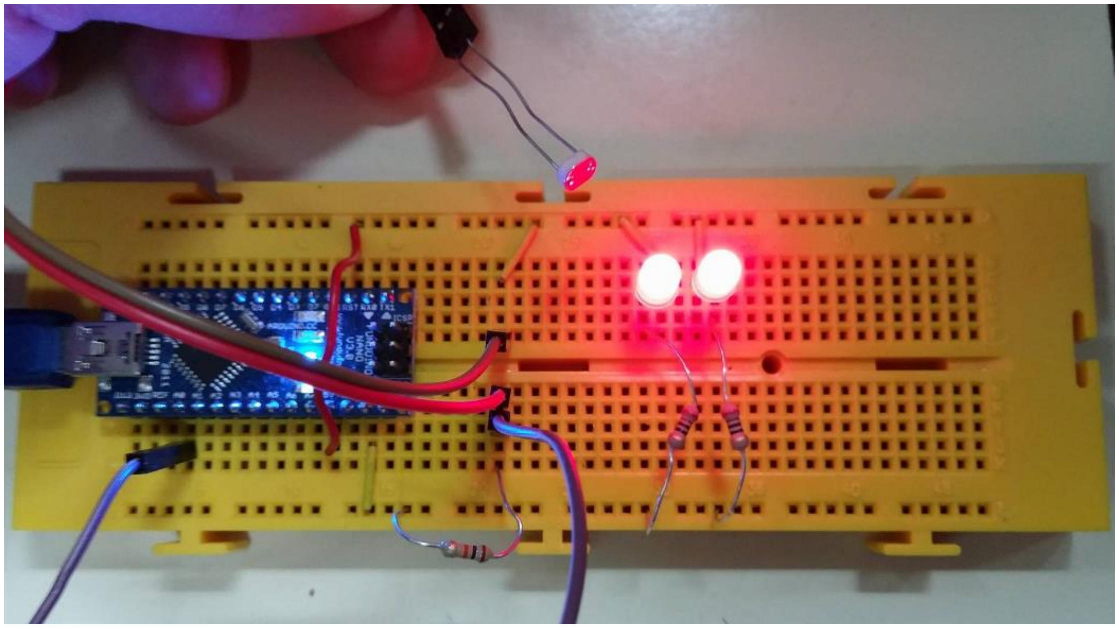


Figure 1.4.1 - Heart beat project development.

The next step is to program both the Arduino and the presentation application. After that we can use Arduino to obtain data from the sensor and display them in the presentation application.
The Figure 1.4.2 displays results from the heart rate measurements. Repeating peaks with a period of roughly one second can be clearly visible. With high probability, it corresponds to real heart beat.

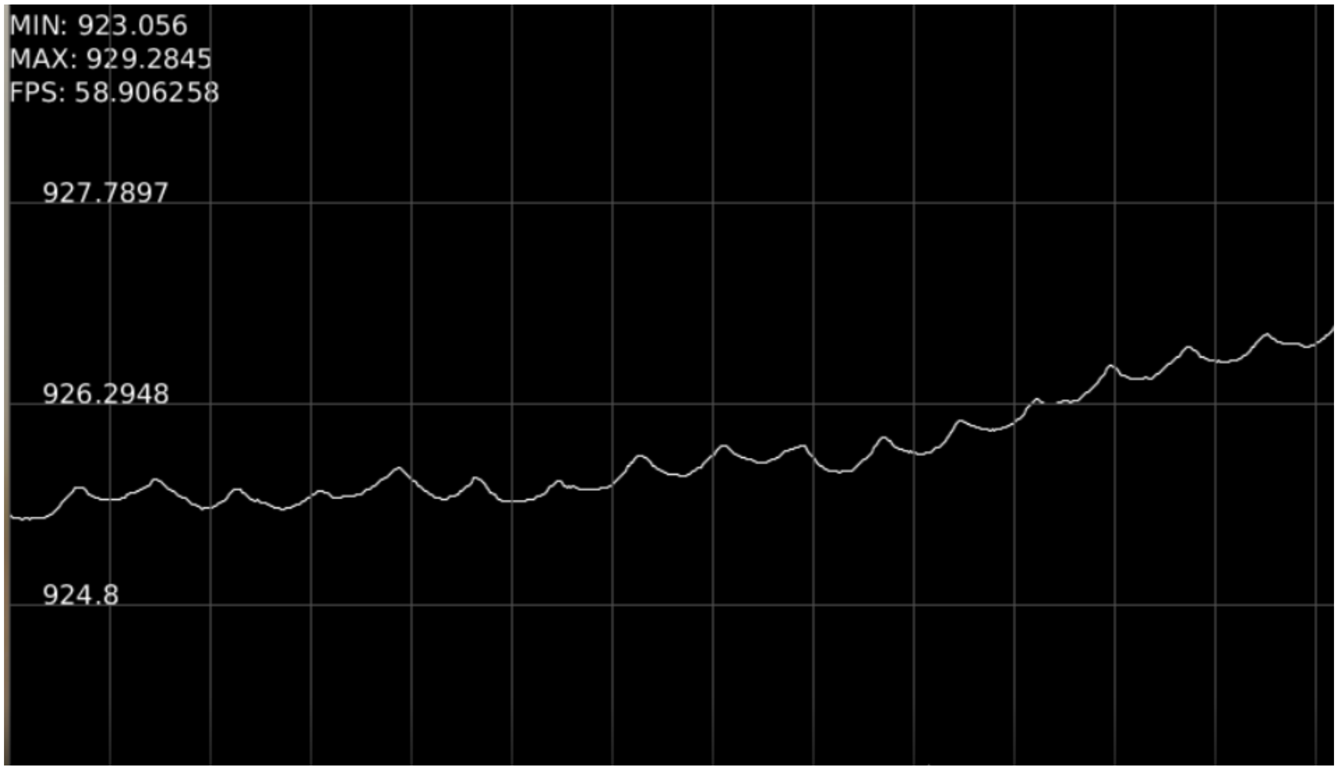


Figure 1.4.2 – Heart beat response.

1.5 Proposal

To further improve the functionality of the experiment, a peak detection algorithm could be applied on the data in the buffer. With it’s help one could get the time difference between peaks - the heart rate could be obtained.

Appendix A

The Arduino code is reported below.

```
int PIN_BLOOD = A0;
int sample_rate = 20;
double RA = 0;

unsigned long cmills, pmills;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pmills = millis();
  cmills = millis();
}

double take_samples(int count){
  double avg = 0;
  for (int i=0; i<count; i++){
    avg += analogRead(PIN_BLOOD);
  }
  avg /= count;
  return avg;
}

void loop() {
  // put your main code here, to run repeatedly:

  cmills = millis();
  if (cmills - pmills >= sample_rate){
     RA = 0.97*RA + 0.03*take_samples(10);
     Serial.print('<');
     Serial.print(RA,5);
     Serial.print('>');
     Serial.println("");
     Serial.flush();

  }

}
```

Appendix B

The Processing code is below.

```
import processing.serial.*;
import java.util.*;

Serial myPort;
PFont f;
List<Float> values;

int max_size = 800;
int graph_height = 500;
float min, max;
String s;

void setup(){
  size(800, 600);
  myPort = new Serial(this, Serial.list()[32],  9600);
  values = new ArrayList<Float>();
  f = createFont("Arial",16,true); // STEP 2 Create Font
}

void update_values(){
 float value = read_value();
  values.add(value);
  if (values.size() > max_size){
    values.remove(0);
  }
  //help_arr = toIntArray(values);
}


float get_max(){
  float max = 0;
  for (int i=0; i< values.size(); i++){
     if (values.get(i)>max){
       max = values.get(i);
     }
  }
  return max;
}

float get_min(){
  float min = 1023;
  for (int i=0; i< values.size(); i++){
     if (values.get(i)<min){
       min = values.get(i);
     }
  }
  return min;
```

```
}

void draw_points(){
   int point_count= values.size();
   stroke(255);
   min = get_min();
   max = get_max();
   //println("min:max " + min+ " " +max);

   float x1, x2, y1, y2;
   for (int i=point_count - 2; i>=0; i--){
     //point(width-point_count+i, graph_height-map(values.get(i), min-
10,max+10, 0,graph_height ));
     x1 = width-point_count+i;
     x2 = width-point_count+i+1;
     y1 = graph_height-map(values.get(i), min-10,max+10, 0,graph_height );
     y2 = graph_height-map(values.get(i+1), min-10,max+10, 0,graph_height
);
     line(x1, y1, x2, y2);
   }
}

void draw_labels(){
   textFont(f,16);                  // STEP 3 Specify font to be used
  fill(255);                        // STEP 4 Specify font color
  text("MIN: " + min, 0, 20);   // STEP 5 Display Text
  text("MAX: " + max, 0, 40);
  text("FPS: " + frameRate , 0, 60);

  stroke(80);
  line(0, height-15, width, height-15);
  for (int i=0; i<width; i+=60){
    line(i, 0, i, height);
  }

  for (int i=0; i<height-100; i+=120){
    text(""+map(i, 0, height-100, max,min), 20,i);
    line(0, i, width, i);
  }
}

void draw(){
  background(0);
  update_values();
  draw_points();
  draw_labels();
```

```
   //println(frameRate);
}

float read_value(){
  s = null;
  while (s == null){
    myPort.readStringUntil('<');
    s = myPort.readStringUntil('>');
    if  (s != null && s.length() > 0 && s.charAt(s.length()-1) == '>'){
        s = s.substring(0, s.length()-1);
    }
  }
  myPort.clear();
  return float(s);
}
```