Politecnico di Milano

School of Industrial and Information Engineering
Biomedical Engineering

Technologies for Sensors and Clinical Instrumentation

PROJECT REPORT

Color sensor

Massimo ALFONZO
Erik BROVELLI
Chloé CANAVATE
Hortense LEIGNEL
Carlo MANGIANTE
Iva MILOJKOVIC

Academic year 2020–2021

Contents

Demo's Link: *click here*

List of Figures

1 Introduction

The objective of this project was to construct a device capable of detecting colors using a single photodiode and a RGB LED.

The market for color detecting sensors of this type has seen a steady growth in the recent years, since they are used in various industries including automotive, manufacturing, textiles, chemical, food and beverage, pharmaceutical. The purposes of this device are varied: in consumer electronics they are used for backlight control and display calibration, in manufacturing for detecting components and matching their colors, and they are even used for fluid and gas analysis and in the pharmaceutical field. The characteristic most sought after by producers is the ability for the sensor to be able to differentiate accurately and consistently a lot of shades of colors both in the market for private citizens and in the one for businesses. This is often achieved by using more photodiodes with coloured filters in an array configuration and by working digitally, reaching accuracies of $\pm 0.5\%$ F.S. [1]. Since our sensor only uses one photodiode and works analogically it reaches a lower accuracy while allowing for a wider range of applications.

The device itself works thanks to the photodiode, a sensor capable of transforming light irradiance into current, and therefore sensible to the light intensity. Based on the properties of color reflection, we knew that the resulting output current of the PD represented how optimal the light reflection had been, and an optimal color reflection means that either the surface is white, therefore reflecting all wavelengths, or the same color as the transmitted one.

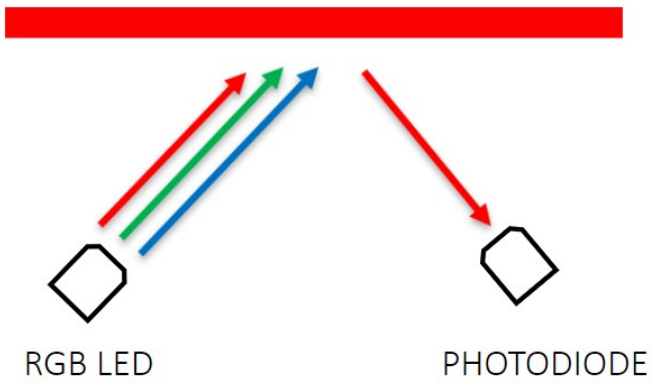


Figure 1: Sensor's principle

With these properties in mind, we created a color sensor which would work following this procedure: by sequentially illuminating the object with the three colors of the LED, and measuring the current output given by the photodiode, for each of those LED colors, we could assign three values of color intensity to the object in an RGB format, and put them together to obtain an estimation of the unknown color.

In order to translate the output of the photodiode into RGB values, we needed to properly calibrate the device, meaning we had to retrieve the color range of converted currents using only a simple black and white calibration. The *white* calibration, giving the highest photodiode response, consisted in sequentially radiating a white piece of paper with the LEDs and measuring the photodiode output for each of the colors, namely r_w, g_w and b_w. The *black* calibration consisted in measuring the output signal without turning on the LED, in order to measure the dark currents, getting r_b, g_b, b_b.

Therefore, the color detection worked thanks to the following equations. While presenting a colored object to the sensor, the three measured intensities - r_c, g_c, b_c - are then converted into a value between 0 and 255 with respect to the ranges of measurements for each color component, obtained with the black and white calibration.

$$R = \frac{r_c - r_b}{r_w - r_b} \times 255 \qquad G = \frac{g_c - g_b}{g_w - g_b} \times 255 \qquad B = \frac{b_c - b_b}{b_w - b_b} \times 255 \tag{1}$$

After acquiring a professional color palette with exact RGB values, we decided to do a further calibration in order to compare it with the initial one (see section Results). This consisted not only in the use of the black and white range obtained before, but with the addition of selecting 70 different colors in the color palette, measuring the R, G, B components by following the same process described above and then plotting the real values against the measured ones on Excel.


```
double arrayX1[] = {2.2194214917, 2.1882544837, 2.1099261184};
double arrayX2[] = {-0.0081525719, -0.0083532605, -0.0081855873};
double arrayX3[] = {0.0000132053, 0.0000144655, 0.0000150248};
```

Figure 2: Coefficients obtained through regression analysis

By applying a 3^{rd} order polynomial regression analysis to those graphs, we obtained a set of coefficients (see Fig.2) allowing us to convert the R, G, B calculated values into ones closer to the reality and therefore having the best accuracy possible. For example, the red component of a color is now first computed thanks to the equation (1), and then by applying the correction coefficients:

$$R_{corr} = 0.0000132053R^3 - 0.0081525719R^2 + 2.2194214917R \tag{2}$$

Those resulting coefficients are saved in the memory of the device along with the white and black ranges (since we verified they are constant over time), in order to remove the need of performing a calibration on the user's end, thus making our device factory calibrated.

2 Hardware

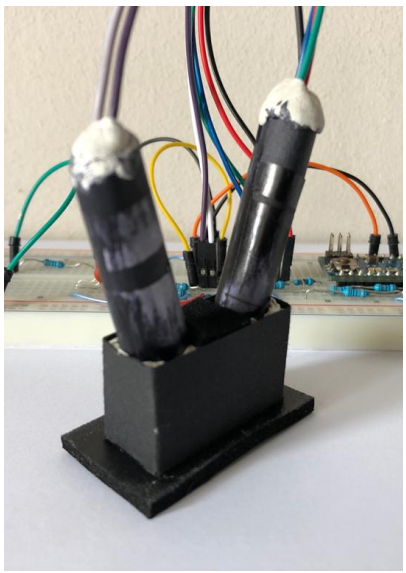
This section aims to present the whole hardware, giving an overview about how our colour sensor was built and the different issues we faced. A general hardware description followed by a detailed history of our sensor conditioning is presented and finally an explanation about how we dealt with a repeatability issue is provided.

2.1 Hardware description

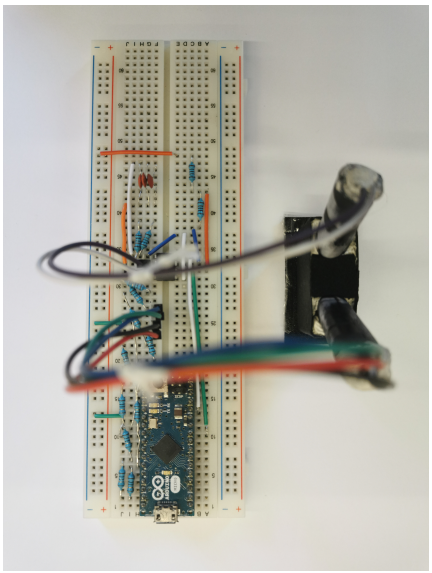
2.1.1 Sensor probe

Our sensor aims to detect colors, therefore the first step needed was to create an environment that prevented ambient light from interfering with the photodiode measurements. In order to do this, we initially created a black box made of cardboard surrounding the LED and the photodiode, with the idea of placing the object on top of the cardboard box, and therefore preventing light from reaching the photodiode.

Later on during the development of the device we opted to have a more "mobile" sensor (external to the breadboard), in order to scan object surfaces in an easier way. Regarding the sensor probe, we decided to place the photodiode and the RGB LED (both encapsulated in cardboard tubes) so as to form a 30° angle between them, aiming to maximize the light intensity collected by the photodiode. We also isolated the photodiode from the LED to avoid direct lighting onto the sensor. Additionally, we created a cardboard casing in which we put both the sensor and the photodiode and, moreover, we covered the top of the probe with a felt pad, which allowed us to put some pressure on the device and limit even more the environmental light interference, without any risk of ruining the object in exam.



(a) Sensor probe



(b) Breadboard

(c) Probe's inside

Figure 3: Hardware setup

2.1.2 RGB LED conditioning

Photodiodes have a wavelength dependent response. Therefore, when we sequentially turn on the red, green and blue lights, the photodiode does not generate the same current even if the light intensity is constant. Moreover, the light intensity produced by the RGB LED might be different between each pin. For those reasons, in order to work on the same scale, we added resistances acting as voltage dividers on each pin of the RGB LED: 50Ω for the red pin, 10Ω for the green one and 50Ω for the blue one.

2.1.3 Schematics of hardware connections

The whole hardware is presented through a Fritzing schema, showing the external probe composed by the photodiode in photovoltaic mode (short circuit) and the RGB LED, followed by the conditioning circuit further described below.

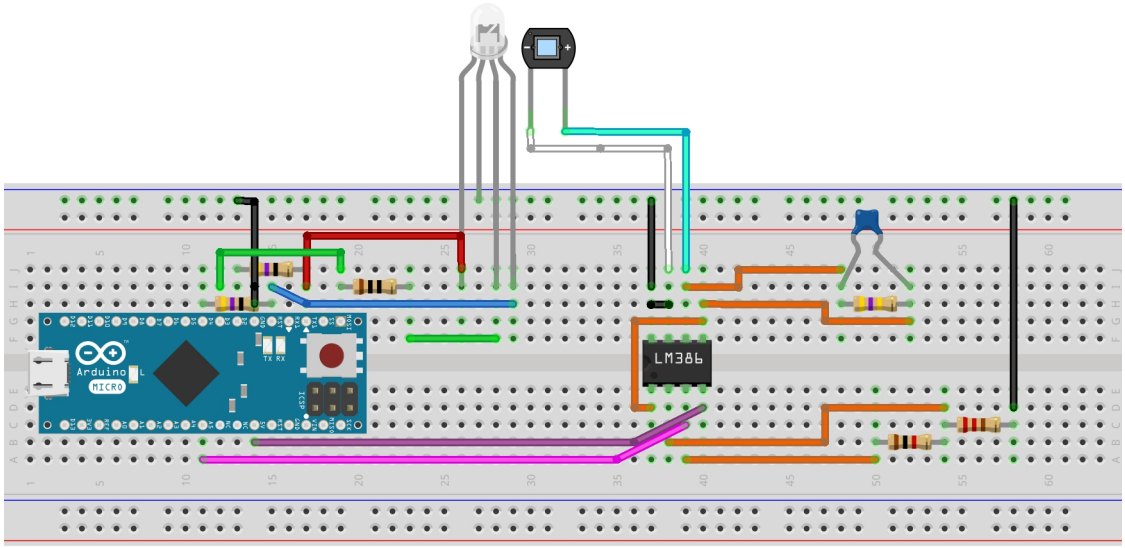
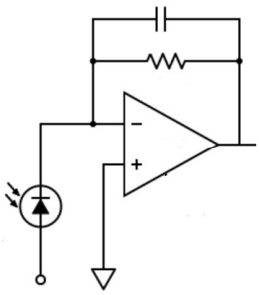


Figure 4: Fritzing schema

2.2 Sensor conditioning

2.2.1 First version

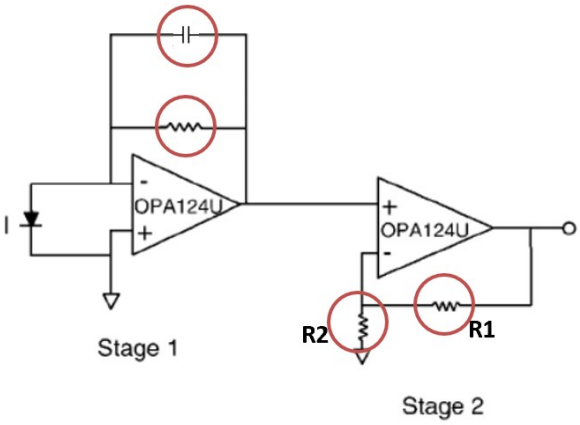


The low currents produced by the photodiode needed first to be converted into voltages, to become readable by the Arduino ADC, and then amplified. Our first idea was to use a transimpedence amplifier circuit with a high resistance in order to obtain a sufficient gain. With the aim of choosing the optimal values for the resistance and the capacitor, we measured the time needed for the response to stabilize with different R-C value pairs. To do so, we retrieved the outcomes after reflection of the RGB LED light onto a white paper, then we normalized the values and compared the results. Finally, the couples 5MΩ-104pF and 10MΩ- 104pF were found to be the optimal choices, with a stabilized outcome after respectively 1 and 2 seconds.

However, remembering that for a color detection we need to sequentially measure the reflected red, green and blue lights, the final time needed to detect a color with our system was between 3 and 6 seconds. Thus, it was critical for our system to find a solution that drastically reduced this period.

2.2.2 2 stages amplifier

For the purposes of drastically reducing the time needed to detect a color, we made the choice to improve the conditioning circuit through a two-stages amplifier. In it, stage 1 is composed by the transimpedence amplifier, aiming here only to convert the current into a voltage; stage 2 is a non-inverting amplifier with a gain of $G = 1 + \frac{R1}{R2}$. We had to choose not two, but four parameters: R and C in the transimpedence circuit; R1 and R2 in the non-inverting amplifier.



Firstly, we sought to determine the orders of magnitude of R1 and R2 sufficient to avoid saturation within the amplifier. After testing, $10^3\Omega$ and $10^2\Omega$ came up to be reasonable choices for R1 and R2 respectively.

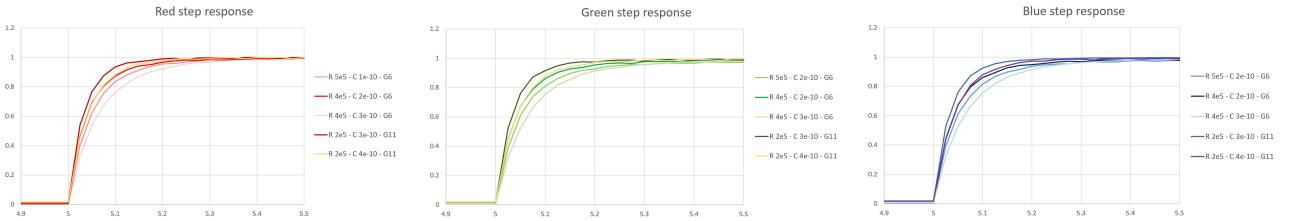
To study this improved conditioning circuit, we repeated the same experiments as above. Indeed, for 5 different configurations summarized in the table below, we plotted the normalized step response (see Fig.5).

	1	2	3	4	5
R (Ω)	5.10^5	4.10^5	4.10^5	2.10^5	2.10^5
C (F)	1.10^{-10}	2.10^{-10}	3.10^{-10}	3.10^{-10}	4.10^{-10}
R1 (Ω)	1000	1000	1000	1000	1000
R2 (Ω)	200	200	200	100	100
G	6	6	6	11	11

This new two-stage configuration provides a much faster stable response, but inevitably increases noise. Therefore we had to find a trade-off between speed and noise. We estimated the noise by measuring the signal amplitudes after stabilization. During each color detection we averaged 200 measurements, taken every 0.1 ms, to obtain a more accurate value.

The noise did not change too much throughout the 5 configurations, so the optimal choice was to go for the fastest circuit, defined by $R = 200k\Omega$, $C = 312pF$, $R1 = 1k\Omega$ and $R2 = 100\Omega$, for which the response stabilizes after **300 ms**. In the end, this improved conditioning circuit has significantly reduced the time needed for a color detection to about **one second**.

During those experiments, we noticed that the response was slightly different between the red, green and blue lights. This observation is in perfect concordance with the wavelenght-dependent response of a photodiode.



(a) Red (b) Green (c) Blue

Figure 5: Normalized step responses of the sensor

2.3 Repeatability issue

The last important part about hardware is how we dealt with a repeatability problem. Indeed, during repeated measurements (without delay), the outcome of the circuit was decreasing over time. For example, after five consecutive measurements (without moving anything), the red component of a color could change from 255 to 250, and this can be problematic if we imagine a real situation where the user tests the device by repeating a measurement.

We did not manage to find a clear explanation to this phenomena, but our hypothesis was that capacitors in the circuit may produce a current for a brief while after light has gone away. This current may be a disturbance for additional measurements. Therefore, waiting for them to discharge appeared to be the best solution. How much "rest time" was enough, to solve the repeatability problem?

To answer, we simulated 10 consecutive color measurements, each one obtained by switching alternatively the red, the green and the blue lights and measuring the outcome after reflection onto a white paper. We repeated this procedure for 5 different rest times (see Fig.6).

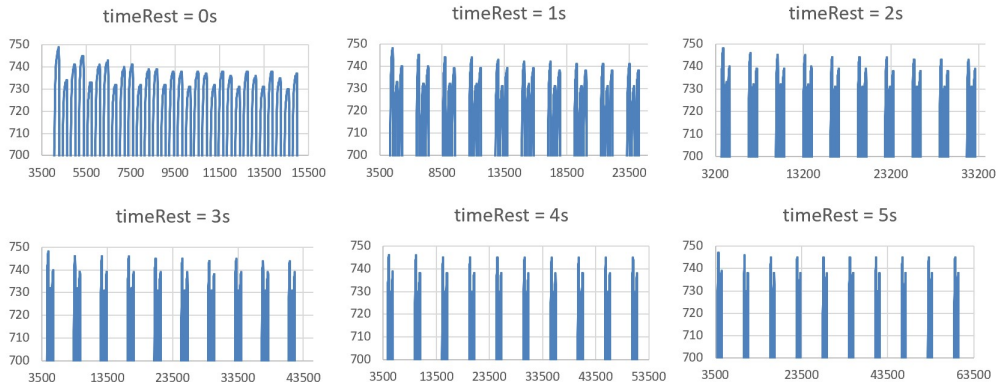


Figure 6: Color detection cycle simulations

Now, we computed the "error" as the difference on the 255 scale between the values in the 1^{st} cycle and the ones in the 10^{th} cycle. We wanted those errors to be lower than 1% for each color component, which corresponds to 2 points in the 255 scale. Letting the circuit rests during 2 seconds came up to be the best trade-off, since the error limit is respected and it just corresponds to the time spent by the user to read the results on the screen.

	max-min			max intensity			error (255 scale)		
timeRest	R	G	B	R	G	B	R	G	B
0	14	4	4	749	734	741	4,8	1,4	1,4
1	8	2	2	748	733	740	2,8	0,7	0,7
2	5	1	2	748	733	740	1,7	0,4	0,7
3	4	1	1	748	732	740	1,4	0,4	0,4
4	1	0	1	746	730	738	0,3	0,0	0,4
5	2	0	1	747	730	738	0,7	0,0	0,4

3 Firmware

The goal of the Arduino code is to be as efficient and user friendly as possible. Excluding the *setup()* and *loop()* functions the sketch has 3 different types of functions: *checkColour()*, *printColour()*, *getReading()*. Moreover, all variables are declared as global and are explained in the corresponding function.
Starting from the *loop()* function, to keep the code simple, Arduino takes instruction from Processing only when the colour detection is needed. As soon as the serial message "COLOR" is received, Arduino starts detecting the colour, computes the results and then sends a serial message with the outcomes to Processing.

```
void loop(void){
  //receive serial messages from Processing
  if(Serial.available()){
    string = Serial.readStringUntil('\n');//read serial message from processing
    if(string == "COLOR"){                  //until the end character "\n"
      string="";
      checkColour();
      printColour();
    }
  }
}
```

Figure 7: Arduino code - loop() function

During the measurement the main function is *getReading()*, that takes as input "times", which is the number of samples requested, and waits "timeRead" ms between one *analogRead(photodiode)* and the other. In this configuration the variables chosen were "times"=200 and "timeRead"=0,1ms. At the end the average result is saved on "avgRead".

```
//take times reading and calculate avgRead
void getReading(int times){
  reading=0;                    //inizialize reading
  for(int j=0; j<times; j++){
    reading += analogRead(photodiode);
    delay(timeRead);            //timeRead for each reading
  }
  avgRead = (reading/times);  //calculate the average and set it
}
```

Figure 8: Arduino code - getReading() function

The detection of the colour is done in the *checkColour()* function. Arduino will send serial messages to Processing every time a different phase starts, so, following the led's light changes, at the beginning and at the end of the detection. Immediately after having lit up each colour of the RBG led Arduino waits accordingly to the variable *delay(ohm)* which is 300ms, allowing the circuit stabilization. A *delay(wait)*, where "wait" is equal to 50ms, has been added in order to avoid all possible serial printing errors. In the Arduino's memory there are present the maximum and minimum values in voltage terms named "whiteArray", "blackArray" and "greyDiff" which is the difference between the first two. The values obatined are: whiteArray=[748,733,740] and blackArray=[12,12,12]. The computation of the result is done with two equations: the first is useful to transform the input voltage value read on the photodiode's pin in RBG scale (0-255), the second uses the regression equations to adjust the result. After the computation Arduino checks the result and avoids incorrect values, meaning the ones above 255 and below 0.

```
//colour detection
void checkColour(){
  Serial.println("colorStart");
  for(int i=0; i<=2; i++){
    delay(wait);                                  //delay to avoid serial messages errors
    digitalWrite(ledArray[i],HIGH);               //turn on one led at a time
    Serial.println(rgbArray[i]);                  //Processing will colours the graphic led
    delay(ohm);                                   //delay for the photodiode time response
    getReading(nRead);                            //nRead is the number of readings
    colourArray[i] = avgRead;                     //save average reading in the array

    //calculating the colour outcome
    colourArray[i] = (colourArray[i] - blackArray[i])/(greyDiff[i])*255;
    colourArray[i] = arrayX1[i]*colourArray[i] + arrayX2[i]*colourArray[i]*colourArray[i]
      + arrayX3[i]*colourArray[i]*colourArray[i]*colourArray[i];

    if(colourArray[i]>255) colourArray[i]=255;  //avoid sensor errors
    if(colourArray[i]<0) colourArray[i]=0;
    digitalWrite(ledArray[i],LOW);                //turn off the current LED
    Serial.println("f");                          //turn off the graphic led on processing
  }
  delay(wait);
  Serial.println("colorDone");
}
```

Figure 9: Arduino code - checkColour() function

At the end Arduino sends to Processing the final results computed, separated by a comma in order to easily locate the values. The *round()* function is used to change the values of "colourArray" with the closest integer values. After that, Arduino handles the rest time: it sends a serial message every rest/3 ms in order to draw in the GUI 3 different "z" delayed by rest/3 ms. This feature gives to the user a graphic idea of the rest time remaining before a new allowed measurement and can be interpreted as the time needed for the user to read the result, minimising the impact of this constraint. The time rest chosen is 2000ms in order to avoid errors during repeated detections, as previously explained in the Hardware section.

```
//send to Processing colour outcomes
void printColour(){
  Serial.print(round(colourArray[0]));      //transform float value in int
  Serial.print(",");
  Serial.print(round(colourArray[1]));
  Serial.print(",");
  Serial.print(round(colourArray[2]));
  for(int i=0; i<=2; i++){
    delay(rest/3);
    if(i==0) Serial.println("zz");          //second "Z"
    else if(i==1) Serial.println("zzz");    //third "Z"
    else if(i==2) Serial.println("zzzEnd"); //rubber the led graphic
  }
}
```

Figure 10: Arduino code - printColour() function

4 Software

The processing graphic interface has been created in order to give the user a software as efficient, as aesthetically pleasing and as easy to use as possible. Before explaining the noteworthy parts of the written code, it could be useful to show the final graphic interface. The figure below represents the graphical interface after a few detections. Immediately visible are the colour wheel with the saturation rectangle below representing the colour detected with the HSB (Hue Saturation Brightness) model, the history of the last 5 detected colours with RGB values and the colour printed in a small rectangle beside, the actual values of the last colour detected represented both numerically and graphically (vertical bars) with the actual colour printed in the big square on the right, the messages from the system in the box in the upper right side, the color button, a graphical representation of a led in the upper left and last but not least a red reset button on the bottom right.

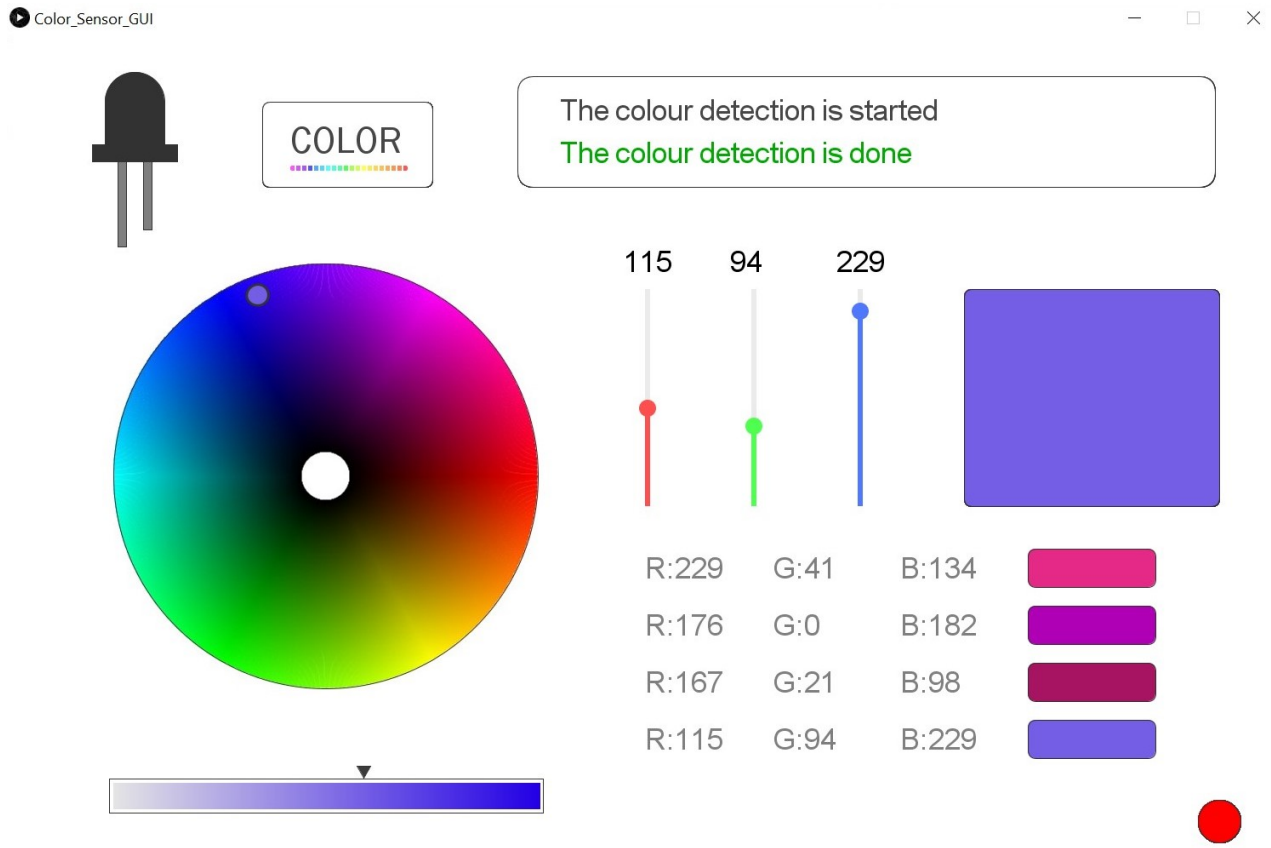


Figure 11: Processing GUI

Basically the main functions of the graphical interface are: reading the serial messages from Arduino (during the colour detection) and, using the GUI buttons, checking which button has been clicked and then perform the related function.
In *draw()* Processing checks if there is any message in the serial port, if there is one, it will read the message and do the corresponding tasks. Moreover, it checks the position of the mouse pointer in order to highlight (*colourButton(false)*) or not (*colourButton(true)*) the buttons whenever the mouse's arrow passes over a button, which is extremely useful in order to give a graphical feedback to the user before clicking a button.

```
void draw(){
  if(port.available()>0) getInstruction();  //getting serial messages from arduino

  //highlight the button if we pass the mouse over it
  if(mouseX>300 && mouseX<500 && mouseY>80 && mouseY<180) colourButton(false);
  else colourButton(true);
  if((y_center_res+sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))>=mouseY &&
    (y_center_res-sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))<=mouseY)
      resetButton(false);
  else resetButton(true);
}
```

Figure 12: Processing code - draw() function

In the *mousePressed()* interrupt service routine function, the software checks if the user has clicked on the colour or on the reset button. It is important to note that the "buttonFlag" boolean variable is used to block the user from sending any input during a detection, therefore "buttonFlag" remains *True* until Arduino and Processing have completed their tasks.

```
void mousePressed(){
  if(!buttonFlag){
    if(mouseX>300 && mouseX<500 && mouseY>80 && mouseY<180){
      port.write("COLOR\n");
      buttonFlag=true;
    }
    else if((y_center_res+sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))>=mouseY &&
      (y_center_res-sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))<=mouseY) reset();
  }
}
```

Figure 13: Processing code - mousePressed() function

Analyzing the functions shown before, starting from *getInstruction()*: this function reads all serial messages from Arduino and according to the string performs different actions:

- "colorDone" is received when Arduino finishes the colour detection, Processing will use the *dectected()* function to draw all graphic objects related to a new detection:
  - The colour wheel.
  - The saturation rectangle positioned under the wheel.
  - The small circle inside the colour wheel containing the detected colour.
  - The RGB vertical bars graphically representing the RGB values and the big square on the right side filled in with the detected colour.
  - The triangle used as a pointer in the saturation rectangle.

  After having finished drawing all the graphical objects, Processing adds to the colours' history the new detection.
- "colorStart" writes in the messages box that the detection is started.
- "f", "r", "g", "b" color the graphic led following in time the real RBG led colours.
- "zz", "zzz", "zzzEnd" change the state of the led during the rest displaying three different "z" in the top left of the led, representing the time progress.

```
void getInstruction(){
  arduinoMsg = port.readString(); //reads all "menu changes" + RGB components detection
  instruction = trim(split(arduinoMsg, '\n')[0]);  //get only "menu changes"

  if(instruction!=null){
    if(instruction.equals("colorDone")){         //colour detection finished
      RGB_components = trim(split(arduinoMsg, '\n')[1]);  //RGB numerical values
      rubberMsg();                               //rubber function messages
      fill(0,170,0);
      text("The colour detection is done", x1,y1);
      y1=y1+off1;                                //add space before next row
      wRed = int(split(RGB_components,',')[0]);  //taking RGB values
      wGreen = int(split(RGB_components,',')[1]);
      wBlue = int(split(RGB_components,',')[2]);
      detected();                                //draw new colour in the detecting zone
      rubberRGB();                               //rubber function colour detected
      fill(127);
      text("R:"+wRed, x2, y2-10);                //-10 to center the text with colour rect
      text("G:"+wGreen, x2+150, y2-10);
      text("B:"+wBlue, x2+300, y2-10);
      fill(wRed,wGreen,wBlue);
      rect(x2+450, y2-50+5, 150, 50-5, 10);  //colour rectangle next to RGB values
      y2=y2+off2;
      restLed(1);
    }
    else if(instruction.equals("f")){              //graphic led off
      fill(50);
      led();
    }
    else if(instruction.equals("r")){              //graphic led red
      fill(255,0,0);
      led();
    }
    else if(instruction.equals("g")){              //graphic led green
      fill(0,255,0);
      led();
    }
    else if(instruction.equals("b")){              //graphic led blue
      fill(0,0,255);
      led();
    }
    else if(instruction.equals("zz")) restLed(2);   //graphic led rest zz
    else if(instruction.equals("zzz")) restLed(3);  //graphic led rest zzz
    else if(instruction.equals("zzzEnd")){          //graphic led rest off
      restLed(0);
      buttonFlag=false;
    }
    else if(instruction.equals("colorStart")){
      rubberMsg();
      fill(70);
      text("The colour detection is started", x1, y1);
      y1=y1+off1;
    }
  }
}
```

Figure 14: Processing code - getInstruction() function

The other function done in *mousePressed()* is *reset()* which is a button that the user can use to reset the entire GUI to the initial state.

```
//reset processing
void reset(){
  //erase colour detected hystory
  fill(255);
  noStroke();
  rect(750,600+3,620,325);//delete colour detected hystory
  stroke(50);
  count2=0;

  //erase arduino messages
  arduinoMsg();
  count1=0;
  y1=102;

  //resetting from the beginning
  wRed=255; wGreen=255; wBlue=255;
  detected();
}
```

Figure 15: Processing code - reset() function

The colour wheel is drawn with a specific function which only performs graphical actions. It is important to highlight that the decision to use a JPG image of the colour wheel instead of drawing it through the code has been taken in order to keep the software as efficient and as quick as possible. So, after having drawn the colour wheel using the code for the first time, we later opted to take a screenshot of it and use it as an image in the code.

Two rubber functions are used in order to handle the messages box and the RBG colour detected history:

- *rubberRGB()*: draws a maximum of 5 rows of RGB values and a small rectangle with the detected colour represented in it. After the first 5 rows Processing will also erase the row immediately below the last colour detected in order to better show the user which row has been added.

```
//erase messages of RBG hystory
void rubberRGB(){
  count2++;
  rest2=count2%5;
  if(rest2==1) y2=650;   //when i need to start from the first row
  fill(255);
  noStroke();
  rect(750,y2-50+3,600+3,100+35);
  stroke(50);
}
```

Figure 16: Processing code - rubberRGB() function

- *rubberMsg()*: controls the messages box on the upper right side of the GUI.

```
//erase messages in the upper right when it's needed
void rubberMsg(){
  count1++;
  if(count1>2){
    arduinoMsg();
    count1=1;
    y1=102;
  }
}
```

Figure 17: Processing code - rubberMsg() function

In the report the decision has been taken to show and explain only the most noteworthy functions of the software. It is important to consider that besides *detected()* and *colourWheel()*, which have been explained without showing the code, there are many other functions, mostly graphic ones (meaning the functions useful to draw objects in the graphic interface) like *setup()*, *restLed()*, *led()*, *resetButton()*, *arduinoMsg()* and *colourButton()* that are present in the code but have not been explained in the report.

In the code any variable, if it was possible, has been declared as global. Conversely for most of the graphic objects the numerical values have been written directly in the code, since they only refer to coordinates in the GUI window.

In order to have a better visualization of the whole firmware and software implemented, all the code's rows have been added in the Appendix (section 6).

5 Results

5.1 Results of the calibration

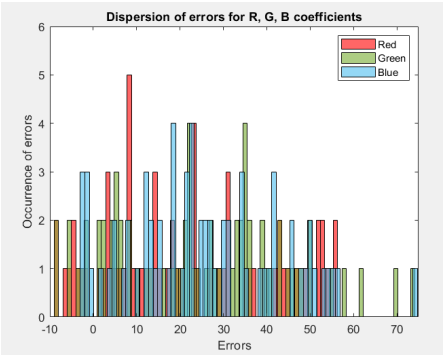


Figure 18: Dispersion of error before calibr.

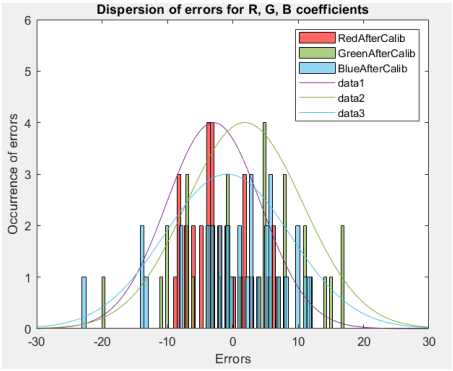


Figure 19: Dispersion of error after calibr.

The figures herein-above show the dispersion of errors between the expected RGB values and the sensor's values, before and after calibration. The error is computed as the difference between the theoretical R, G or B value and the measured value. The theoretical value of the colour we want to assess is indicated on the colour palette.
Before calibration, we noticed that we had a dispersion of errors around 20. The exact values are reported in the table below. This means that our sensor had a darker vision of the colours. To solve this problem, we made a calibration based on the colour palette. According to the second histogram, we managed to reduce the darkening effect of our sensor since the mean value of the errors is now closer to zero. The standard deviation used to be close to 20 before calibration, and it is now lower than 10. The sensor's detected colours are now definitely closer to the analyzed colours.

	Red		Green		Blue	
	Mean	Std	Mean	Std	Mean	Std
Before Calibration	21.91	19.68	24.14	19.96	25.13	17.58
After Calibration	-2.94	7.22	1.82	8.92	-0.85	9.63

Figure 20: Gaussian characteristics before and after calibration

5.2 Sensor's performances

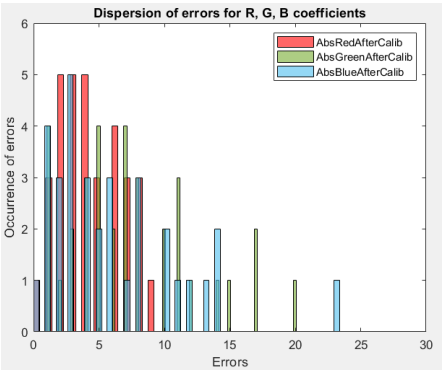


Figure 21: Dispersion of the absolute errors

	Red	Green	Blue
Mean Absolute Error	5.18	7.53	6.91
Mean Absolute Error, %	2.03 %	2.95 %	2.71 %
Maximum Absolute Error	9	20	23
Maximum Absolute Error, %	3.5 %	7.8 %	9.0 %

Figure 22: Characteristics of the sensor

Here above is a histogram of the absolute errors. Through the previous histogram, we define the following parameters as an indication of our sensor's performances.
The accuracy in percentage of the FSO (255) corresponds to the highest maximal error. Therefore, our sensor has an accuracy of ±9%.

5.3 Application

We used the sensor on other supports to check its "external validity". We tried on a different type of paper, on rough lemon and orange's skins and on our own skin, since color sensors are used in the biomedical field to detect, for example, skin diseases.

Figure 23: Different colour papers and their RGB values, assessed by the sensor

Figure 24: Assessment of the lemon's, lime's and orange's skin colour

Figure 25: Assessment of the skin colour through the sensor

Through those last measurements, we assessed the performance of the sensor over other types of surfaces. For the papers, the results seem accurate. For the lime and lemon, the measured colours are darker than the real colours. This may be due to the rough and porous surface of the citrus fruits which doesn't reflect the light correctly. Concerning the orange, the sensor colours are also darker than in reality. We made a precision assessment on that fruit and the colour, sensed over a short period of time, in the same conditions, varies very little. R,G,B values vary from less than 0.4 %. In the lemon and lime measurements we've changed the detected zone to establish different colour of the fruit's peel. For the skin, the values are again darker than the real colours. This may be due to the porous texture of the skin as well as an increased absorption of the rays by the internal layers of the skin.

6 Appendix

Listing 1: Arduino code

```
int photodiode = A5; //photodiode to pin A5
int ledArray[] = {2,3,4}; //led array in D2, D3, D4
String rgbArray[]={"r","g","b"}; //used in serial communication
float reading; //used in getReading()
String string; //used for reading messages from processing
int avgRead; //temporary variable during sampling

//floats to hold colour arrays
//when we print the result we transform these as int variable
double colourArray[] = {0,0,0};
int whiteArray[] = {748,733,740};//white memory of the program
int blackArray[] = {12,12,12};//black memory of the program
int greyDiff[] = {whiteArray[0] - blackArray[0], whiteArray[1] - blackArray[1],
  whiteArray[2] - blackArray[2]};//range for values

//calibration regression coefficients
double arrayX1[] = {2.2194214917, 2.1882544837, 2.1099261184};
double arrayX2[] = {-0.0081525719, -0.0083532605, -0.0081855873};
double arrayX3[] = {0.0000132053, 0.0000144655, 0.0000150248};

//variable declared
float timeRead=0.1; //time delay between each reading
int nRead=200; //numbers of readings in getReading() function
int ohm=300; //time delay for 2 opamp circuit
int wait=50; //time delay beetween 2 different serial messages arduino-->processing
int rest=2000; //rest time after detection or white calibration


void setup(void){
  //setup the outputs for the colour sensor
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(photodiode, INPUT);
  Serial.begin(9600);
}

void loop(void){
  //receive serial messages from Processing
  if(Serial.available()){
    string = Serial.readStringUntil('\n');//read serial message from processing
    if(string == "COLOR"){ //until the end character "\n"
      string="";
      checkColour();
      printColour();
    }
  }
}

```

```
50
51
52 //colour detection
53 void checkColour(){
54   Serial.println("colorStart");
55   for(int i=0; i<=2; i++){
56     delay(wait); //delay to avoid serial messages errors
57     digitalWrite(ledArray[i],HIGH); //turn on one led at a time
58     Serial.println(rgbArray[i]); //Processing will colours the graphic led
59     delay(ohm); //delay for the photodiode time response
60     getReading(nRead); //nRead is the number of readings
61     colourArray[i] = avgRead; //save average reading in the array
62
63     //calculating the colour outcome
64     colourArray[i] = (colourArray[i] - blackArray[i])/(greyDiff[i])*255;
65     colourArray[i] = arrayX1[i]*colourArray[i] + arrayX2[i]*colourArray[i]*colourArray[i]
66       + arrayX3[i]*colourArray[i]*colourArray[i]*colourArray[i];
67
68     if(colourArray[i]>255) colourArray[i]=255; //avoid sensor errors
69     if(colourArray[i]<0) colourArray[i]=0;
70     digitalWrite(ledArray[i],LOW); //turn off the current LED
71     Serial.println("f"); //turn off the graphic led on processing
72   }
73   delay(wait);
74   Serial.println("colorDone");
75 }
76
77 //send to Processing colour outcomes
78 void printColour(){
79   Serial.print(round(colourArray[0])); //transform float value in int
80   Serial.print(",");
81   Serial.print(round(colourArray[1]));
82   Serial.print(",");
83   Serial.print(round(colourArray[2]));
84   for(int i=0; i<=2; i++){
85     delay(rest/3);
86     if(i==0) Serial.println("zz"); //second "Z"
87     else if(i==1) Serial.println("zzz"); //third "Z"
88     else if(i==2) Serial.println("zzzEnd"); //rubber the led graphic
89   }
90 }
91
92 //take times reading and calculate avgRead
93 void getReading(int times){
94   reading=0; //inizialize reading
95   for(int j=0; j<times; j++){
96     reading += analogRead(photodiode);
97     delay(timeRead); //timeRead for each reading
98   }
99   avgRead = (reading/times); //calculate the average and set it
100 }
```

Listing 2: Processing code

```
import processing.serial.*;
PFont fontA; //text font dim 35
PFont fontB; //for colour button
int wRed=255, wGreen=255, wBlue=255;//starting from white
int x1=650; //coordinate for detection messages
int y1=102; //coordinate for detection messages
int x2=750; //coordinate for colour detected rgb
int y2=650; //coordinate for colour detected rgb
int off1=50; //space beetween two text rows messages
int off2=50+17; //space beetween two text rows detection rgb
int count1=0; //count variable for rubberMsg() function
int count2=0; //count variable for rubberRGB() function
int rest2=0; //rest variable for rubberRGB() function
int ri=30, ro=250; //internal and external radius of the wheel

int x_center_res=1425; //reset button dimensions
int y_center_res=925;
int radius_res=50;

//colour button
int height_c=6;
int color_transparency=170;
int y=155;

Serial port;
String arduinoMsg; //reads all messages from arduino
String instruction; //saves "menu changes"
String RGB_components; //saves RGB values

int i; //for() cycles
//float w; //used in colourWheel()
float abs,s; //used for the rect under the wheel
color col; //used for the rect under the wheel
color c; //used for draw the wheel
float H,S,B; //hue, saturation, brightness for small circle in the wheel
PImage wheel; //colour's wheel
boolean buttonFlag=false; //blocks the user during functions

void setup(){
  size(1500,1000); //window dimension
  port = new Serial(this, "COM6", 9600);
  background(wRed,wGreen,wBlue); //colour the background of the window
  fontA = loadFont("SansSerif.plain-35.vlw");
  fontB = loadFont("FranklinGothic-Book-45.vlw");
  wheel = loadImage("wheel.jpg");
  textFont(fontA); //Set the font
  colourButton(true);
  resetButton(true);
  fill(50); //colour with the 50th shade of grey
  led();
  arduinoMsg();
```

```
52    detected();
53  }
54
55  void draw(){
56    if(port.available()>0) getInstruction(); //getting serial messages from arduino
57
58    //highlight the button if we pass the mouse over it
59    if(mouseX>300 && mouseX<500 && mouseY>80 && mouseY<180) colourButton(false);
60    else colourButton(true);
61    if((y_center_res+sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))>=mouseY &&
62      (y_center_res-sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))<=mouseY)
63        resetButton(false);
64    else resetButton(true);
65  }
66
67  void mousePressed(){
68    if(!buttonFlag){
69      if(mouseX>300 && mouseX<500 && mouseY>80 && mouseY<180){
70        port.write("COLOR\n");
71        buttonFlag=true;
72      }
73      else if((y_center_res+sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))>=mouseY &&
74        (y_center_res-sqrt(sq(radius_res/2)-sq(mouseX-x_center_res)))<=mouseY) reset();
75    }
76  }
77
78  void getInstruction(){
79    arduinoMsg = port.readString(); //reads all "menu changes" + RGB components detection
80    instruction = trim(split(arduinoMsg, '\n')[0]); //get only "menu changes"
81
82    if(instruction!=null){
83      if(instruction.equals("colorDone")){ //colour detection finished
84        RGB_components = trim(split(arduinoMsg, '\n')[1]); //RGB numerical values
85        rubberMsg(); //rubber function messages
86        fill(0,170,0);
87        text("The colour detection is done", x1,y1);
88        y1=y1+off1; //add space before next row
89        wRed = int(split(RGB_components,',')[0]); //taking RGB values
90        wGreen = int(split(RGB_components,',')[1]);
91        wBlue = int(split(RGB_components,',')[2]);
92        detected(); //draw new colour in the detecting zone
93        rubberRGB(); //rubber function colour detected
94        fill(127);
95        text("R:"+wRed, x2, y2-10); //-10 to center the text with colour rect
96        text("G:"+wGreen, x2+150, y2-10);
97        text("B:"+wBlue, x2+300, y2-10);
98        fill(wRed,wGreen,wBlue);
99        rect(x2+450, y2-50+5, 150, 50-5, 10); //colour rectangle next to RGB values
100       y2=y2+off2;
101       restLed(1);
102     }
103     else if(instruction.equals("f")){ //graphic led off
104       fill(50);
```

```
105         led();
106       }
107       else if(instruction.equals("r")){ //graphic led red
108         fill(255,0,0);
109         led();
110       }
111       else if(instruction.equals("g")){ //graphic led green
112         fill(0,255,0);
113         led();
114       }
115       else if(instruction.equals("b")){ //graphic led blue
116         fill(0,0,255);
117         led();
118       }
119       else if(instruction.equals("zz")) restLed(2); //graphic led rest zz
120       else if(instruction.equals("zzz")) restLed(3); //graphic led rest zzz
121       else if(instruction.equals("zzzEnd")){ //graphic led rest off
122         restLed(0);
123         buttonFlag=false;
124       }
125       else if(instruction.equals("colorStart")){
126         rubberMsg();
127         fill(70);
128         text("The colour detection is started", x1, y1);
129         y1=y1+off1;
130       }
131     }
132 }
133
134 //"delete" the past words and writes new one
135 void detected(){
136   colourWheel();
137
138   //draw the detection circle in the wheel
139   translate(width/4, 520);
140   c = color(wRed,wGreen,wBlue);
141   H = hue(c);
142   S = saturation(c);
143   B = brightness(c);
144   rotate(TWO_PI/255*H);
145   fill(c);
146   strokeWeight(3);
147   circle(float(ro-ri)/255*B+ri,0,25);
148   strokeWeight(1);
149   rotate(-TWO_PI/255*H);
150   translate(-width/4,-520);
151
152   //detected colour zone (vertical RGB + detected colour big rect)
153   fill(wRed,wGreen,wBlue);
154   rect(1125, 300, 300, 255, 10); //rect of the detected colour
155   fill(255);
156   noStroke(); //starting eliminate the stroke
157   rect(725,250,350,300+25); //rubber for rgb vertical rects
```

```
158   fill(0);
159   text(wRed, 725, 280);
160   text(wGreen, 850, 280);
161   text(wBlue, 975, 280);
162   fill(235);
163   rect(750,300,6,255); //RGB verical rects
164   rect(875,300,6,255);
165   rect(1000,300,6,255);
166   fill(255,80,80);
167   rect(750,555−wRed,6,wRed); //red cursor
168   circle(753,555−wRed,20);
169   fill(80,255,80);
170   rect(875,555−wGreen,6,wGreen); //green cursor
171   circle(878,555−wGreen,20);
172   fill(80,120,255);
173   rect(1000,555−wBlue,6,wBlue); //blue cursor
174   circle(1003,555−wBlue,20);
175
176   //colour the rect under the wheel
177   abs = 500.0/255.0; //500 is the width of the rectangle
178   i = 0;
179   colorMode(HSB);
180   for (s = 0; s <= 255; s++) {
181     col = color(H,s,B);
182     stroke(col);
183     fill(col);
184     rect(125 + i∗abs, 880, abs, 30);
185     i++;
186   }
187   colorMode(RGB);
188
189   //draw the triangle used as a pointer
190   stroke(50); //return to stroke
191   fill(60);
192   triangle(125+abs∗S,874, 125+abs∗S−8,860, 125+abs∗S+8,860);
193 }
194
195 //reset processing
196 void reset(){
197   //erase colour detected hystory
198   fill(255);
199   noStroke();
200   rect(750,600+3,620,325);//delete colour detected hystory
201   stroke(50);
202   count2=0;
203
204   //erase arduino messages
205   arduinoMsg();
206   count1=0;
207   y1=102;
208
209   //resetting from the beginning
210   wRed=255; wGreen=255; wBlue=255;
```

```
211   detected();
212 }
213
214 //erase messages in the upper right when it's needed
215 void rubberMsg(){
216   count1++;
217   if(count1>2){
218     arduinoMsg();
219     count1=1;
220     y1=102;
221   }
222 }
223
224 //erase messages of RBG hystory
225 void rubberRGB(){
226   count2++;
227   rest2=count2%5;
228   if(rest2==1) y2=650; //when i need to start from the first row
229   fill(255);
230   noStroke();
231   rect(750,y2-50+3,600+3,100+35);
232   stroke(50);
233 }
234
235 //draw the "zzz" during the rest phase
236 void restLed(int mode){
237   fill(255);
238   noStroke();
239   rect(0, 0, 250, 275);//erase led
240   stroke(50);
241   fill(50);
242   led();
243   if(mode==1||mode==2||mode==3){
244     textFont(fontA,50);
245     fill(50);
246     text("Z",75,75);
247   }
248   if(mode==2||mode==3) text("z",45,50);
249   if(mode==3){
250     textFont(fontA,30);
251     text("z",25,30);
252     textFont(fontA);
253   }
254 }
255
256 //draw the colour wheel and the rectangle below
257 void colourWheel(){
258   fill(255);
259   noStroke();
260   rect(width/4-ro-25,520-ro-15,550,550);
261   image(wheel,width/4-ro,520-ro,500,500);
262   /*
263   translate(width/4, 520);
```

```
264   fill(255);
265   noStroke();
266   rect(-270,-265,540,540);//white rect below the color wheel to erase the last point
267   colorMode(HSB, TWO_PI, 100, 100);//start HSB colour mode
268   strokeWeight(2);//if not, we will have empty space in the wheel
269   for (w = 0; w <= TWO_PI; w +=0.005) {//drawing the wheel
270     push();
271     rotate(w);
272     for (i = ri; i< ro; i++) {
273       stroke(w, 100, map(i, ri, ro, 0, 100));
274       point(i, 0);
275     }
276     pop();
277   }
278   strokeWeight(1); //return to normal strokeWeight
279   colorMode(RGB,255,255,255);//return to RBG colour mode
280   translate(-width/4, -520);
281   */
282   rect(110,855,555,30); //erase the past triangle cursor on the saturation bar
283   stroke(50);
284   fill(255);
285   rect(120, 875, 510, 40); //saturation bar initialization
286   /*
287   fill(0,0);
288   circle(125+ro,520,2*ro); //stroke of the wheel
289   */
290 }
291
292 //draw a led on the upper sx side
293 void led(){
294   stroke(50);
295   arc(150, 80, 70, 70, -PI, 0);
296   noStroke();
297   rect(115, 80, 70, 50);
298   stroke(50);
299   line(115, 80, 115, 130);
300   line(185, 80, 185, 130);
301   rect(100, 130, 100, 20);
302   fill(127);
303   rect(130, 150, 10, 100);
304   rect(160, 150, 10, 80);
305 }
306
307 void resetButton(boolean over){
308   if(over) fill(255,0,0);//normal button
309   else fill(225,0,0);
310   circle(x_center_res, y_center_res, radius_res);
311 }
312
313 void arduinoMsg(){
314   fill(255);
315   rect(600,50,820,130,20);
316 }
```

```
317
318 void colourButton(boolean over){
319   if(over) fill(255);//normal button
320   else fill(240);
321   rect(300, 80, 200, 100, 10);
322   noStroke();
323   fill(255,0,255,color_transparency);//fucsia
324   rect(332+1, y, 5, height_c, 5,0,0,5);//first
325   fill(192,0,255,color_transparency);//magenta
326   rect(332+8, y, 5, height_c);
327   fill(128,0,255,color_transparency);//violetto scuro
328   rect(332+15, y, 5, height_c);
329   fill(0,0,255,color_transparency);//blu
330   rect(332+22, y, 5, height_c);
331   fill(0,128,255,color_transparency);//blu manganese
332   rect(332+29, y, 5, height_c);
333   fill(0,192,255,color_transparency);//ceruleo chiaro
334   rect(332+36, y, 5, height_c);
335   fill(0,255,255,color_transparency);//turchese
336   rect(332+43, y, 5, height_c);
337   fill(0,255,192,color_transparency);//verde acqua
338   rect(332+50, y, 5, height_c);
339   fill(0,255,128,color_transparency);//id
340   rect(332+57, y, 5, height_c);
341   fill(0,255,0,color_transparency);//verde
342   rect(332+64, y, 5, height_c);
343   fill(128,255,0,color_transparency);//verde foglia
344   rect(332+71, y, 5, height_c);
345   fill(192,255,0,color_transparency);//giallo acido zolfo
346   rect(332+78, y, 5, height_c);
347   fill(255,255,0,color_transparency);//giallo
348   rect(332+85, y, 5, height_c);
349   fill(255,224,0,color_transparency);//senape
350   rect(332+92, y, 5, height_c);
351   fill(255,192,0,color_transparency);//oro
352   rect(332+99, y, 5, height_c);
353   fill(255,160,0,color_transparency);//arancio
354   rect(332+106, y, 5, height_c);
355   fill(255,128,0,color_transparency);//id
356   rect(332+113, y, 5, height_c);
357   fill(255,96,0,color_transparency);//id
358   rect(332+120, y, 5, height_c);
359   fill(255,64,0,color_transparency);//pesca scuro
360   rect(332+127, y, 5, height_c);
361   fill(255,0,0,color_transparency);//rosso
362   rect(332+134, y, 5, height_c, 0,5,5,0);//last
363   stroke(50);
364   fill(70);
365   textFont(fontB);
366   text("COLOR",333,140);
367   textFont(fontA);
368 }
```

References

[1] TAOS, "PROGRAMMABLE COLOR LIGHT-TO-FREQUENCY CONVERTER Texas Advanced Optoelectronic Solutions Inc . PROGRAMMABLE," *The LUMENOLOGY*, no. 972, pp. 1–10, 2009.