Temperature Control Project

Group 5: Alexander van der Zee, Christian Weiser

1 SCOPE

The aim of this project is to control the temperature of a metal block with the Arduino and to display the current values in Processing on a computer. Moreover, it is desired to set the target temperature from the computer. Therefore, a GUI is implemented.

The Project consists of three main components, which are Hardware, Arduino Programming and Processing Programming.

The system shall measure the temperature of the metal block and ambient air. The Arduino has a controller implemented which sets the heat/cool input of a Peltier element mounted to the metal block. This is done in a way, so the block reaches the temperature commanded by the processing GUI. The current and a block temperature values and the current ambient temperature are shown in the GUI, moreover new temperature commands can be sent to the Arduino.

In the following those three components are described and in the last part of the report the system as a whole is considered and the outcome will be discussed.

2 HARDWARE

2.1 COMPONENTS USED

First, all components used in the system are listed:

- Arduino Nano
- Breadboard
- L298 Bridge
- DC Source
- 2 LEs (red, blue)
- 2 resistors (220 Ohms)
- 2 temperature sensors (LM35)

- Peltier Element
- Metal Block
- Wiring

2.2 Breadboard Layout

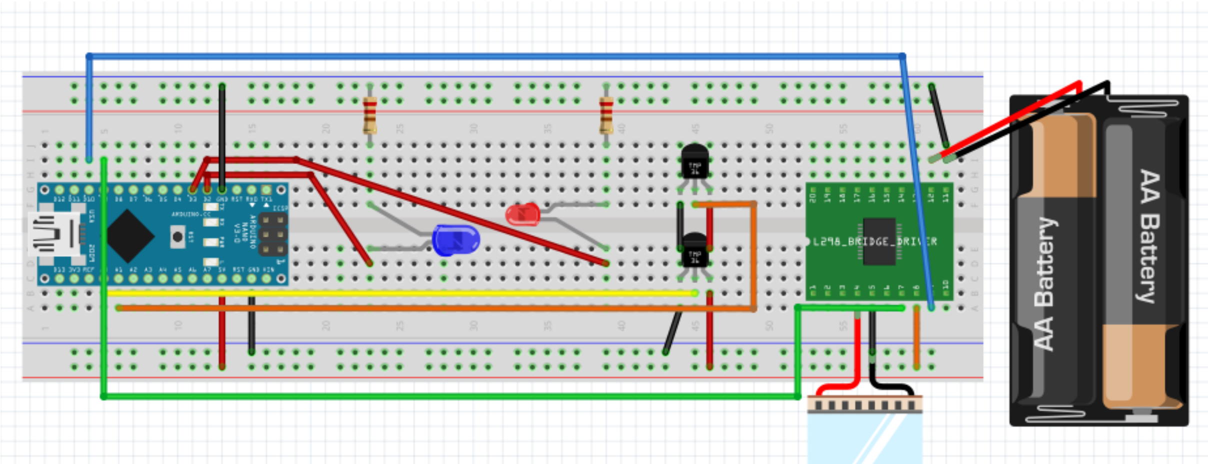


Figure 2-1: Board Setup

Figure 2-1 shows the current Breadboard Setup. The battery pack on the right is symbolic for a DC Source which is needed to power the Peltier Element. In order to switch the Peltier element the right way, a L298 DC motor driver is used.

3 Arduino Code

The Purpose of the Code implemented on the Arduino is to attain and hold the current temperature value by setting the right. Moreover, it shall indicate the status of the Peltier element with the LEDs (Blue: cooling, red: heating).

The Code also watches for commands via the serial port and sends as a response the current temperature values for the metal block and ambient temperature.

The temperature control on the Arduino is implemented via a PI controller function with the constants Kp = 25 [1/K] ad Ki = 0.1 [1/K] which have proven to result in a good behavior and steady state around a 1 [K] zone of the target temperature.

The code is shown below:

```
//TempSensors
float tempBlock;
float tempAmb;
float tempDiff = 0.0;
int internal;
int ambient;
int intPin = 0;
int ambPin = 1;
float desTemp = 20.0;


//Input
String in;

//Commmand
float setval = 0.0;
bool Heat = false;
bool Cool = false;

//Peltier
const int Pin1 = 9;
const int Pin2 = 10;

//LEDs
const int LEDred = 3;
const int LEDblue = 2;

 //PID
 const float Kp = 25;
 const float Ki = 0.1;
 float iMax = 200;
 float iMin = -200;
 float P_Term = 0;
 float I_Term = 0;
 float i_Temp = 0;

 void setup()
 {
   analogReference(INTERNAL);
   pinMode(Pin1, OUTPUT);
   pinMode(Pin2, OUTPUT);
   pinMode(LEDred, OUTPUT);
   pinMode(LEDblue, OUTPUT);
   Serial.begin(9600);
   digitalWrite(LEDred, HIGH);
   digitalWrite(LEDblue, HIGH);
   delay(1000);
   digitalWrite(LEDred, LOW);
   digitalWrite(LEDblue, LOW);
 }
```

```
void loop()
{
  internal = analogRead(intPin);
  ambient = analogRead(ambPin);
  tempBlock = internal / 9.31;
  tempAmb = ambient / 9.31;

  Serial.flush();
  if (Serial.available() > 0) {
    in = Serial.readString();
    desTemp = in.toFloat();
    setval = PIDControl(desTemp, tempBlock);
    Serial.print(tempBlock);
    Serial.print(";");
    Serial.println(tempAmb);
  }
  else
  {
    setval = PIDControl(desTemp, tempBlock);
  }

  if (setval > 250)
    setval = 250;
  if (setval < -250)
    setval = -250;

    if (setval > 1.0)
    { //HEAT
      if (!Heat)
      {
        analogWrite(Pin1, 0);
        analogWrite(Pin2, setval);
        digitalWrite(LEDred, HIGH);
        digitalWrite(LEDblue, LOW);
        Heat != Heat;
      }
    }
    else if (setval < -1.0)
    { //COOL
      if (!Cool)
      {
        analogWrite(Pin1, -(int)setval);
        analogWrite(Pin2, 0);
        digitalWrite(LEDred, LOW);
        digitalWrite(LEDblue, HIGH);
        Cool != Cool;
      }
    }
```

```
  }
  else
  { //OFF
    analogWrite(Pin1, 0);
    analogWrite(Pin2, 0);
    digitalWrite(LEDred, LOW);
    digitalWrite(LEDblue, LOW);
  }
  //  Serial.print("setval: ");
  //  Serial.println((int)setval);
  delay(1000);

}

float PIDControl(float desired, float actual) {
  float err = desired - actual;
  P_Term = Kp * err;
  i_Temp += err;
  if (i_Temp > iMax) {
    i_Temp = iMax;
  }
  else if (i_Temp < iMin) {
    i_Temp = iMin;
  }
  I_Term = Ki * i_Temp;

  float set =  P_Term  + I_Term;
  if (set > 250)
    set = 250;
  if (set < -250)
    set = -250;
  return set;
}
```

4 PROCESSING

The computer communicates with the Arduino through the serial communication. The computer is considered as master, sending the desired values to the Arduino. The latter one will have a control loop and controls the hardware.

As can been seen on Figure 4-1 a slider controls the desired temperature. Processing will send this value to the Arduino and wait for a response. This response contains the temperature of the Peltier cell and the ambient temperature. The measured values are put into an array together with a timestamp. This array is then plotted into an xy-graph. The giCentre library was used for this graph.

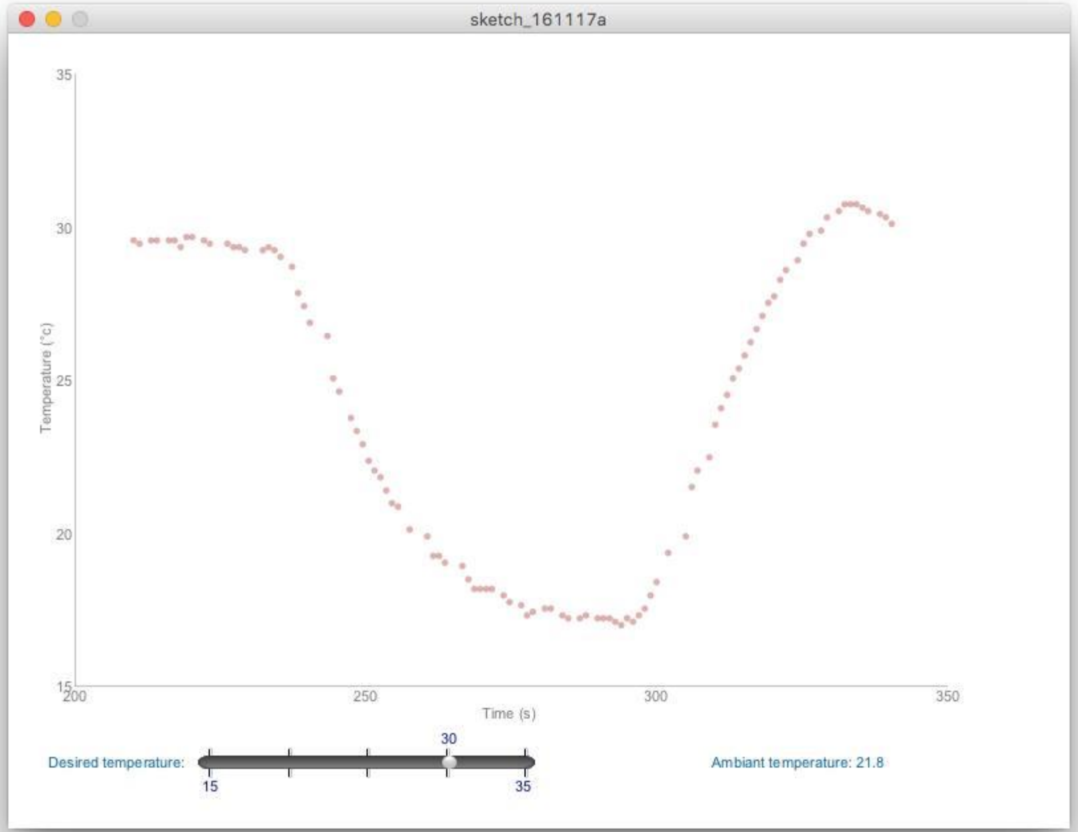
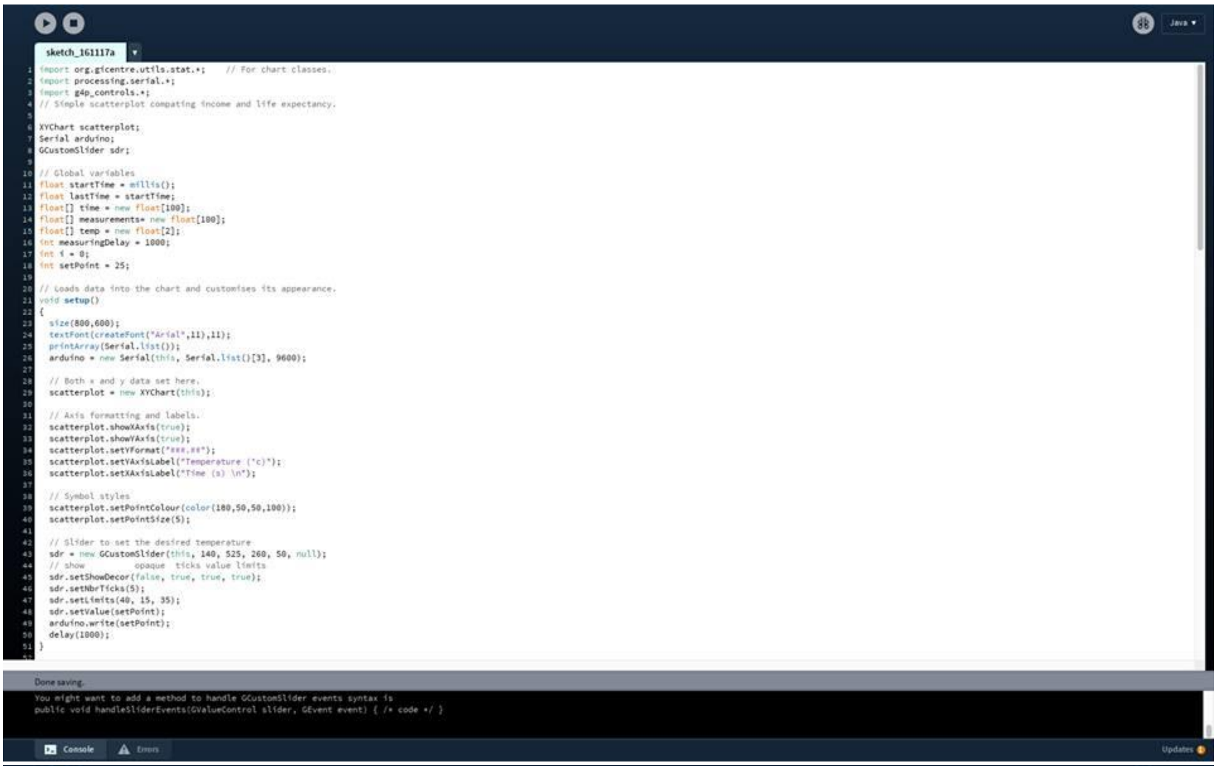


Figure 4-1: GUI

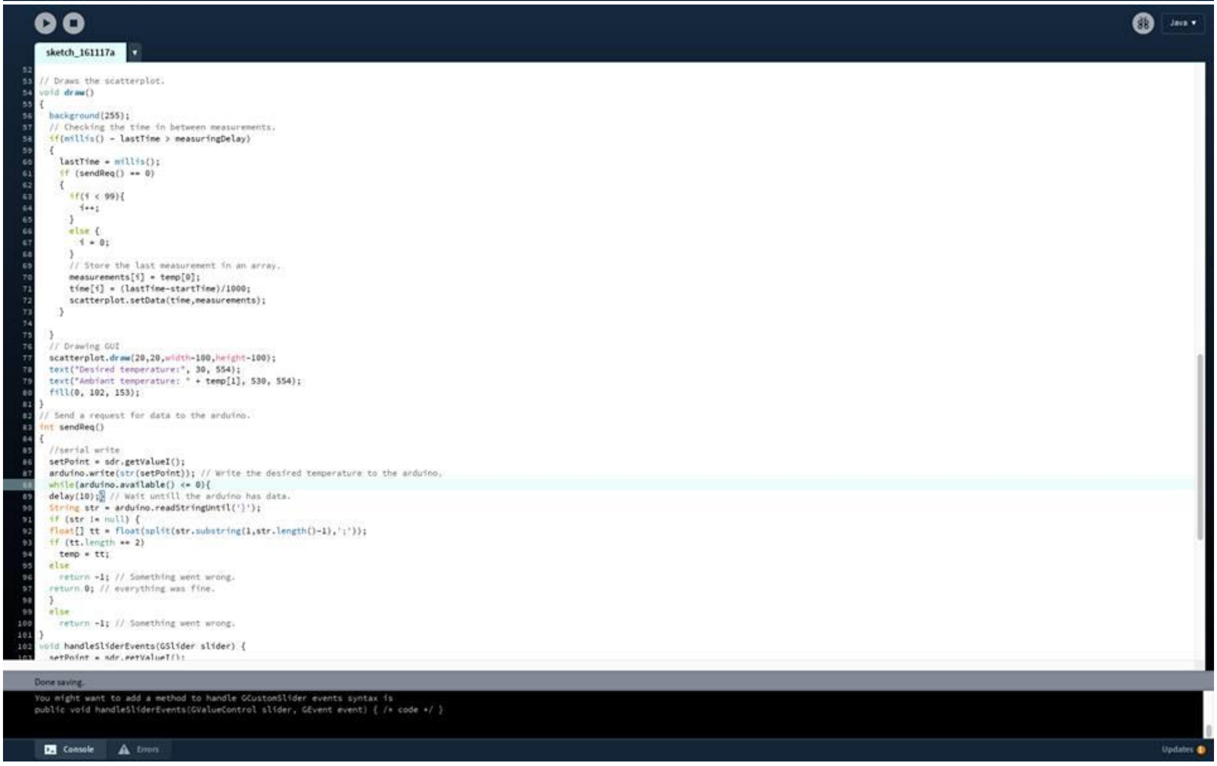
4.1 CODE

Code used in Processing:



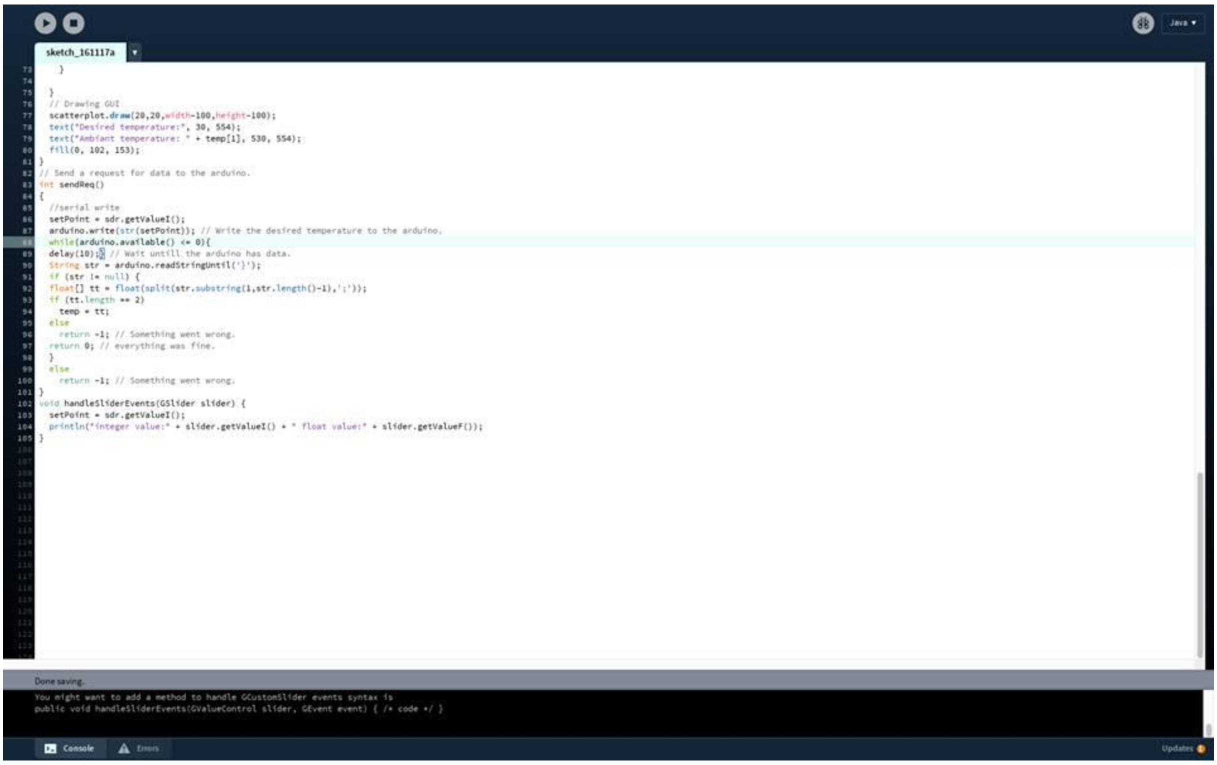
```
import org.gicentre.utils.stat.*;    // For chart classes.
import processing.serial.*;
import g4p_controls.*;
// Simple scatterplot compating income and life expectancy.

XYChart scatterplot;
Serial arduino;
GCustomSlider sdr;

// Global variables
float startTime = millis();
float lastTime = startTime;
float[] time = new float[100];
float[] measurements= new float[100];
float[] temp = new float[2];
int measuringDelay = 1000;
int i = 0;
int setPoint = 25;

// Loads data into the chart and customises its appearance.
void setup()
{
  size(800,600);
  textFont(createFont("Arial",11),11);
  printArray(Serial.list());
  arduino = new Serial(this, Serial.list()[3], 9600);

  // Both x and y data set here.
  scatterplot = new XYChart(this);

  // Axis formatting and labels.
  scatterplot.showXAxis(true);
  scatterplot.showYAxis(true);
  scatterplot.setYFormat("###.##");
  scatterplot.setYAxisLabel("Temperature ('c)");
  scatterplot.setXAxisLabel("Time (s) \n");

  // Symbol styles
  scatterplot.setPointColour(color(180,50,50,100));
  scatterplot.setPointSize(5);

  // Slider to set the desired temperature
  sdr = new GCustomSlider(this, 140, 525, 260, 50, null);
  // show          opaque  ticks value limits
  sdr.setShowDecor(false, true, true, true);
  sdr.setNbrTicks(5);
  sdr.setLimits(40, 15, 35);
  sdr.setValue(setPoint);
  arduino.write(setPoint);
  delay(1000);
}
```

```
Done saving.
You might want to add a method to handle GCustomSlider events syntax is
public void handleSliderEvents(GValueControl slider, GEvent event) { /* code */ }
```



```
// Draws the scatterplot.
void draw()
{
  background(255);
  // Checking the time in between measurements.
  if(millis() - lastTime > measuringDelay)
  {
    lastTime = millis();
    if (sendReq() == 0)
    {
      if(i < 99){
        i++;
      }
      else {
        i = 0;
      }
      // Store the last measurement in an array.
      measurements[i] = temp[0];
      time[i] = (lastTime-startTime)/1000;
      scatterplot.setData(time,measurements);
    }

  }
  // Drawing GUI
  scatterplot.draw(20,20,width-100,height-100);
  text("Desired temperature:", 30, 554);
  text("Ambiant temperature: " + temp[1], 530, 554);
  fill(0, 102, 153);
}
// Send a request for data to the arduino.
int sendReq()
{
  //serial write
  setPoint = sdr.getValueI();
  arduino.write(str(setPoint)); // Write the desired temperature to the arduino.
  while(arduino.available() <= 0){
  delay(10);} // Wait untill the arduino has data.
  String str = arduino.readStringUntil('}');
  if (str != null) {
  float[] tt = float(split(str.substring(1,str.length()-1),';'));
  if (tt.length == 2)
    temp = tt;
  else
    return -1; // Something went wrong.
  return 0; // everything was fine.
  }
  else
    return -1; // Something went wrong.
}
void handleSliderEvents(GSlider slider) {
  setPoint = sdr.getValueI();
```

```
Done saving.
You might want to add a method to handle GCustomSlider events syntax is
public void handleSliderEvents(GValueControl slider, GEvent event) { /* code */ }
```


sketch_161117a
}
}
// Drawing GUI
scatterplot.draw(20,20,width-100,height-100);
text("Desired temperature:", 30, 554);
text("Ambiant temperature: " + temp[1], 530, 554);
fill(0, 102, 153);
}
// Send a request for data to the arduino.
int sendReq()
{
//serial write
setPoint = sdr.getValueI();
arduino.write(str(setPoint)); // Write the desired temperature to the arduino.
while(arduino.available() <= 0){
delay(10);} // Wait untill the arduino has data.
String str = arduino.readStringUntil('}');
if (str != null) {
float[] tt = float(split(str.substring(1,str.length()-1),';'));
if (tt.length == 2)
temp = tt;
else
return -1; // Something went wrong.
return 0; // everything was fine.
}
else
return -1; // Something went wrong.
}
void handleSliderEvents(GSlider slider) {
setPoint = sdr.getValueI();
println("integer value:" + slider.getValueI() + " float value:" + slider.getValueF());
}
Done saving.
You might want to add a method to handle GCustomSlider events syntax is
public void handleSliderEvents(GValueControl slider, GEvent event) { /* code */ }
Console
Errors
Updates
Java

5 RÉSUMÉ

The complete setup can be seen in figure 5-1. This project showed the flexibility of open hardware and software platforms. These platforms are easy to use and to implement and allow to create solutions to a big variety of problems.

In this case a good functioning temperature controller was created in a few days.

We really enjoyed this course, thanks a lot!

Alexander & Christian

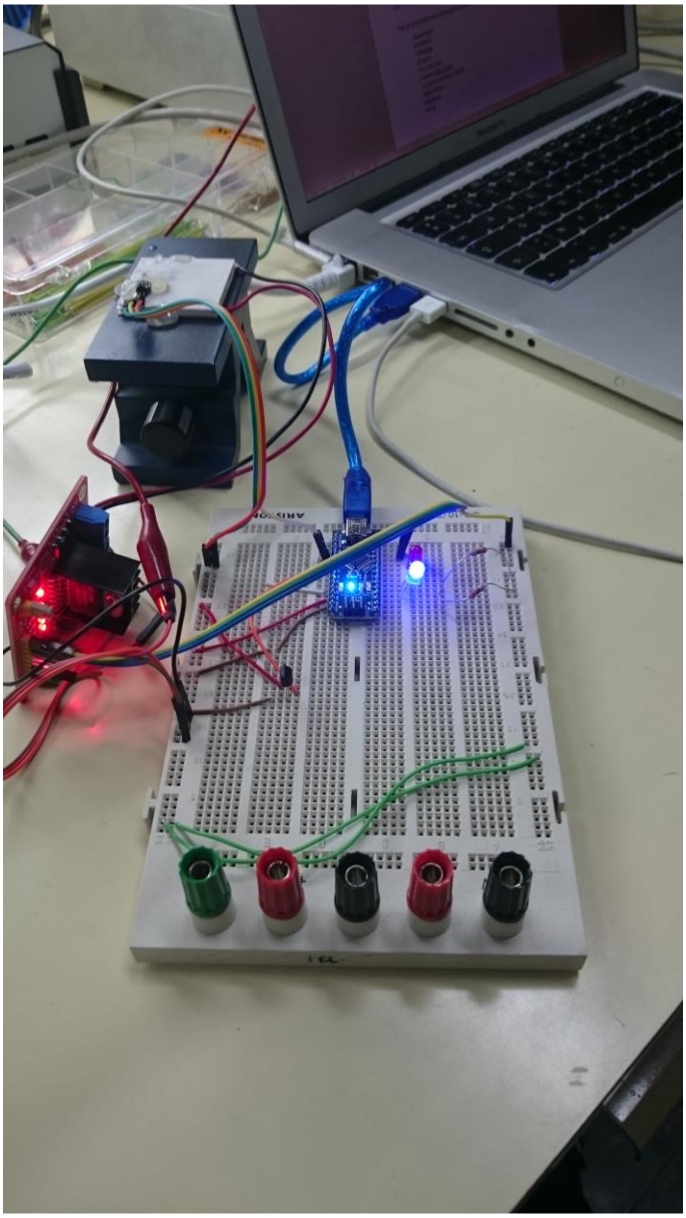


Figure 5-1: Setup