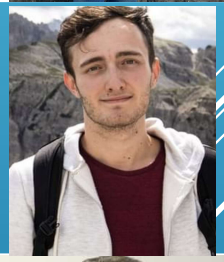
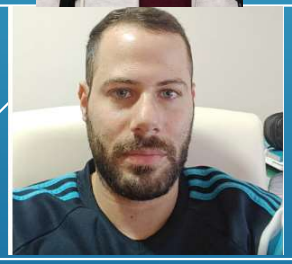
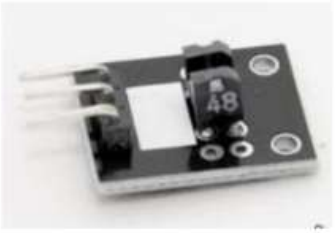
ELECTRONIC INSTRUMENTATION

HOMEWORK 02 – OPTICAL DISTANCE SENSOR-HARDWARE ONLY

	ID	Name	Photo of the student
Student #1	**316352210**	**Omri pony**	
Student #2	**316146836**	**Steve Reznikov**	
Student #3	**313601866**	**Or Salganik**	

Lecturer Dr. Samuel Kosolapov

OUR PROJECT CONTAIN THE FOLLOWING COMPONENTS:

Table 1. RGB Optical Distance Sensor. Components from Arduino 37 in 1 Sensor Kit		
1	Arduino KY-018 Photo resistor module	
2	Arduino KY-016 3-color LED module	
3	Arduino KY-010 Photo Interrupter Module (Explain operation)	

THESE ARE THE REAL WORLD COMPONENTS – BEFORE ASSEMBLING THEM ON THE MINI-BREADBOARD

Arduino uno R3 with prototype shield

bulb module –ky016

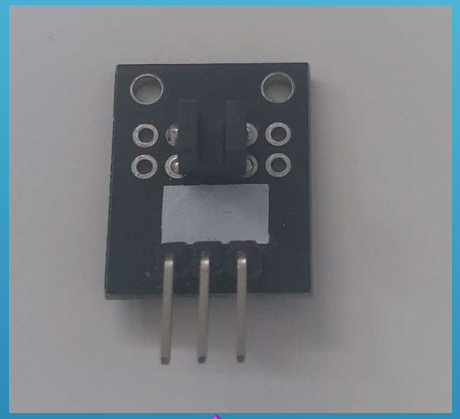


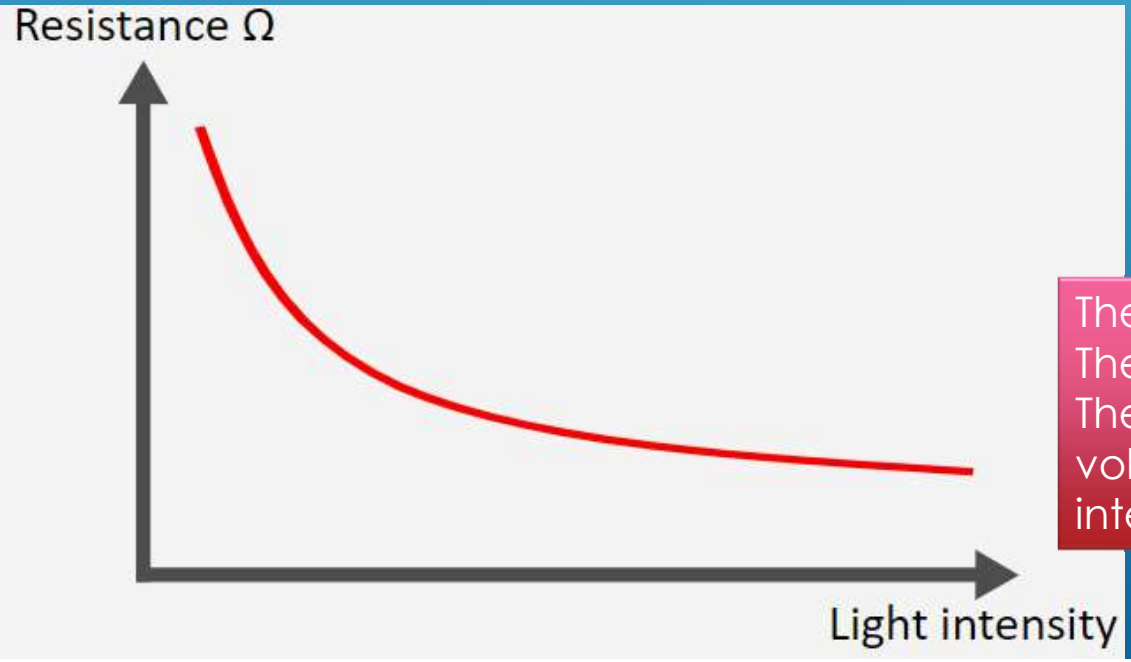
Photo interrupt module –ky010

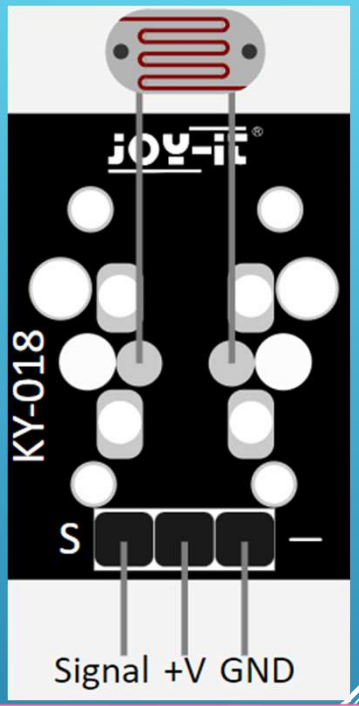
photoresistor module –ky018

THE PHOTORESISTOR MODULE - KY-018

This module contains an LDR resistor whose **resistance value decreases with brighter surroundings**.

LDR= light-dependent resistor

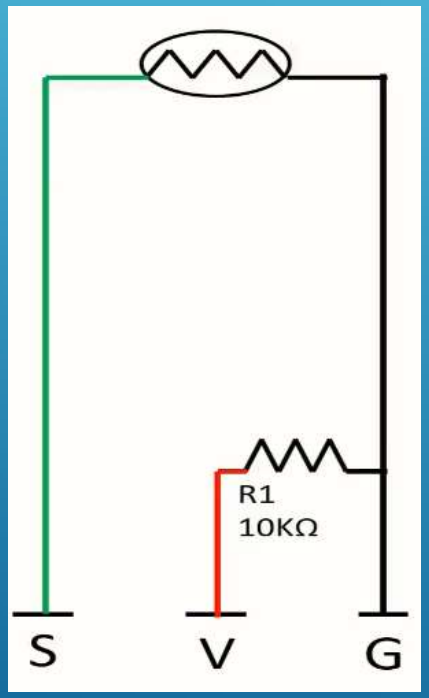


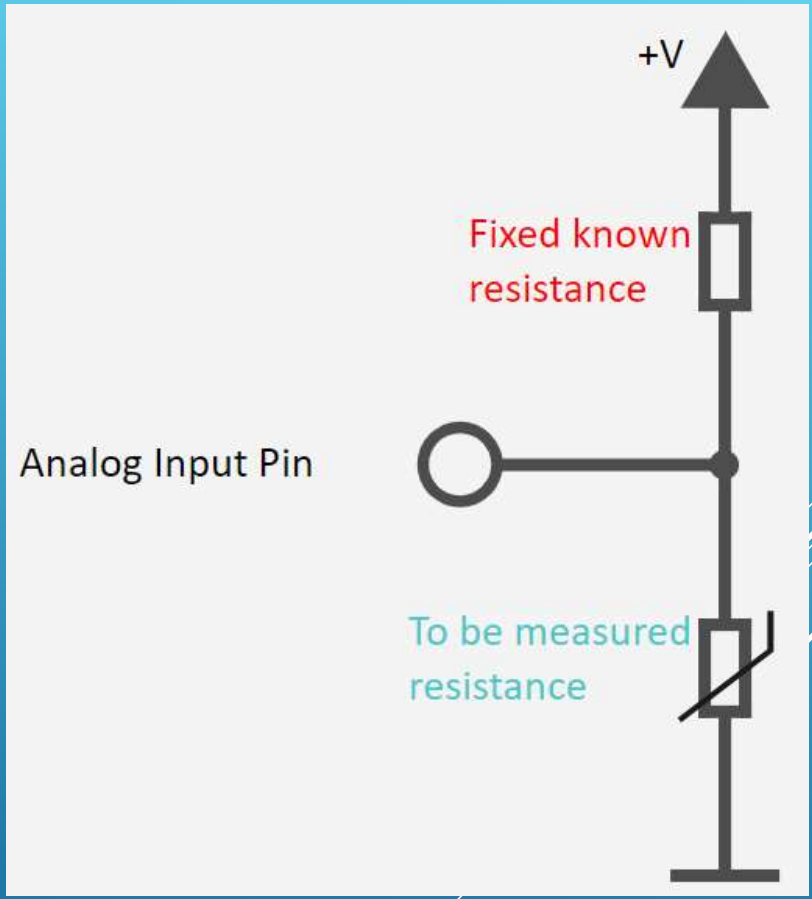


The **VCC** pin is used to provide power to the module.
The **GND** pin is connected to the ground of the Arduino.
The **signal** pin is the output pin that provides an analog voltage signal to the Arduino proportional to the intensity of light falling on the photoresistor.

THE PHOTORESISTOR MODULE - KY-018 - WORKING PRINCIPLE

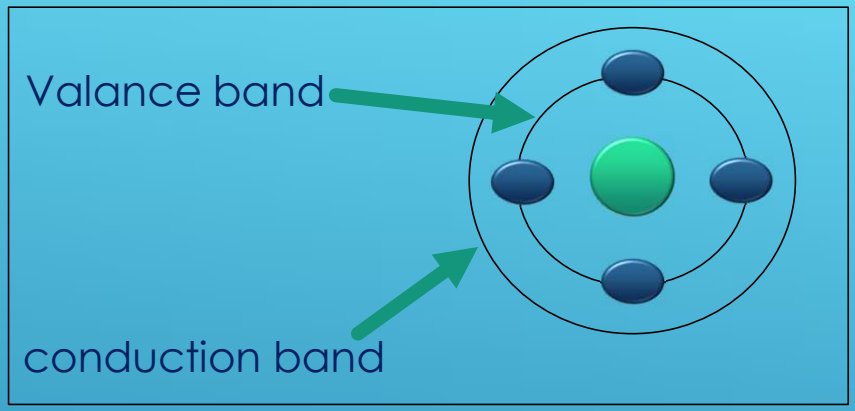
This resistance can be determined using a voltage divider, where a known voltage is divided across a known (10 kΩ) and an unknown (variable) resistance. Using this measured voltage, the resistance can then be calculated.





THE PHOTORESISTOR MODULE - KY-018 - PHYSICAL OPERATION EXPLANATION

The semiconductor material on the resistor works on the principle of Photoconductivity:



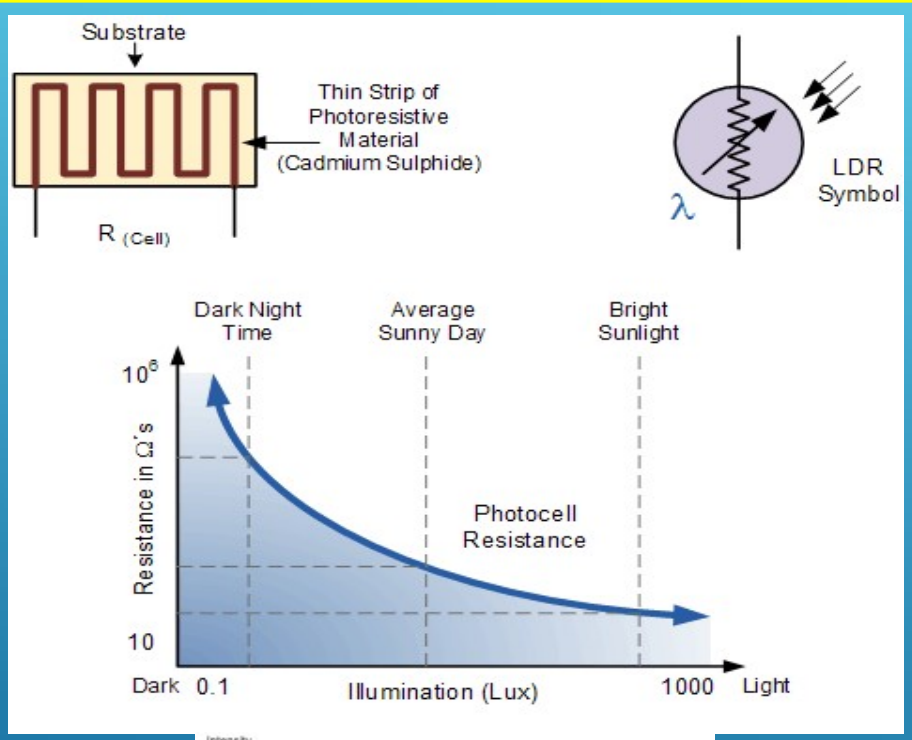
In the atom there are electrons in a low energy state stored in a band called the **valance band** – the atoms are bound to the band and cannot move.

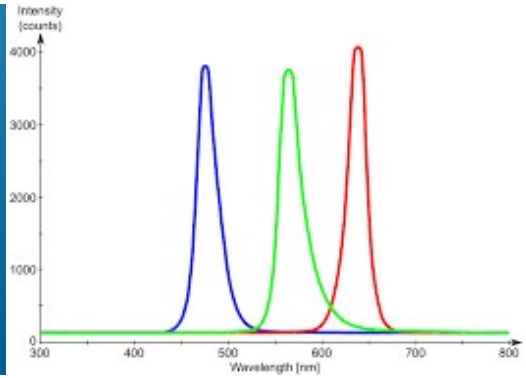
In the **conduction band** – electrons on him are free to move around.

Light is made of tiny energy packets called **photons** – then a light hit the semiconductor – the photons energizes the electrons located in the valance band – and forces them to jump to the conduction band – creating **current** in the circuit.

When there are no more photons-these electrons will fall back to the valance band – and current will stop.

THE PHOTORESISTOR MODULE - KY-018 – THE CRUCIAL ROLL OF THE PHOTORESISTOR IN THIS PROJECT





To calibrate the system and to measure the distance we want we will use the RED LED light, We also need to use photoresistor.

The relation between wavelength and energy:
$E = h * f$ when $\boldsymbol{h}$ is the Planck constant.
In addition $\lambda = \frac{f}{c}$ → $\boldsymbol{E = h * \lambda * c}$

It means that when the LED flashes – an energy in a specific wavelength is emitted (according to the color – every color has its our wavelength). In our case the photoresistor absorbs energy of **Red**(~665nm) LED.

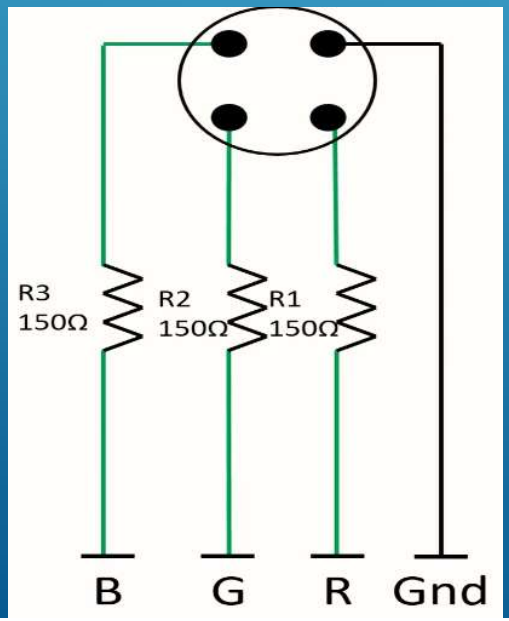
Therefore-we'll illuminated on white object using the RED LED for a "full reflection" and subtracted with the dark light value, in that way we manage somewhat to eliminate the back ground color.

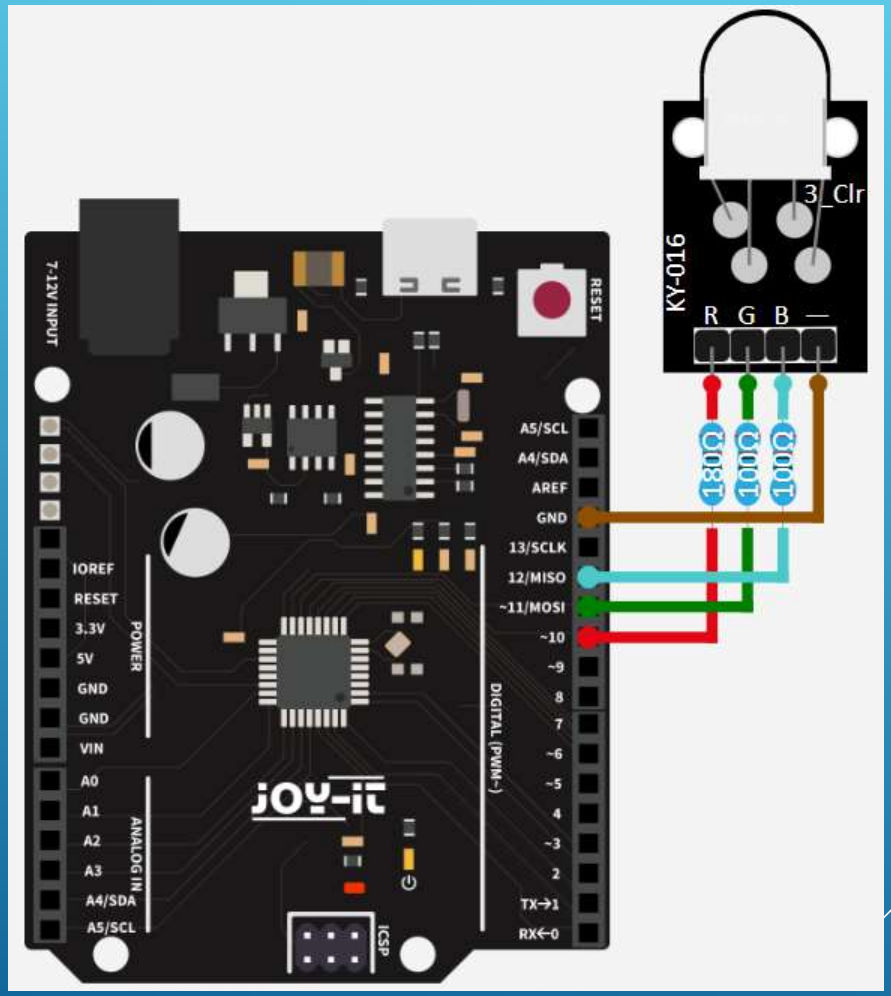
The role of the photo resistor is to absorb the energy and compute it to a usable value.

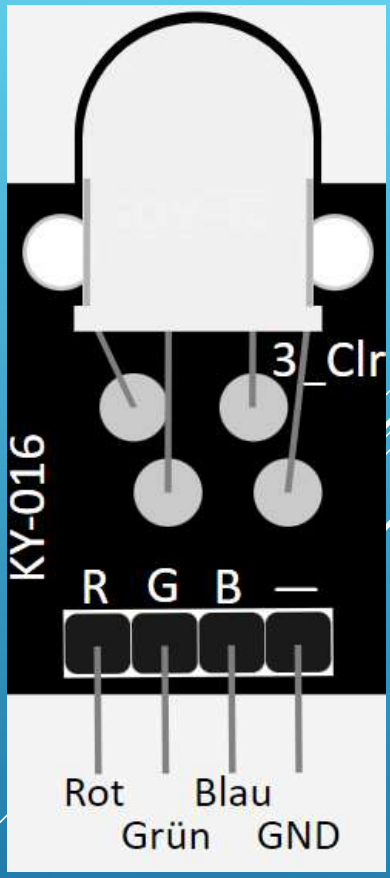
THE RGB LED MODULE - KY-016

LED module which contains a red, blue and green LED. These are connected to each other by means of a **common cathode**.

Forward voltage [Red]=1.8 V
Forward voltage [Green, Blue]=2.8 V
Forward current=20 mA



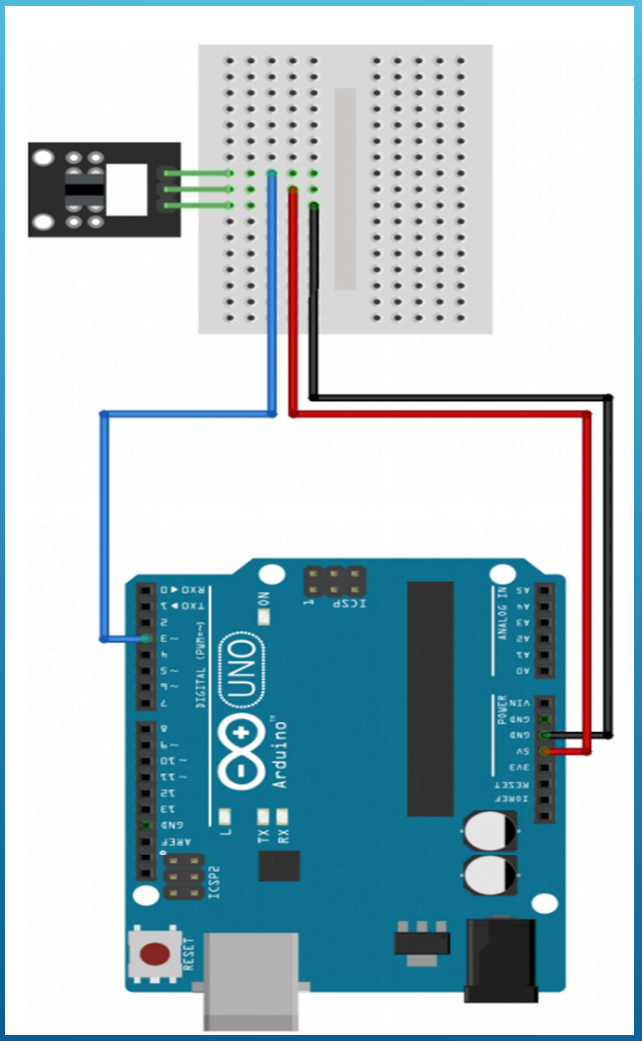


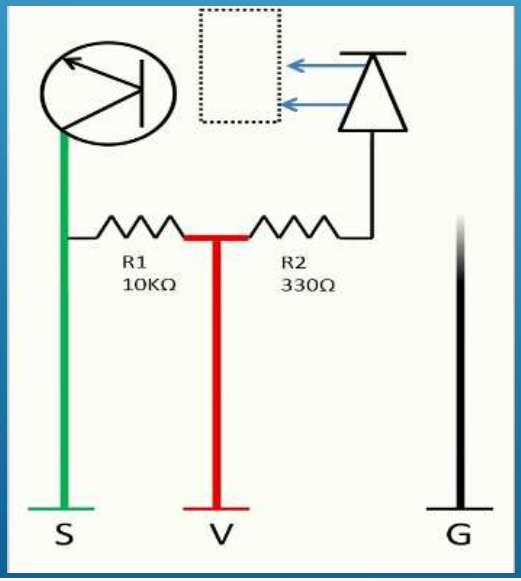


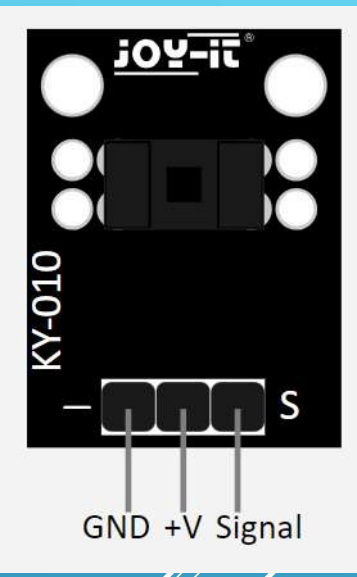
THE PHOTO INTERRUPT MODULE - KY-010

The KY-010 Photo Interrupter module is a switch that will trigger a signal when light between the sensor's gap is blocked.

This module consists of an optical emitter/detector and 3 male header pins on the front. On the back there are two resistors of 1kΩ and 33Ω.
The sensor uses a beam of light between the emitter and detector to check if the path between both is being blocked by an opaque object.

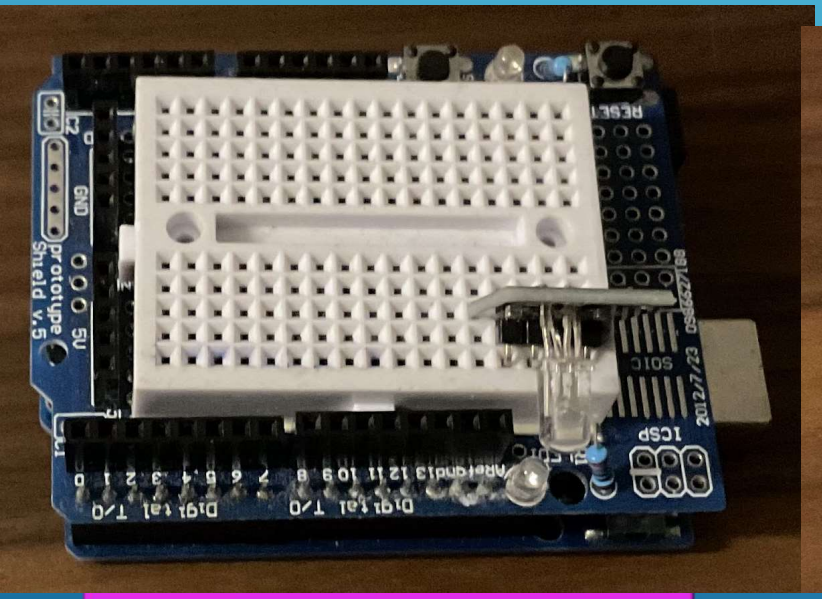




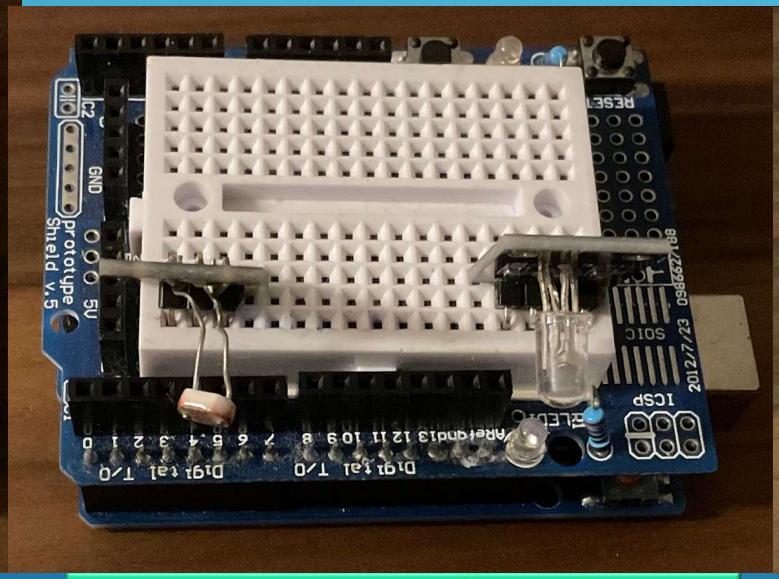


HW02.11-PRESENT THREE CLEAR PHOTOS OF THE MODULES POSITIONED ONE BY ONE ON THE BREADBOARD MINI WITHOUT WIRES

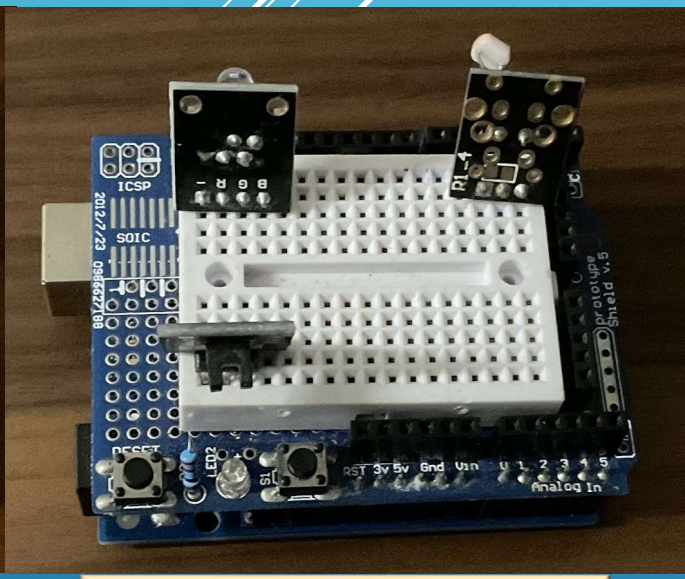
Figure 1 → **Figure 2** → **Figure 3**



Here we add the RGB LED to the mini-Breadboard.



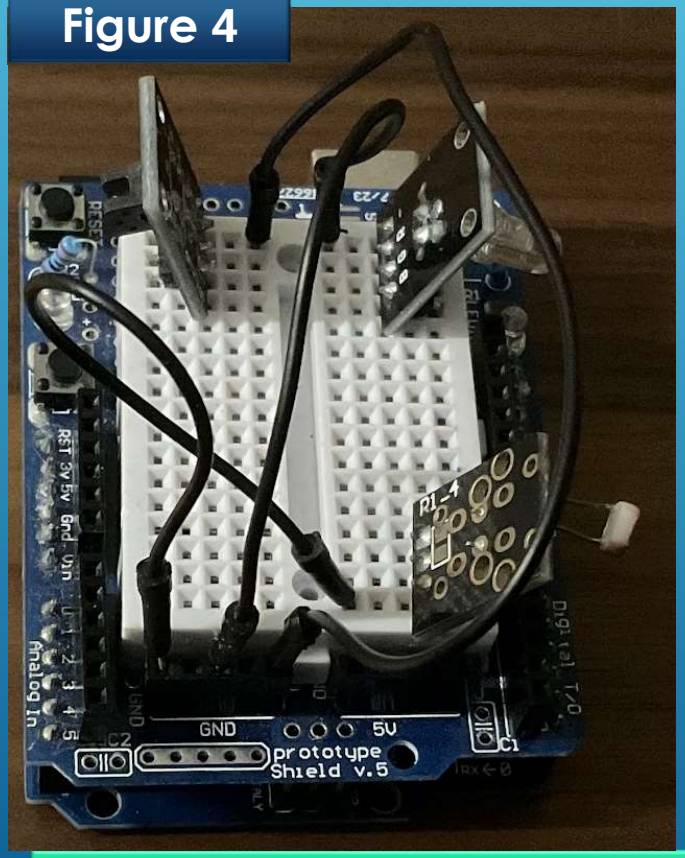
Here we add the photoresistor to the mini-Breadboard.



Here we add the photo-interrupter to the mini-Breadboard.

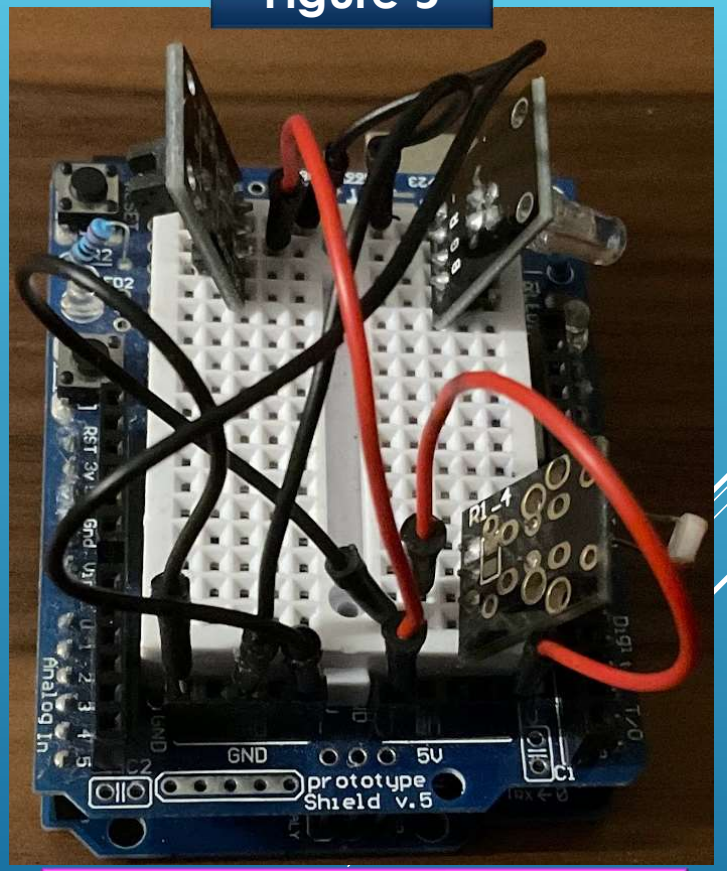
HW02.11-PRESENT FOUR CLEAR PHOTOS OF THE MODULES POSITIONED ON THE BREADBOARD MINI WITH ALL THE NECESSARY WIRES

Figure 4



GND connection between the breadboard and the Arduino.

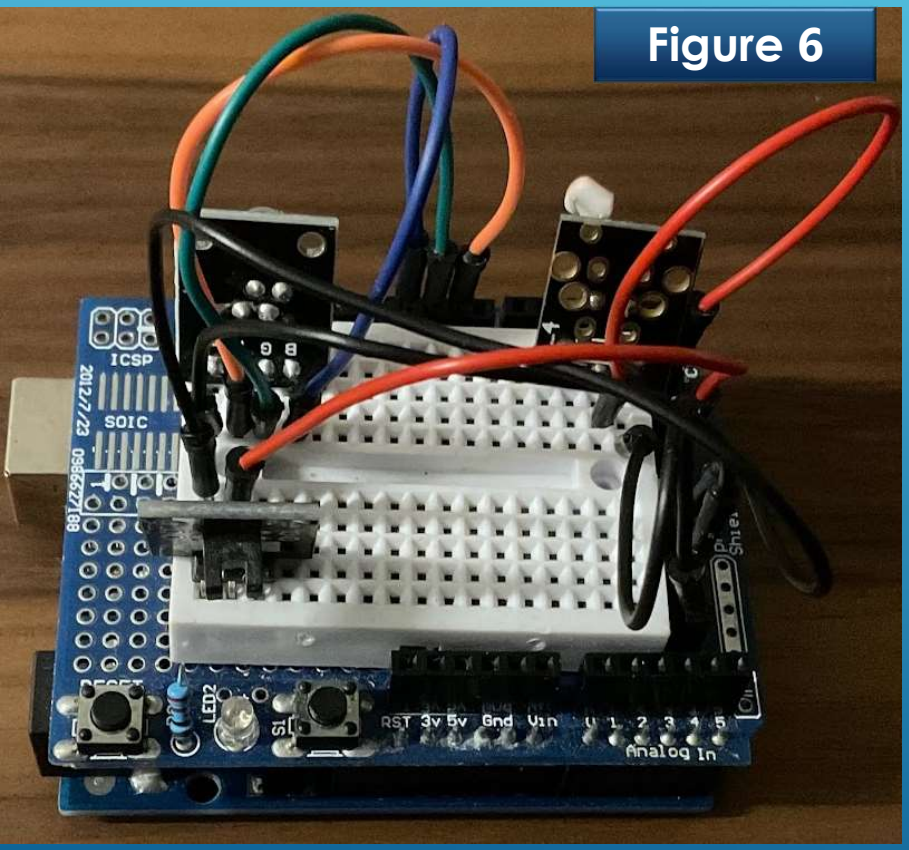
Figure 5



5V connection between the modules and the Arduino.

HW02.11-PRESENT FOUR CLEAR PHOTOS OF THE MODULES POSITIONED ON THE BREADBOARD MINI WITH ALL THE NECESSARY WIRES

Figure 6



Connections of the R G B pins to pins 9, 10 and 11, respectively.

Figure 7.1

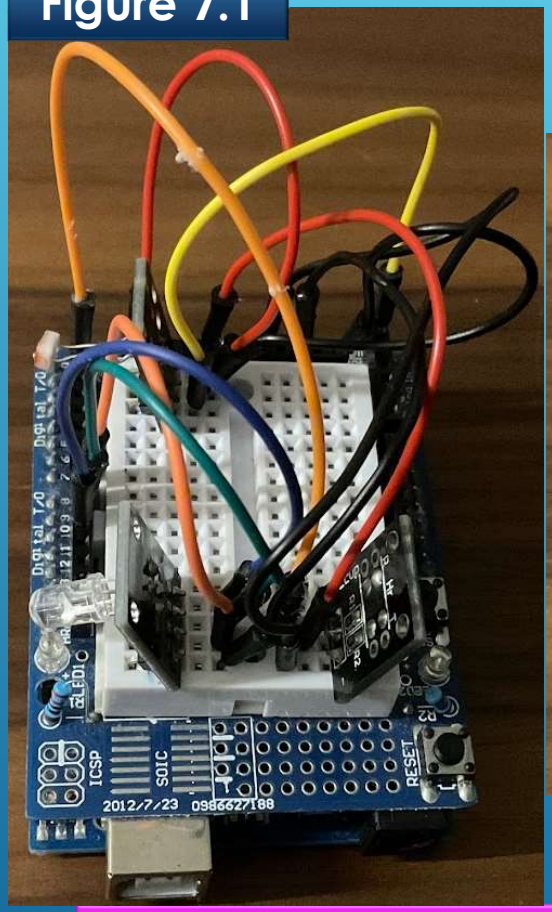
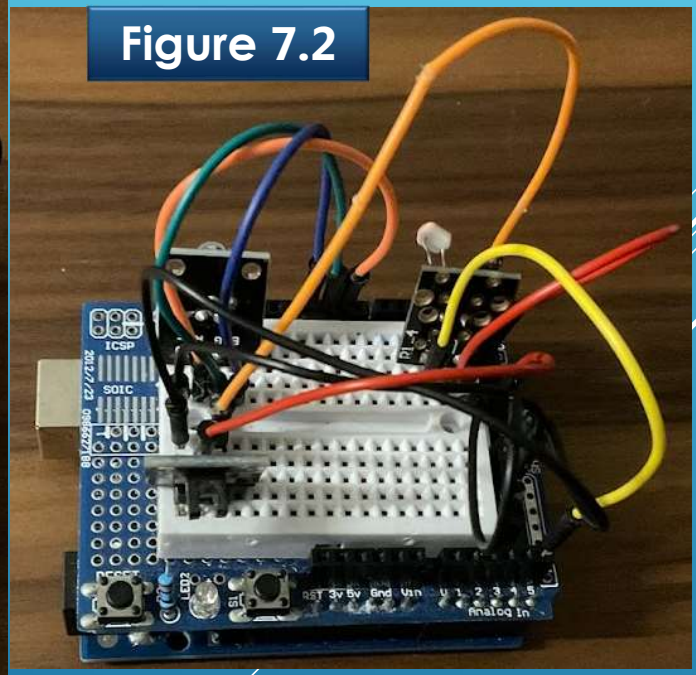


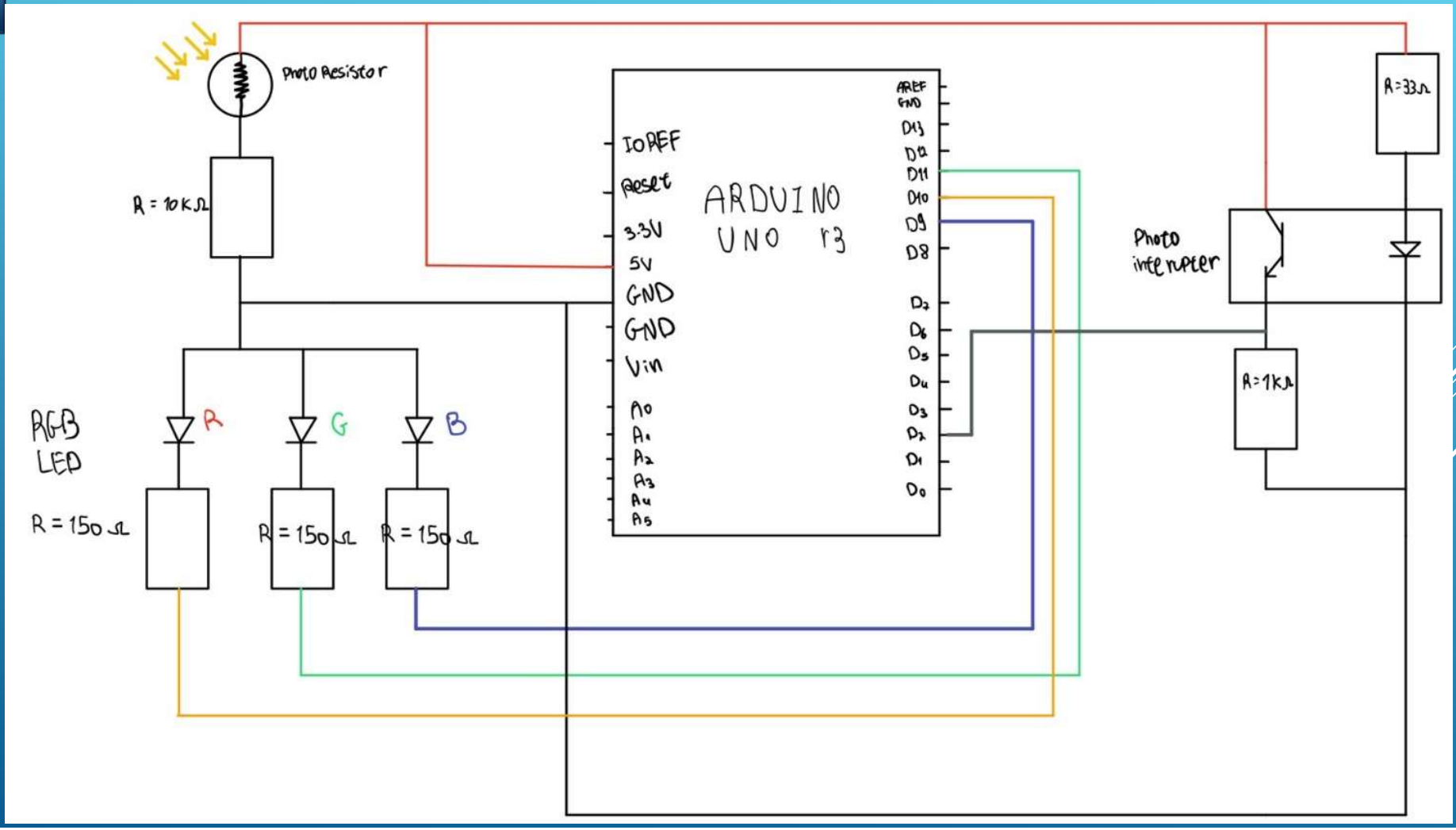
Figure 7.2



Connections of the photo-resistor signal to pin A5, And photo Interrupter to pin 2.

HW02.12-PRESENT SYSTEM' ELECTRICAL CIRCUIT THAT CONTAINS ALL COMPONENTS AND ALL CONNECTIONS

Figure 8



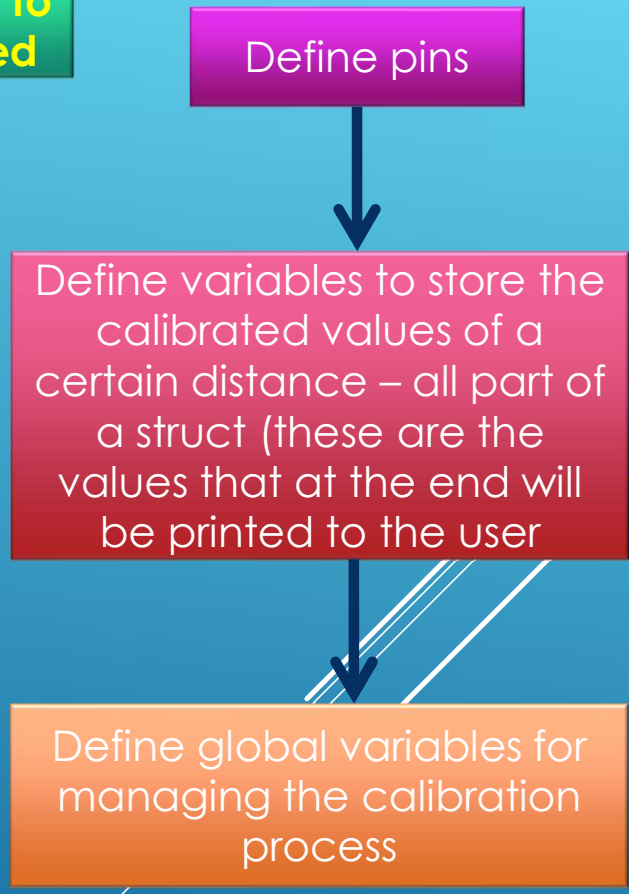
HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 1

In parts 1-4 of HW02.13 we'll create detailed flow chart in words and translate it to a basic flow chart. In parts 5-8 a "classic version" of flow chart will be presented

1. Start with defining constants `RED` as 10 and `photoResistorPin` as `A0`.

2. A `struct` named **`CalibrationObject`** is defined to store calibration values such as **`calibrationDarkValue`**, **`calibrationLightValue`**, and **`calibrationDelta`.** These values will be presented to the user after doing the calibration (on a specific distance) and he will have to insert only the calibrationDelta to the excel as will be explained later.

3. Global variables are declared including `LEDpin`, `photoInterrupterPin`, `tempDarkValue`, `tempLightValue`, `darkValue`, `lightValue`, `delta`, `incomingChar`, `current_color_idx`, and `colorCalibrationRequested`.



HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 2

5. The `photoInterrupterInterrupt()` function sets `colorCalibrationRequested` to `true` when the interrupt is triggered.

6. The `GetRGBdata()` function is used to read the photoresistor values in both dark and light conditions by toggling the LED.
- It takes 10 readings for both dark and light conditions and averages them.

7. The `DoCalibration()` function calls `GetRGBdata()` and stores the dark value, light value, and delta in the `calibrationObject`.

8. The `printCalibrationData()` function prints the calibration data for the red color including dark value, light value, and delta.

Define a function that gets activated when an interrupt is triggered – aka when the light between the "walls" of the module are blocked

Define another function to read photoresistor readings values in the dark and in light (aka when the LED flashes and between flashes).

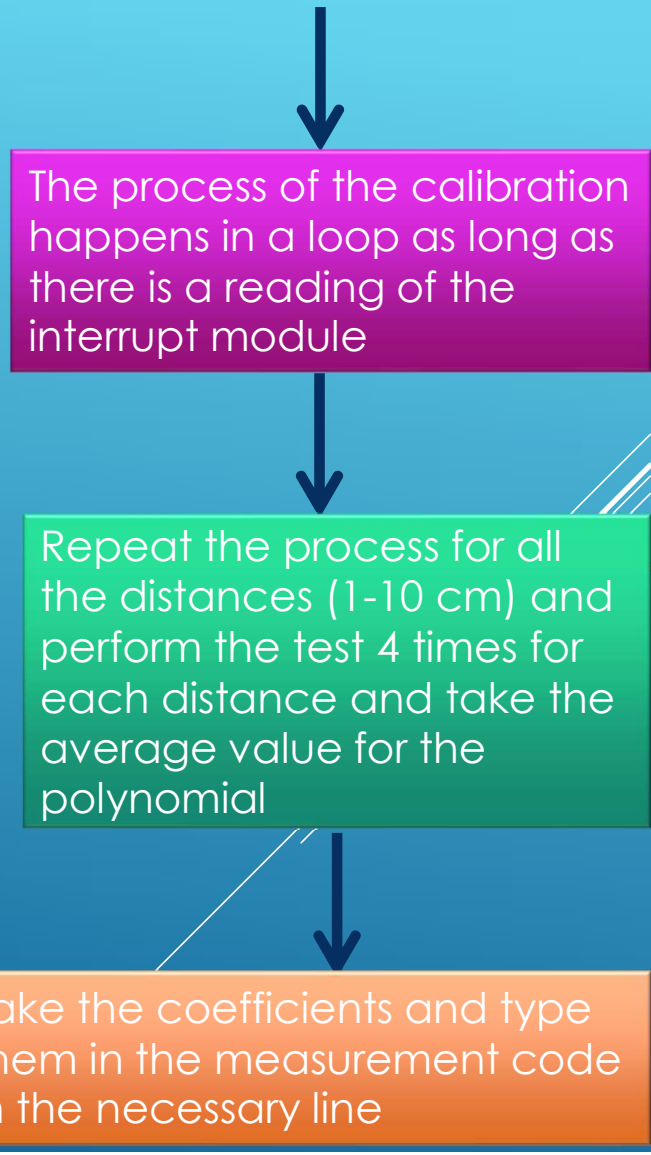
After measuring the values by the photoresistor – they will be stored in the variables we defined earlier
Then printing the values to the use of the user

HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 3

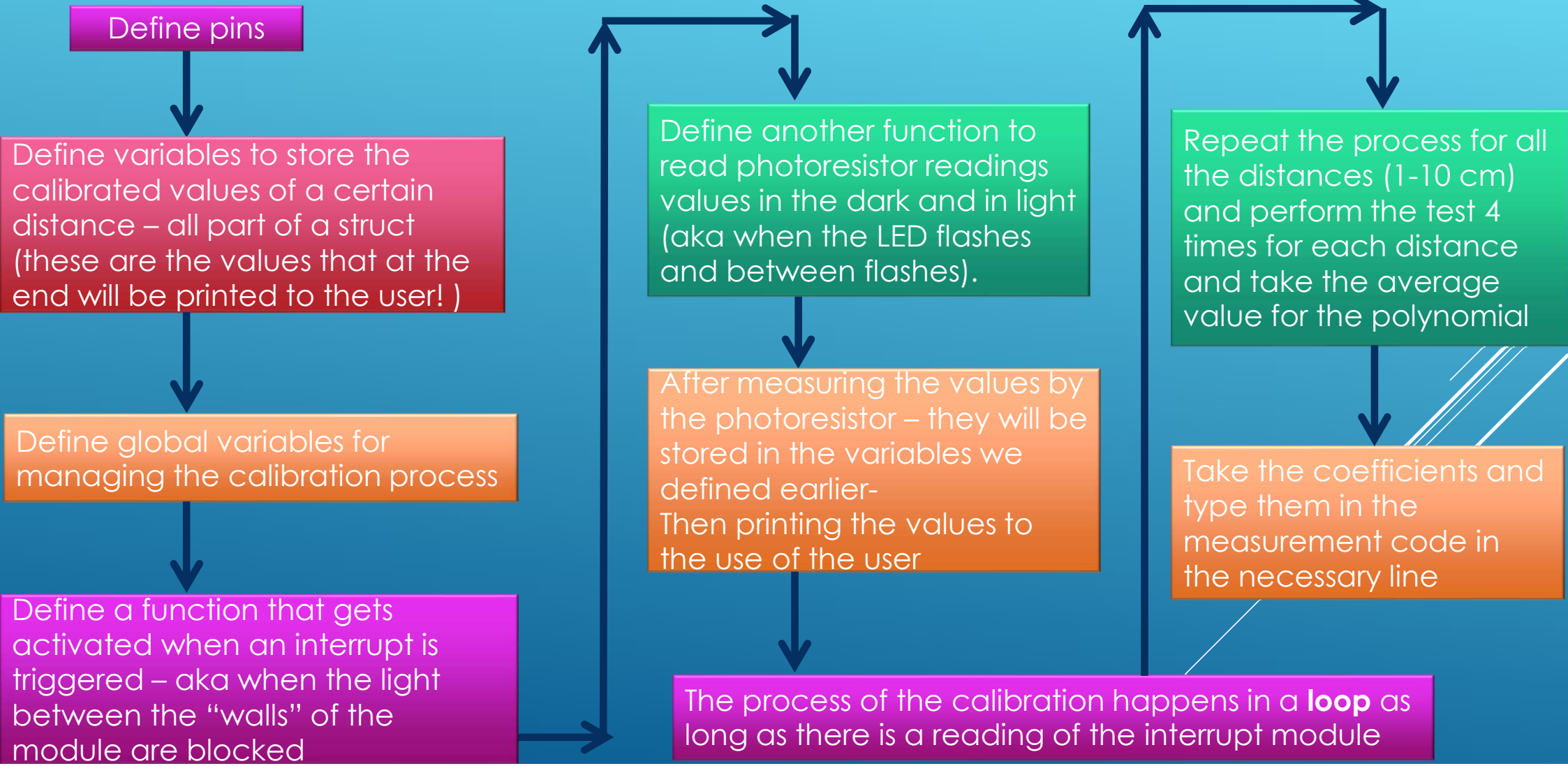
9. In the `loop()` function:
- If `colorCalibrationRequested` is `true`, calibration is performed by calling `DoCalibration()` and then the calibration data is printed using `printCalibrationData()`.

10. The collected data should be typed in the excel table.
For every distance the process should be done **4** times and then the average should be used. (4 measures for each distance → total number of 40 measures.)

11. After finish filling the excel calibration table – the process is done and the coefficients will be available to type in the measurement code!



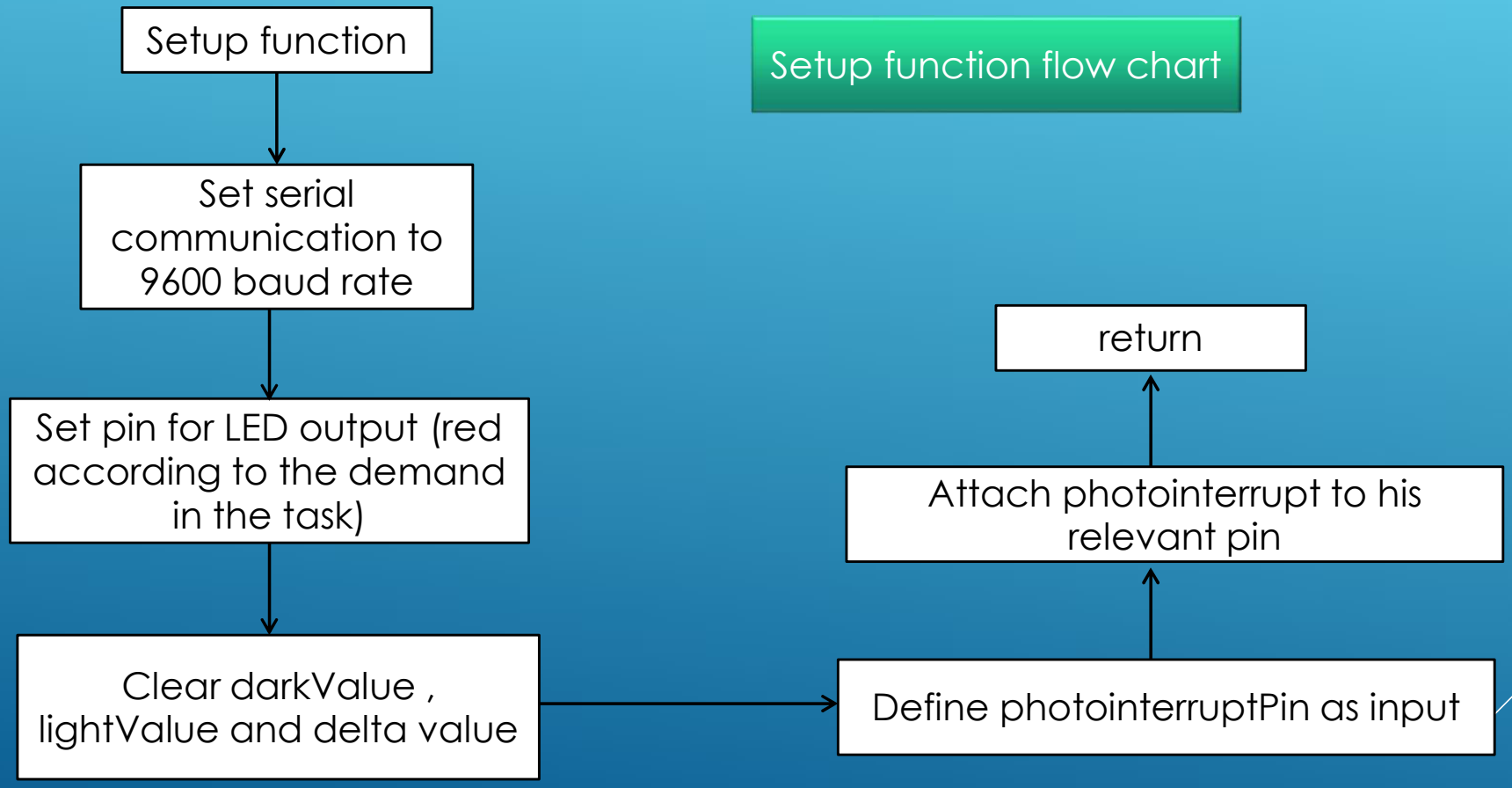
HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 4



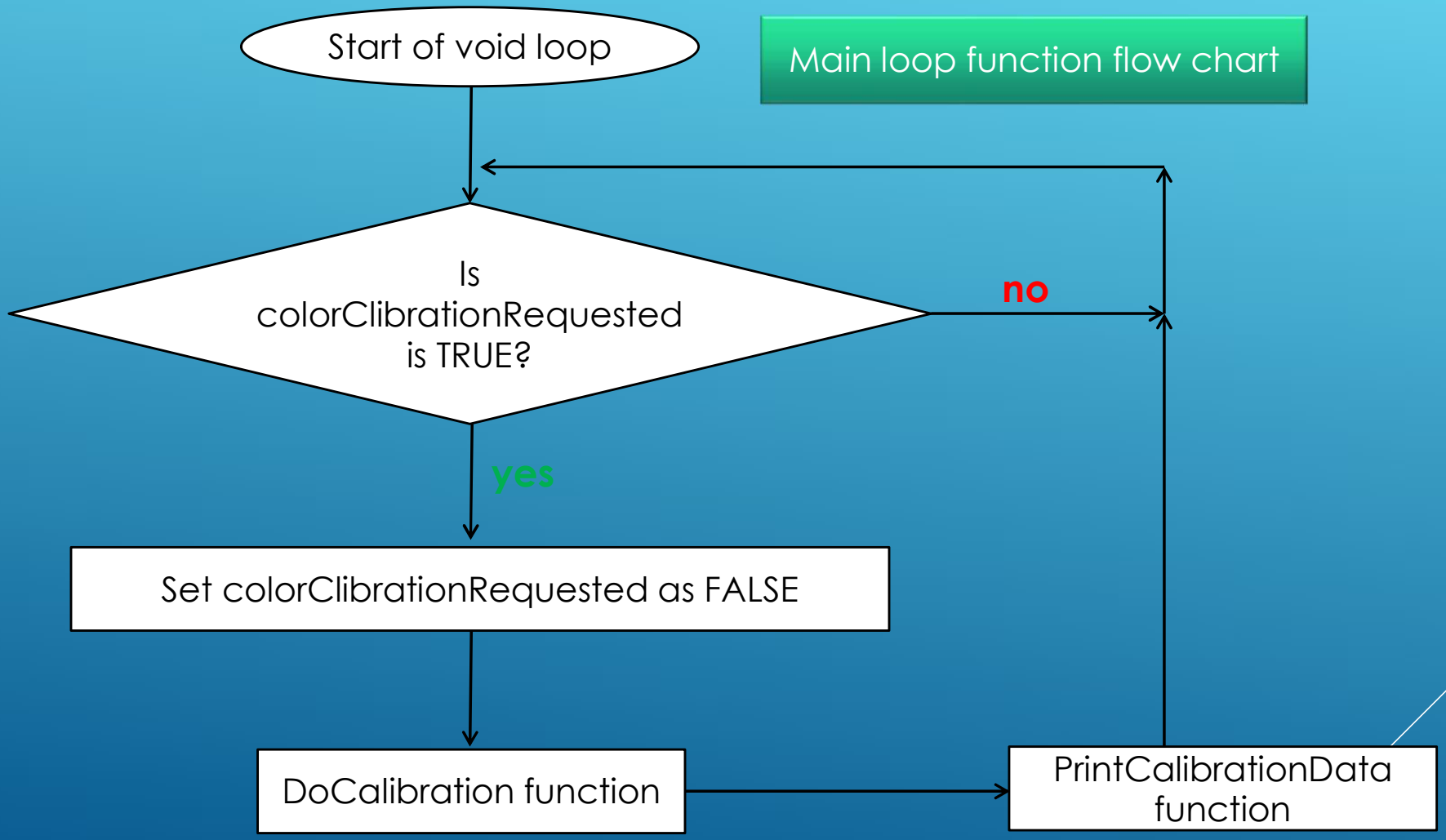
HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 5

As mentioned in part 1 of HW02.13 – in parts 5-11 will be presented a more "classic" flow chart of the code

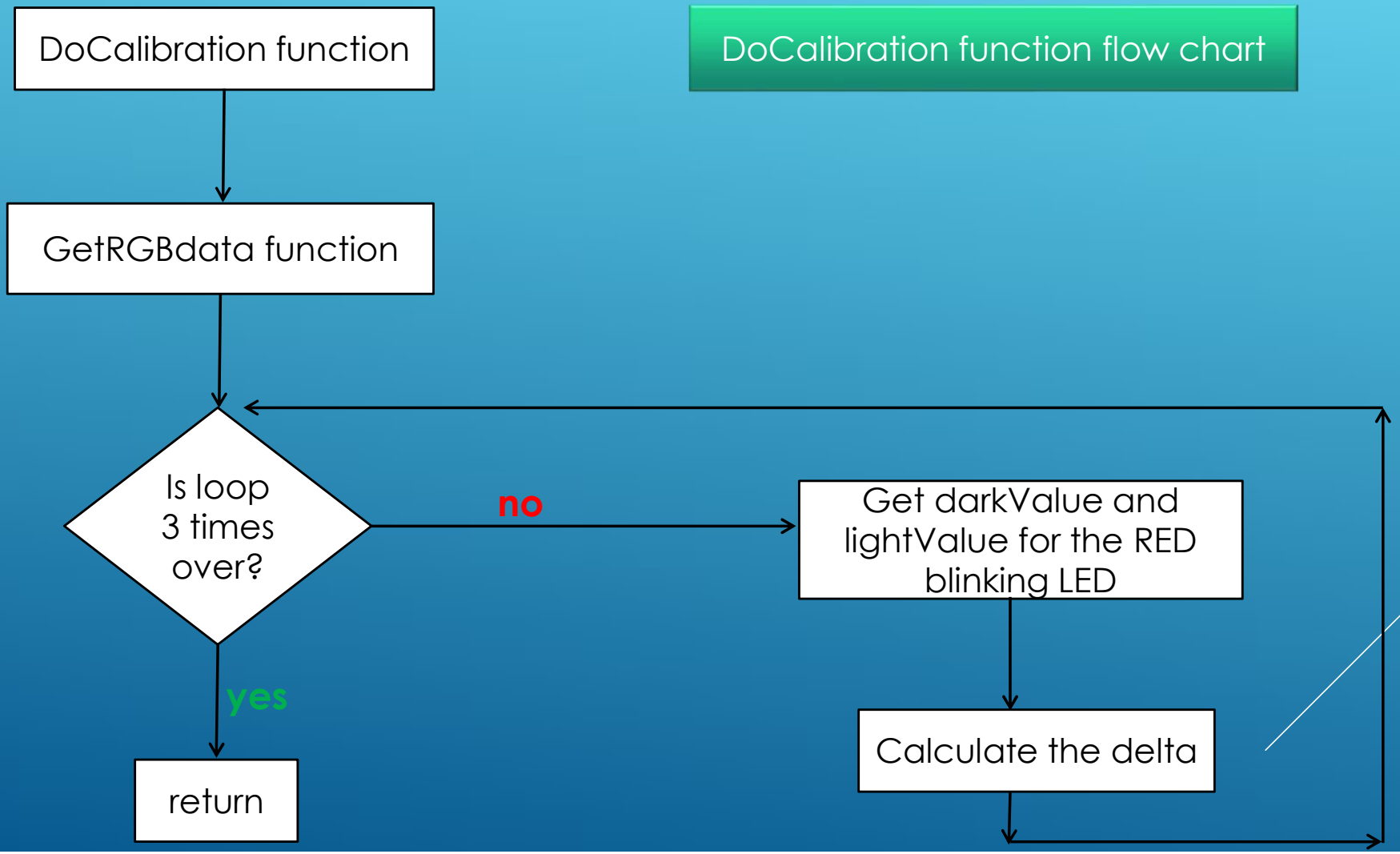
Setup function flow chart



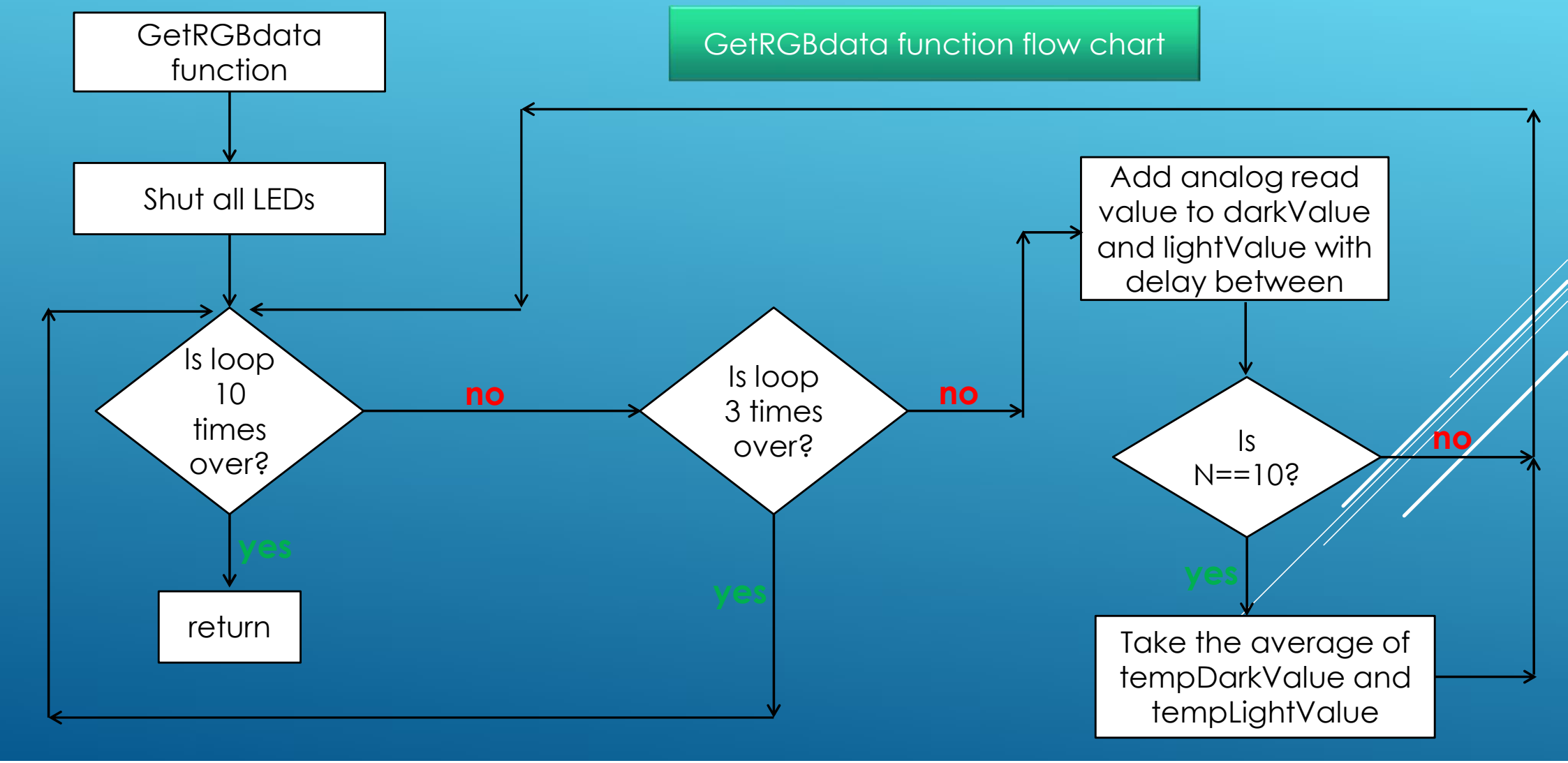
HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 6



HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 7

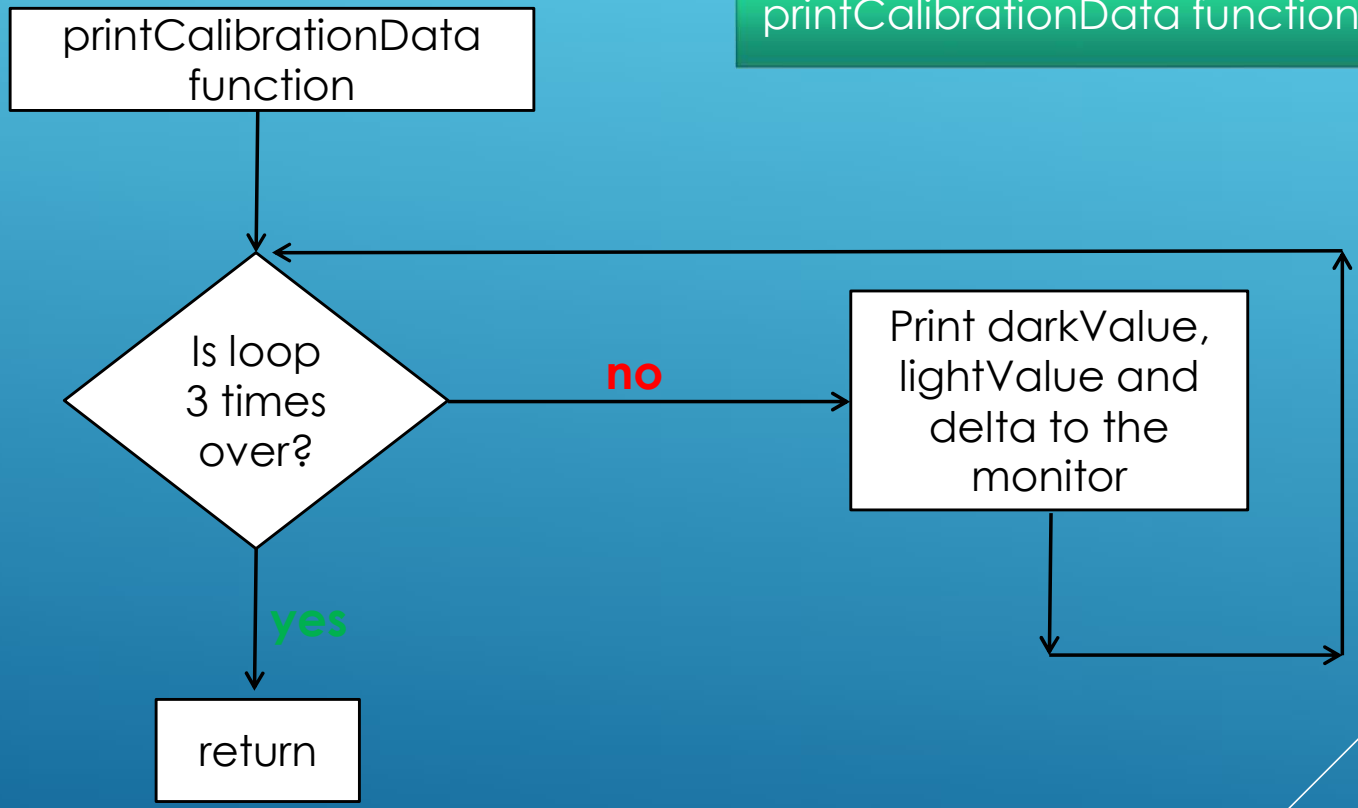


HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 8



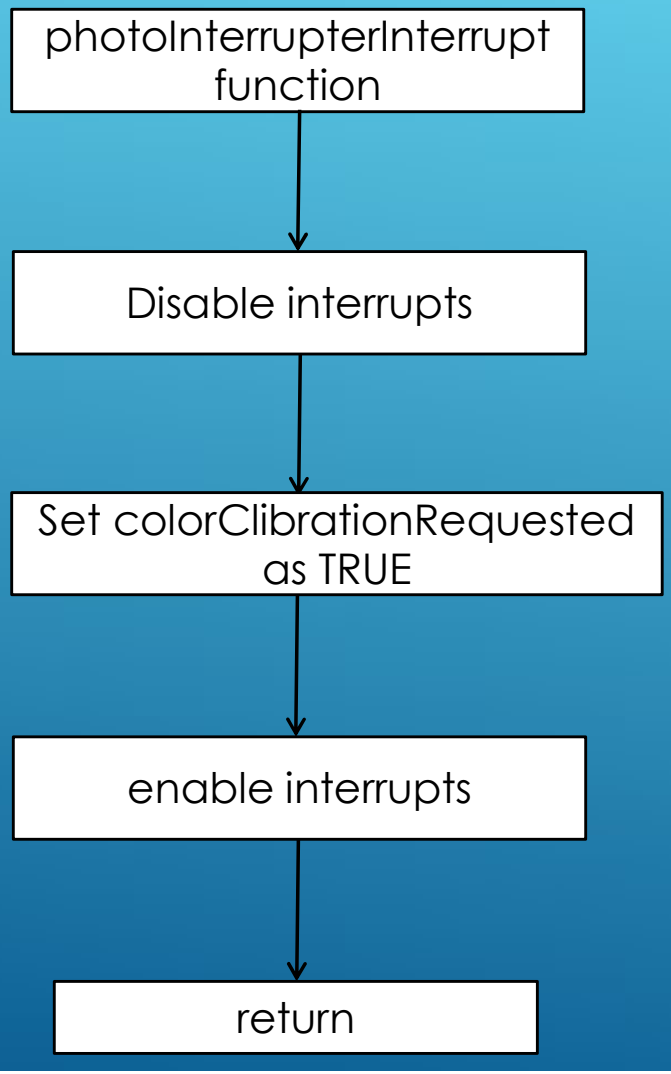
HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 9

printCalibrationData function flow chart



HW02.13-DETAILED DATA FLOW-CHART OF THE SYSTEM ' OPERATION DURING CALIBRATION-PART 10

interrupt function flow chart



HW02.14-FULL CODE FOR THE CALIBRATION PROCEDURE-PART 1

Lines 1-26

```
#define RED   10
#define photoResistorPin A0

//calibration values stored in object because it much easier to load
and save them as objects- no need for specific memory allocation
struct CalibrationObject
{
  float calibrationDarkValue;
  float calibrationLightValue;
  float calibrationDelta;
}

calibrationObject;

int LEDpin;
int photoInterrupterPin;

float tempDarkValue;
float tempLightValue;

float darkValue;
float lightValue;
float delta;

char incomingChar = 0;
int current_color_idx;
bool colorCalibrationRequested;
```

Explanation regarding the: struct CalibrationObject:
This structure holds the calibration values for the photoresistor when the LED is off (dark) nd when it is on (light) as well as the difference between these two values, which is referred to as delta.

HW02.14-FULL CODE FOR THE CALIBRATION PROCEDURE-PART 2

Lines 29-45

```
void setup()
{
  Serial.begin(9600);
  LEDpin = 10 ; // red pin is at pin 10
  pinMode(LEDpin, OUTPUT);

  darkValue = 0;
  lightValue = 0;
  delta = 0 ;

  colorCalibrationRequested = false;

  photoInterrupterPin = 2;
  pinMode( photoInterrupterPin, INPUT);

  attachInterrupt(digitalPinToInterrupt(photoInterrupterPin), photoInterrupterInterrupt, RISING);
}
```

Explanation regarding the: void setup() function:

This function runs once when the Arduino is reset or powered on. It initializes serial communication, sets up the LED and photointerrupter pins, zeroes out the calibration values, and attaches an interrupt to the photo interrupter pin that will trigger photoInterrupterInterrupt() when a change is detected.

HW02.14-FULL CODE FOR THE CALIBRATION PROCEDURE-PART 3

Lines 47-52

```
void photoInterrupterInterrupt()
{
  noInterrupts();
  colorCalibrationRequested = true;
  interrupts();
}
```

Explanation regarding the: void photoInterrupterInterrupt() function:
This is an interrupt service routine (ISR) that is called whenever the photointerrupter pin goes from LOW to HIGH (RISING).
It sets a flag colorCalibrationRequested to true, which indicates that a color calibration needs to be performed.

HW02.14-FULL CODE FOR THE CALIBRATION PROCEDURE-PART 4

Lines 54-83

```
void GetRGBdata()
{
  digitalWrite(LEDpin, LOW );

  int stabilizationDelayValue = 10000;
  unsigned long last_micros = micros();

  for (int N = 1; N <= 10 ; N++)
  {
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      tempDarkValue += analogRead(photoResistorPin);
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      digitalWrite(LEDpin, HIGH );
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      tempLightValue += analogRead(photoResistorPin);
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      digitalWrite(LEDpin, LOW );
      if ( N == 10 )
      {
        tempDarkValue = tempDarkValue / 10;
        tempLightValue = tempLightValue / 10;
      }
  }
}
```

Explanation regarding the: void GetRGBdata() function:

This function turns off the LED, waits for a stabilization period, and then takes 10 readings of the photoresistor's value in both dark and light conditions.
It averages these readings to get a stable measurement for calibration.

HW02.14-FULL CODE FOR THE CALIBRATION PROCEDURE-PART 5

Lines 85-107

Explanation regarding the: void DoCalibration() function:
This function calls GetRGBdata() to gather sensor data and then calculates the calibration values. It stores the average dark and light values as well as their delta into the calibrationObject.

```
void DoCalibration()
{
    GetRGBdata();
      calibrationObject.calibrationDarkValue = tempDarkValue;
      calibrationObject.calibrationLightValue =  tempLightValue;

      calibrationObject.calibrationDelta =
      calibrationObject.calibrationDarkValue - calibrationObject.calibrationLightValue;
}

void printCalibrationData()
{
  Serial.println("Calibrationd Data:");
  Serial.println("*** RED ***");
  Serial.print("    Dark Value = ");
  Serial.println(calibrationObject.calibrationDarkValue);

  Serial.print("    Light Value = ");
  Serial.println(calibrationObject.calibrationLightValue);

  Serial.print("    DELTA = ");
  Serial.println(calibrationObject.calibrationDelta);
}
```

Explanation regarding the: void printCalibrationData() function:
After calibration, this function is called to print the results to the serial monitor. It displays the dark and light calibration values as well as the delta for the RED LED.

HW02.14-FULL CODE FOR THE CALIBRATION PROCEDURE-PART 6

Lines 109-117

```
void loop()
{
  if (colorCalibrationRequested == true)
  {
    colorCalibrationRequested = false;
    DoCalibration();
    printCalibrationData();
  }
}
```

Explanation regarding the: void loop() function:
This function runs continuously after setup().
It checks if a calibration has been requested (when colorCalibrationRequested is true), and if so, it performs the calibration by calling DoCalibration() and then prints the results by calling printCalibrationData(). After this, it waits for the next calibration request.

HW02.15-USER MANUAL FOR CALIBRATION-PART 1

Step 1 of the calibration: place the Arduino project on a strait place. Put a ruler in parallel of the Arduino. Make sure that in front of the LED and the photoresistor there is a white "wall". (we want to simulate the calibration as much as possible to the measurement process)

Step 2 of the calibration : Connect USB to your PC for power supply and upload the code of the calibration to the Arduino UNO

Figure 9

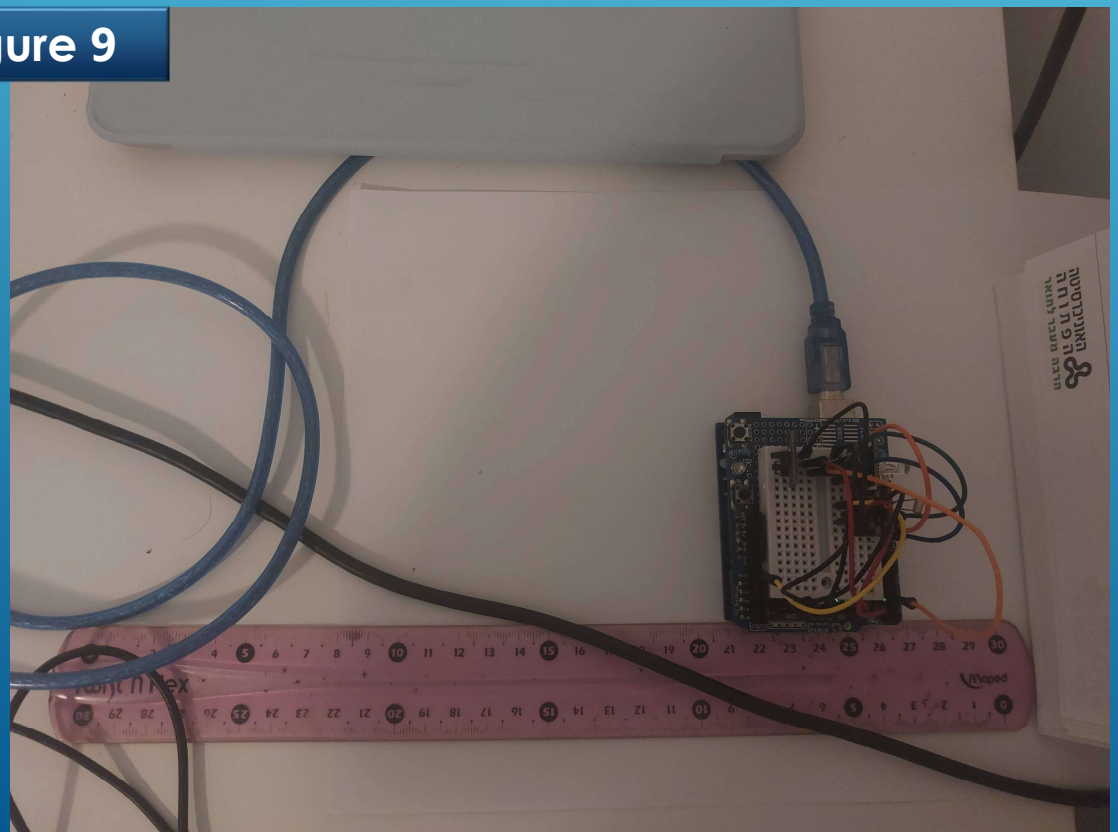
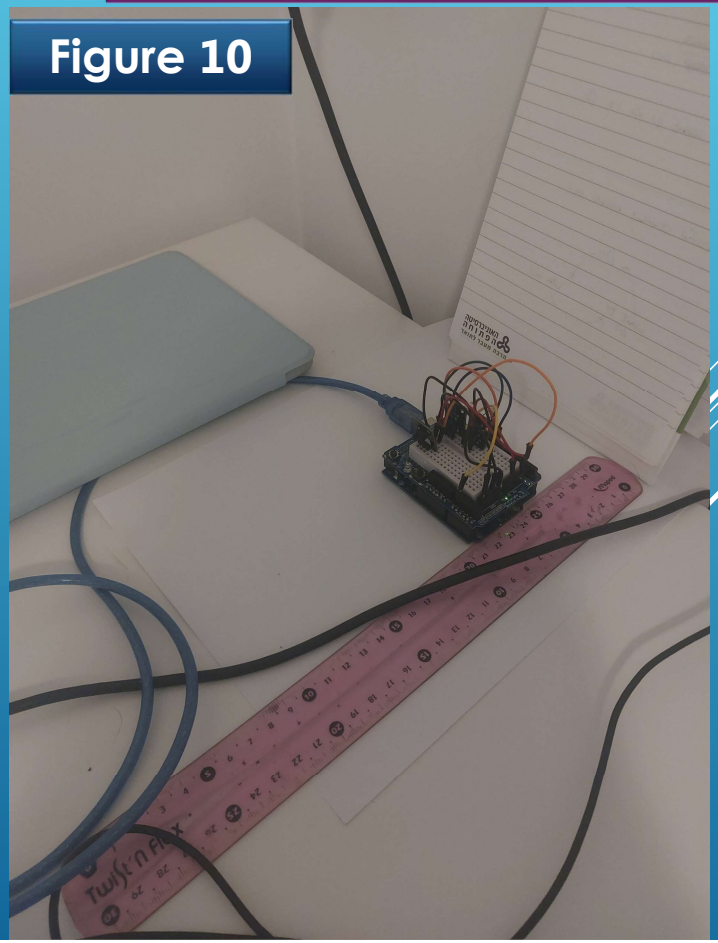


Figure 10



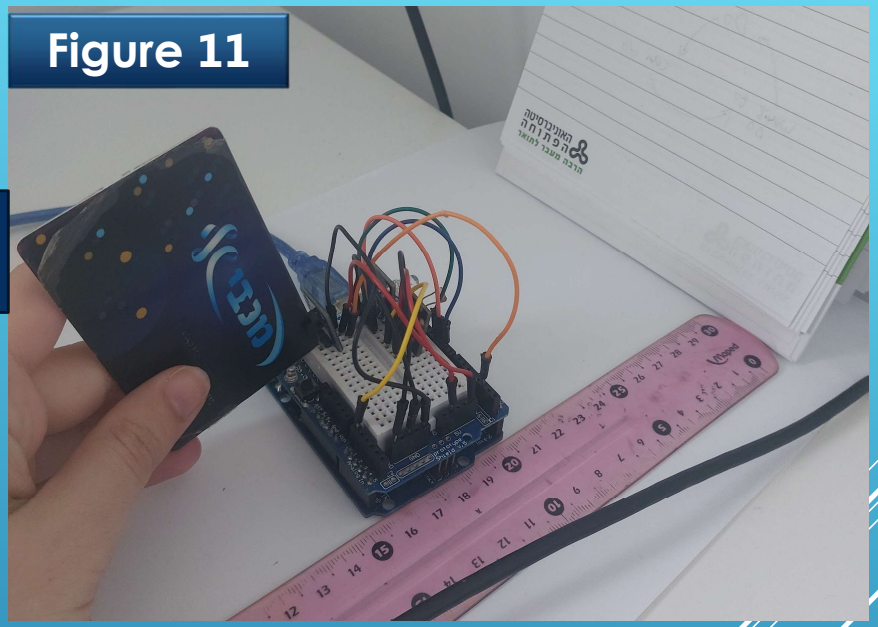
HW02.15-USER MANUAL FOR CALIBRATION-PART 2

The next step is to calibrate the system on all the distances – range of 1-10 [cm] – for every increment of 1 cm – need to perform 3 measures and take the avg of them.

All you have to do is to block the light between the "walls" of the photointerrupter module – and the results "will wait for you" on the monitor

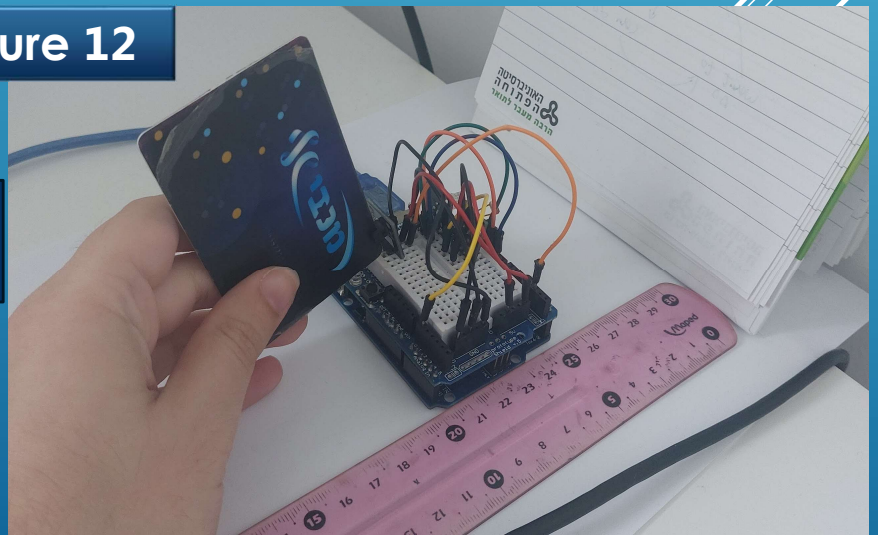
Calibrate for 10 cm

Figure 11



Calibrate for 5 cm

Figure 12



HW02.15-USER MANUAL FOR CALIBRATION-PART 3

10 cm calibration –one can see that the delta values we got are consistent (~214) and I did it 4 times – the avg will be one the 4 values of the deltas – the avg will enter the final table for finding the coefficients

This is an interactive demonstration – please click the link to get to my YouTube channel-the demonstration is stored there:

https://youtu.be/OFnK5745Qac

Pay attention - For each color we got a 'delta' value that represent the difference between the dark value to the light value.

HW02.15-USER MANUAL FOR CALIBRATION-PART 4

Calibration example results: 4 measures of 10 cm → then put the results in the table and finally take the avg of the 4 measures. In the next slide the results of the distance 1-10 cm and find the coefficients of the 3rd order polynomial → the coefficients of the polynomial need to manually typed in the measurements Arduino code

Figure 14

```
Calibrationd Data:
*** RED ***
    Dark Value = 858.35
    Light Value = 644.34
    DELTA = 214.01
Calibrationd Data:
*** RED ***
    Dark Value = 860.34
    Light Value = 645.43
    DELTA = 214.90
Calibrationd Data:
*** RED ***
    Dark Value = 859.43
    Light Value = 644.34
    DELTA = 215.09
Calibrationd Data:
*** RED ***
    Dark Value = 855.04
    Light Value = 641.43
    DELTA = 213.61
```

Take the avg of 4 measures for a given distance

Figure 13

[cm]	1st	2nd	3rd	4th	avg
1	445.4	454.14	451.81	459.18	452.6
2	389.8	387.38	386.84	388.02	388
3	344.67	342.57	343.76	341.58	343.1
4	311.46	312.65	312.76	312.48	312.3
5	285.47	282.95	284.49	282.05	283.7
6	269.2	265.52	265.05	268.31	267
7	241.07	241.81	242.08	239.81	241.2
8	228.68	230.17	232.02	231.2	230.5
9	224.02	224.4	222.14	224.71	223.8
10	214.01	214.9	215.09	213.61	214.4

HW02.15-USER MANUAL FOR CALIBRATION-PART 5

[cm]	1st	2nd	3rd	4th	avg
1	445.4	454.14	451.81	459.18	452.6
2	389.8	387.38	386.84	388.02	388
3	344.67	342.57	343.76	341.58	343.1
4	311.46	312.65	312.76	312.48	312.3
5	285.47	282.95	284.49	282.05	283.7
6	269.2	265.52	265.05	268.31	267
7	241.07	241.81	242.08	239.81	241.2
8	228.68	230.17	232.02	231.2	230.5
9	224.02	224.4	222.14	224.71	223.8
10	214.01	214.9	215.09	213.61	214.4

Figure 15

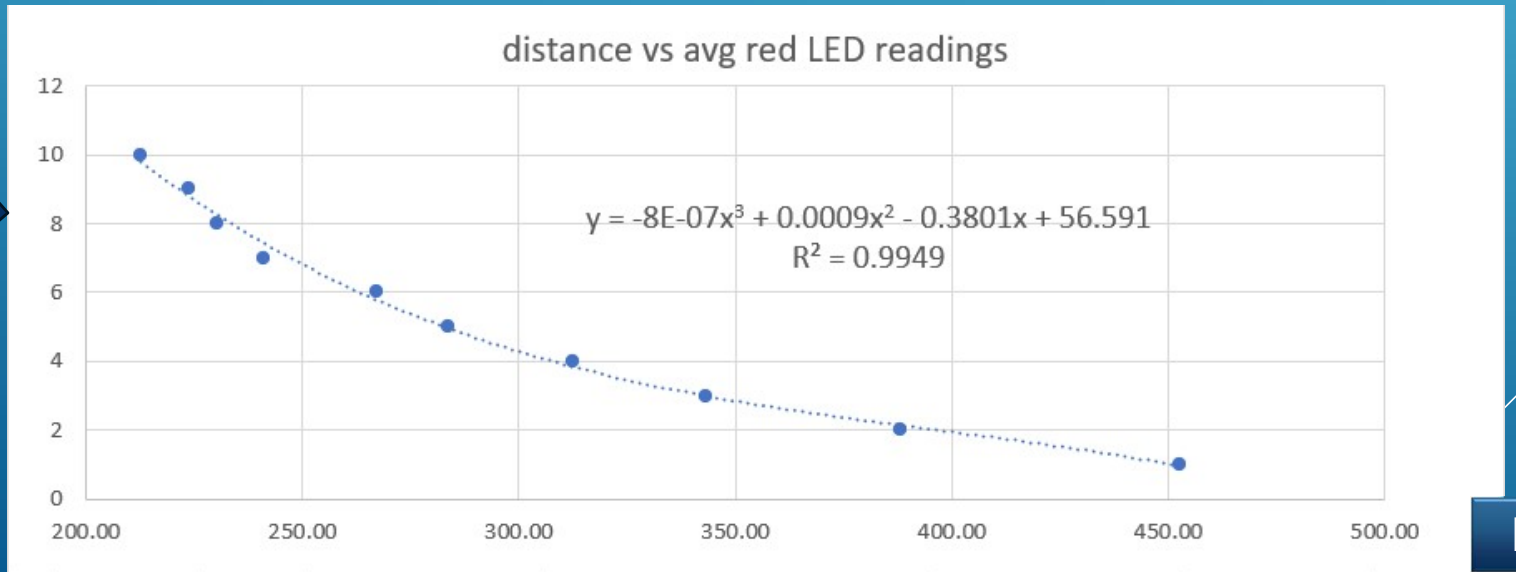
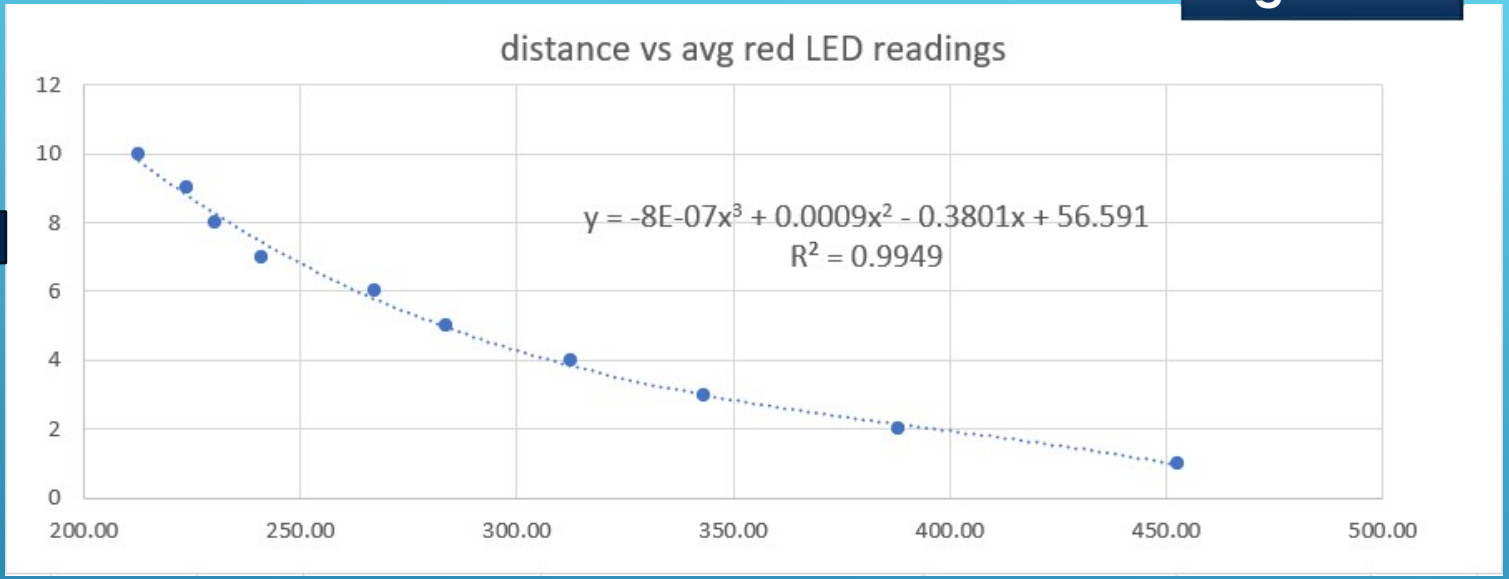


Figure 16

After filling the table and finding the avg of each distance – the graph will automatically (after our setting) show the polynomial approximation to the data – we found that 3rd order approximation is the best fit for measuring the distance with minimum additional fixings

HW02.15-USER MANUAL FOR CALIBRATION-PART 6

Figure 17




```
29   float Poly_Coefficients[4] = { -8E-07,0.0009,- 0.3801,56.591};
```

The inserting of the coefficients of the polynomial need to manually be done in the measurements Arduino code-line 29 inside the poly_coefficients array

After completing that step-user is ready to measure the distance using the measurement program

HW02.15-USER MANUAL FOR CALIBRATION-PART 7

Test example :

Checking the system at ~10[cm] distance from wall. (3 measures to make sure that the system works properly)

```
Optical Distance Sensor - Measured distance is:
10[cm]
NORMALIZED = 217.42
Optical Distance Sensor - Measured distance is:
10[cm]
NORMALIZED = 215.94
Optical Distance Sensor - Measured distance is:
10[cm]
```

Figure 19

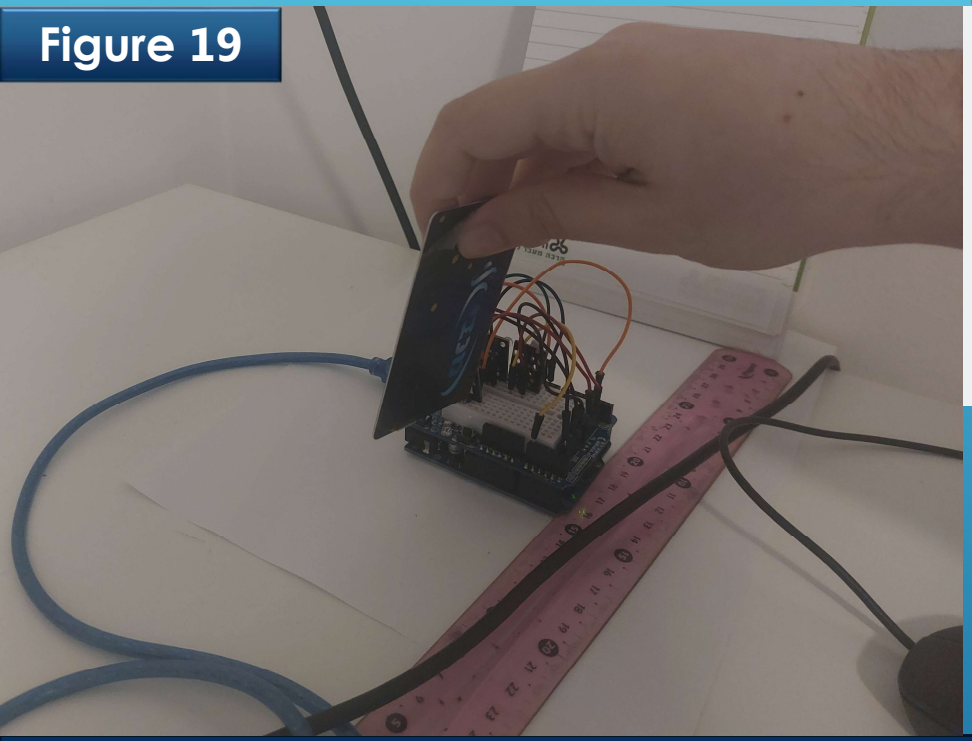
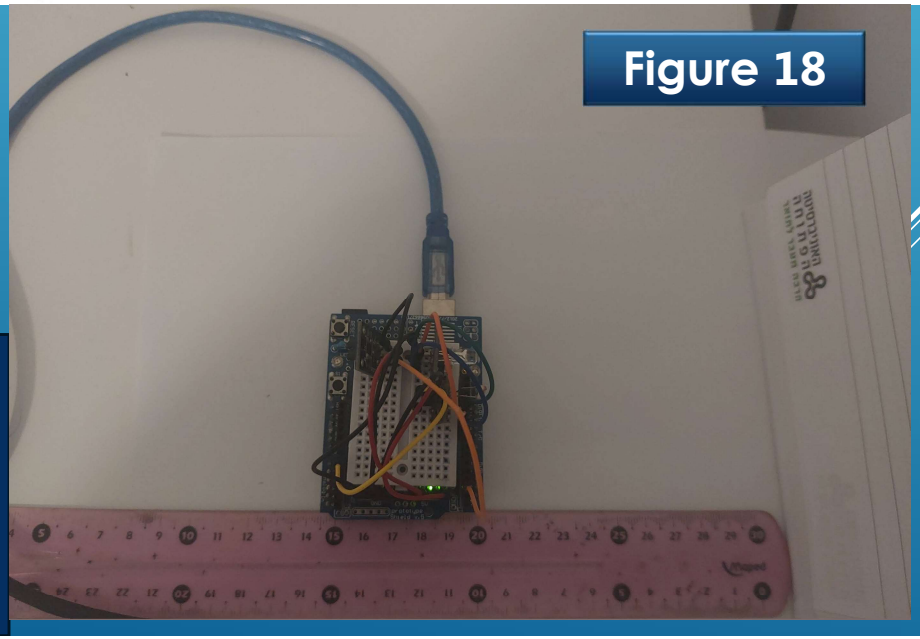


Figure 18



After inserting the 3rd order polynomial coefficients into the measurement code – all you have to do is to upload the code and block the light in the light interrupter module (using a card or paper) and the distance will immediately appear on the monitor

HW02.16+ - REAL DEMONSTRATIONS OF MEASUREMENT ON VARIUS DISTANCES – IN YOUTUBE LINKS-PART 1

3 cm test –one can see that the measurement is accurate (stands in +- 1 cm accuracy)

This is an interactive demonstration – please click the link to get to my YouTube channel-the demonstration is stored there:

https://youtu.be/a1PVtDBmHIE

HW02.16-Code for the measurements-PART 1

Lines 1-26

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define RED   10
#define GREEN 11

#define photoResistorPin A0
#define Max_distance 10

int photoInterrupterPin;
int distance = 0;
int new_distance=0;
int LEDpin;
int green_pin;
const uint16_t t1_comp = 31250 / 2; // 0.5s cycle
const uint16_t t1_load = 0;

float tempDarkValue;
float tempLightValue;

float darkValue;
float lightValue;
float delta;

int current_color_idx;
bool colorDistanceRequested = false;
```

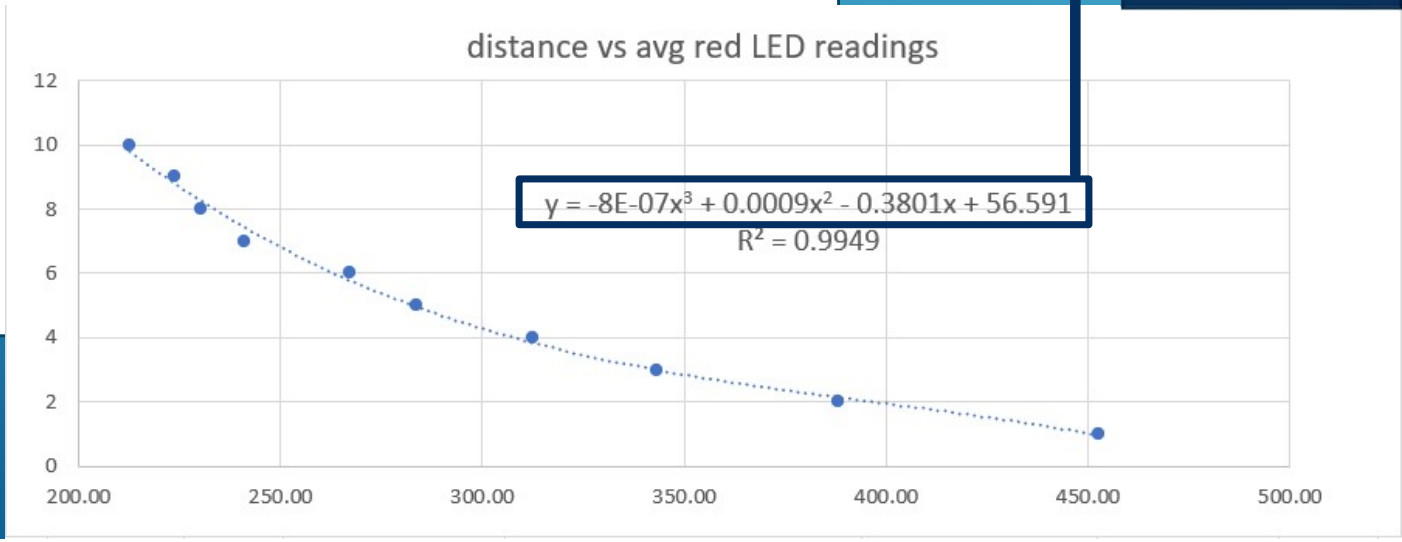
HW02.16-Code for the measurements-PART 2

Lines 28-45

User need to manually type the coefficient of the 3rd order polynomial to the poly_coefficients array in the measurement code

```
float Poly_Coefficients[4] = { -8E-07,0.0009,- 0.3801,56.591};

float DeltaNormal;

unsigned long previousMillis = 0; // will store last time LED was updated
const long interval = 500; // interval at which to blink (milliseconds)
int blinkCount = 0; // Number of blinks
int totalBlinks = 0; // Total blinks for distance
bool isBlinking = false; // Is currently blinking

struct CalibrationObject
{
  float calibrationDarkValue;
  float calibrationLightValue;
  float calibrationDelta;
}

calibrationObject;
```

Figure 20



HW02.16-Code for the measurements-PART 3

Lines 47-74

```
void setup()
{
  Serial.begin(9600);
  // initialize timer1
  cli(); // stop interrupts
  TCCR1A = 0; // set entire TCCR1A register to 0
  TCCR1B |= (1 << CS12); //Set the prescale of 256
  TCCR1B &= ~(1 << CS11);
  TCCR1B &= ~(1 << CS10);
  TCNT1 = t1_load; // Reset Timer1
  OCR1A = t1_comp; //Value for ORC1A for 1s to compare
  TIMSK1 |= (1 << OCIE1A); //Set the interrupt request
  sei(); //Enable interrupt

  LEDpin = 10; // red pin is at pin 10
  green_pin=11;
  pinMode(LEDpin, OUTPUT);
  pinMode(green_pin, OUTPUT); // Initialize the green LED pin as an output
  darkValue = 0;
  lightValue = 0;
  delta = 0 ;

  photoInterrupterPin = 2;
  pinMode( photoInterrupterPin, INPUT);

  attachInterrupt(digitalPinToInterrupt(photoInterrupterPin), photoInterrupterInterrupt, RISING);
}
```

HW02.16-Code for the measurements-PART 4

Lines 76-86

```
ISR(TIMER1_COMPA_vect)          // timer compare interrupt service routine
{
  TCNT1 = t1_load;
}

void photoInterrupterInterrupt()
{
  noInterrupts();
  colorDistanceRequested = true;
  interrupts();
}
```

Explanation regarding the: void photoInterrupterInterrupt() function:

ISR triggered by the photo interrupter:

- Temporarily disables interrupts to prevent re-entrant calls.
- Sets colorDistanceRequested to true, indicating that new distance calculation is needed.
- Re-enables interrupts.

HW02.16-Code for the measurements-PART 5

Lines 88-117

```
void GetRGBdata()
{
  digitalWrite(LEDpin, LOW );
  int stabilizationDelayValue = 10000;
  unsigned long last_micros = micros();
  for (int N = 1; N <= 10 ; N++)
  {
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      tempDarkValue += analogRead(photoResistorPin);
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      digitalWrite(LEDpin, HIGH );
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      tempLightValue += analogRead(photoResistorPin);
      while (micros() - last_micros < stabilizationDelayValue) {}
      last_micros = micros();
      digitalWrite(LEDpin, LOW );
      if ( N == 10 )
      {
        tempDarkValue = tempDarkValue / 10;
        tempLightValue = tempLightValue / 10;
      }
  }
}
```

HW02.16-Code for the measurements-PART 6

Lines 119-142

```
void DoCalibration()
{
  GetRGBdata();

    calibrationObject.calibrationDarkValue = tempDarkValue;
    calibrationObject.calibrationLightValue =  tempLightValue;

    calibrationObject.calibrationDelta =
      calibrationObject.calibrationDarkValue - calibrationObject.calibrationLightValue;
    DeltaNormal = calibrationObject.calibrationDelta;
}

void calcDistance()
{
  float noramlized = 0;
  noramlized += DeltaNormal * DeltaNormal;
  noramlized=sqrt(noramlized);
  distance = (Poly_Coefficients[0] * (noramlized * noramlized * noramlized )) + (Poly_Coefficients[1] * (
noramlized * noramlized)) + Poly_Coefficients[2] * noramlized  + Poly_Coefficients[3] ;
  Serial.print("NORMALIZED = ");
  Serial.println(noramlized);
  Serial.println("Optical Distance Sensor - Measured distance is: ");
  Serial.print(distance+2);
  Serial.println("[cm]");
}
```

Explanation regarding the: void calcDistance() function:

Calculates the distance using a polynomial equation:
Normalizes the delta value by squaring it and taking the square root.
Calculates the distance using the polynomial equation and coefficients.
Outputs the calculated distance to the serial monitor.

HW02.16-Code for the measurements-PART 7

Lines 144-162

```
void loop()
{
  if (colorDistanceRequested == true)
  {
    colorDistanceRequested = false;

    DoCalibration();
    calcDistance();

    // Set total blinks based on the distance
    totalBlinks = distance + 2;
    blinkCount = 0; // Reset blink count
    isBlinking = true; // Start blinking
  }

  if (isBlinking) {
    blinkWithoutDelay();
  }
}
```

Explanation regarding the: void loop() function:

Main program loop that runs continuously:

- **Checks if a distance calculation has been requested and performs it.**
- **Manages the LED blinking based on the calculated distance.**

HW02.16-Code for the measurements-PART 8

Lines 164-185

```
void blinkWithoutDelay() {
  unsigned long currentMillis = millis();

  if (blinkCount < totalBlinks) {
    if(currentMillis - previousMillis >= interval){
      // save the last time you blinked the LED
      previousMillis = currentMillis;

      // if the LED is off turn it on and vice-versa:
      if (digitalRead(GREEN) == LOW) {
        digitalWrite(GREEN, HIGH); // Turn the green LED on
      } else {
        digitalWrite(GREEN, LOW); // Turn the green LED off
        blinkCount++; // Increment blink count after turning off
      }
    }
  } else {
    // Stop blinking after reaching the total blinks
    isBlinking = false;
    digitalWrite(GREEN, LOW); // Ensure LED is off after blinking
  }
}
```

Explanation regarding the: void blinkWithoutDelay() function:

- **Controls the blinking of the green LED without using delay():**
- **Checks the time elapsed since the last blink.**
- **Toggles the LED state and increments the blink counter accordingly.**
- **Stops blinking after reaching the required number of blinks.**

limits of the system – recommendations for using and accuracy measurements-part 1

According to our measurements – the distance meter works well (accuracy of +- 1 cm) in range $R = [1\ cm, 12\ cm]$. For $R \geq 13cm$ the measurement becomes completely wrong and distorted

Real measurements are presented below – one can see that the normalized is totally distorted at 13 cm

```
Optical Distance Sensor - Measured distance is:
10[cm]
NORMALIZED = 207.67
Optical Distance Sensor - Measured distance is:
11[cm]
NORMALIZED = 202.77
Optical Distance Sensor - Measured distance is:
11[cm]
NORMALIZED = 124.88
Optical Distance Sensor - Measured distance is:
23[cm]
```

Real distance of 10 cm

Real distance of 11 cm

Real distance of 12 cm

Real distance of 13 cm

Therefore ,we advise not to measure in range of $R \geq 13cm$

limits of the system – recommendations for using and accuracy measurements-part 2

According to our measurements – the distance meter works well (accuracy of +- 1 cm) in range $R = [1\ cm, 12\ cm]$. For $R \geq 13cm$ the measurement becomes completely wrong and distorted

The reason for that phenomena is Due to the fact that light intensity varies inversely with the square of the distance, the photoresistor is unable to accurately determine the value of $R \geq 13cm$.

$$intensity \propto \frac{1}{distance^2}$$

limits of the system – recommendations for using and accuracy measurements-part 3

Accuracy measurements – the following table presents the accuracy of a measurement that was taken on 24/3/2024 on 18:23 (measurements in slides 33-34 in that presentation were taken on 23/3/2024)

real distance in [cm]	1st normlized	1st reported distance	2nd normlized	2nd reported distance	3rd normlized	3rd reported distance	4th normlized	4th reported distance	avg normlized	avg reported distance	Error	Error in %	Accuracy in %
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													

All the measured data will be presented in the next slides!

We are going to fill that table and then find the accuracy and generate few graphs that will help us understand the behavior of the system much better

limits of the system – recommendations for using and accuracy measurements-part 4

For real distance of 10 cm:

```
NORMALIZED = 216.85
Optical Distance Sensor - Measured distance is:
10[cm]
NORMALIZED = 216.08
Optical Distance Sensor - Measured distance is:
10[cm]
NORMALIZED = 217.81
Optical Distance Sensor - Measured distance is:
10[cm]
NORMALIZED = 216.88
Optical Distance Sensor - Measured distance is:
10[cm]
```

For real distance of 9 cm:

```
NORMALIZED = 229.29
Optical Distance Sensor - Measured distance is:
9[cm]
NORMALIZED = 229.63
Optical Distance Sensor - Measured distance is:
9[cm]
NORMALIZED = 227.16
Optical Distance Sensor - Measured distance is:
9[cm]
NORMALIZED = 230.02
Optical Distance Sensor - Measured distance is:
9[cm]
```

For real distance of 8 cm:

```
NORMALIZED = 238.80
Optical Distance Sensor - Measured distance is:
8[cm]
NORMALIZED = 240.68
Optical Distance Sensor - Measured distance is:
8[cm]
NORMALIZED = 240.27
Optical Distance Sensor - Measured distance is:
8[cm]
NORMALIZED = 240.33
Optical Distance Sensor - Measured distance is:
8[cm]
```

For real distance of 7 cm:

```
NORMALIZED = 248.73
Optical Distance Sensor - Measured distance is:
7[cm]
NORMALIZED = 251.97
Optical Distance Sensor - Measured distance is:
7[cm]
NORMALIZED = 250.00
Optical Distance Sensor - Measured distance is:
7[cm]
NORMALIZED = 251.20
Optical Distance Sensor - Measured distance is:
7[cm]
```

Next slide more collected data

limits of the system – recommendations for using and accuracy measurements-part 5

For real distance of 6 cm:

```
NORMALIZED = 263.22
Optical Distance Sensor - Measured distance is:
6[cm]
NORMALIZED = 262.22
Optical Distance Sensor - Measured distance is:
6[cm]
NORMALIZED = 261.12
Optical Distance Sensor - Measured distance is:
6[cm]
NORMALIZED = 261.41
Optical Distance Sensor - Measured distance is:
6[cm]
```

For real distance of 4 cm:

```
NORMALIZED = 307.13
Optical Distance Sensor - Measured distance is:
3[cm]
NORMALIZED = 305.51
Optical Distance Sensor - Measured distance is:
3[cm]
NORMALIZED = 303.65
Optical Distance Sensor - Measured distance is:
3[cm]
NORMALIZED = 305.07
Optical Distance Sensor - Measured distance is:
3[cm]
```

For real distance of 5 cm:

```
NORMALIZED = 279.66
Optical Distance Sensor - Measured distance is:
5[cm]
NORMALIZED = 278.27
Optical Distance Sensor - Measured distance is:
5[cm]
NORMALIZED = 278.43
Optical Distance Sensor - Measured distance is:
5[cm]
NORMALIZED = 282.34
Optical Distance Sensor - Measured distance is:
5[cm]
```

For real distance of 3 cm:

```
NORMALIZED = 348.48
Optical Distance Sensor - Measured distance is:
2[cm]
NORMALIZED = 350.85
Optical Distance Sensor - Measured distance is:
2[cm]
NORMALIZED = 351.18
Optical Distance Sensor - Measured distance is:
2[cm]
NORMALIZED = 353.92
Optical Distance Sensor - Measured distance is:
2[cm]
```

Next slide more collected data

limits of the system – recommendations for using and accuracy measurements-part 6

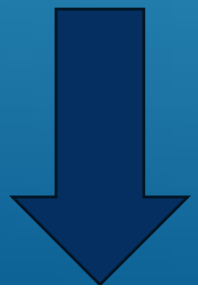
For real distance of 2 cm:

```
NORMALIZED = 384.19
Optical Distance Sensor - Measured distance is:
1[cm]
NORMALIZED = 381.82
Optical Distance Sensor - Measured distance is:
1[cm]
NORMALIZED = 382.88
Optical Distance Sensor - Measured distance is:
1[cm]
NORMALIZED = 383.09
Optical Distance Sensor - Measured distance is:
1[cm]
```

For real distance of 1 cm:

```
NORMALIZED = 400.97
Optical Distance Sensor - Measured distance is:
0[cm]
NORMALIZED = 399.50
Optical Distance Sensor - Measured distance is:
0[cm]
NORMALIZED = 398.45
Optical Distance Sensor - Measured distance is:
0[cm]
NORMALIZED = 399.74
Optical Distance Sensor - Measured distance is:
0[cm]
```

In the next slide the summary of the data and produced graphs



limits of the system – recommendations for using and accuracy measurements-part 7

real distance in [cm]	1st normlized	1st reported distance	2nd normlized	2nd reported distance	3rd normlized	3rd reported distance	4th normlized	4th reported distance	avg normlized	avg reported distance	Error	Error in %	Accuracy in %
1	400.97	0	399.5	0	398.45	0	399.74	0	399.665	0	1	100	0
2	384.19	1	381.82	1	382.88	1	383.09	1	382.995	1	1	50	50
3	348.48	2	350.85	2	351.18	2	353.92	2	351.1075	2	1	33.33333333	66.66666667
4	307.13	3	305.51	3	303.65	3	305.07	3	305.34	3	1	25	75
5	279.66	5	278.27	5	278.43	5	282.34	5	279.675	5	0	0	100
6	263.22	6	262.22	6	261.12	6	261.41	6	261.9925	6	0	0	100
7	248.73	7	251.97	7	250	7	251.2	7	250.475	7	0	0	100
8	238.8	8	240.68	8	240.27	8	240.33	8	240.02	8	0	0	100
9	229.29	9	229.63	9	227.16	9	230.02	9	229.025	9	0	0	100
10	216.85	10	216.08	10	217.81	10	216.88	10	216.905	10	0	0	100

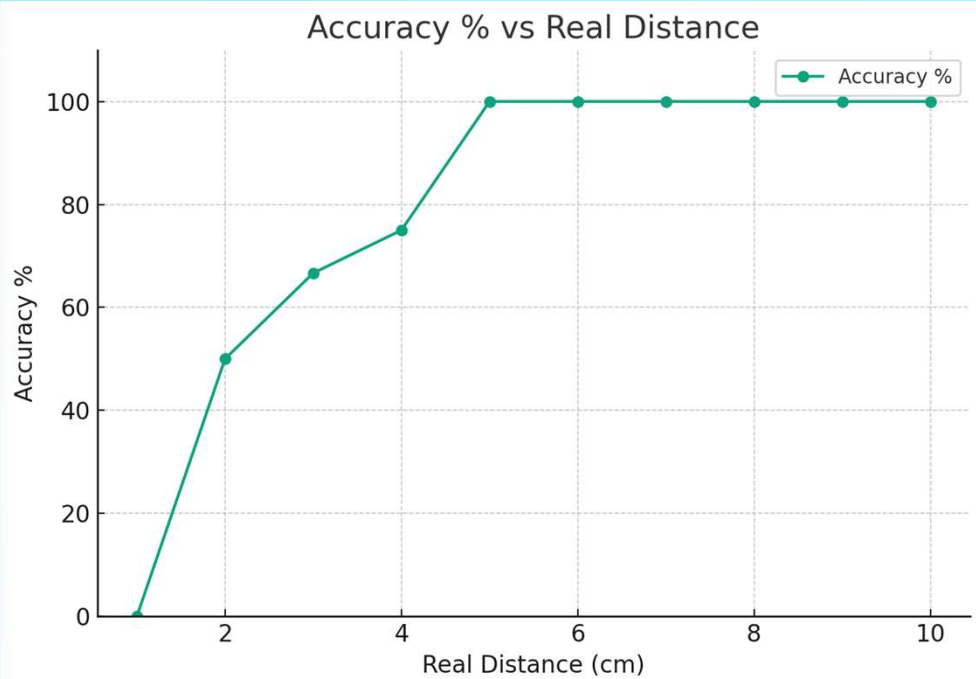
$$Error = real\ distance\ in\ [cm] - avg\ reported\ distance$$

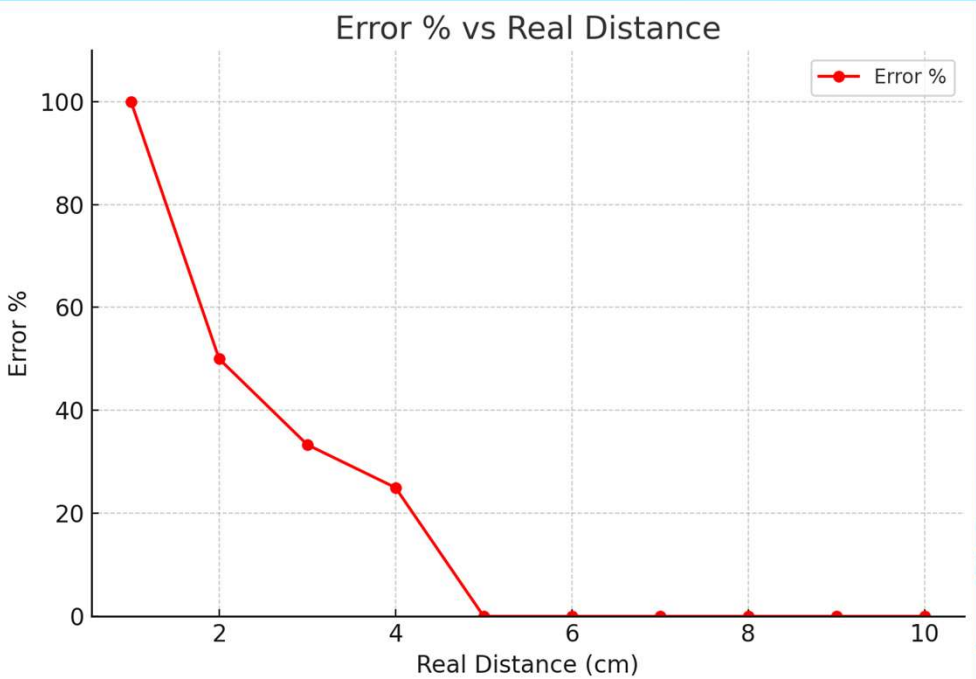
$$Error\ in\ \% = 100 * (\text{"}Error\text{"}/real\ distance\ in\ [cm])$$

$$accuracy = 100 - Error\ in\ \%$$

measurement methodology

limits of the system – recommendations for using and accuracy measurements-part 8 – data presentation - accuracy of the project

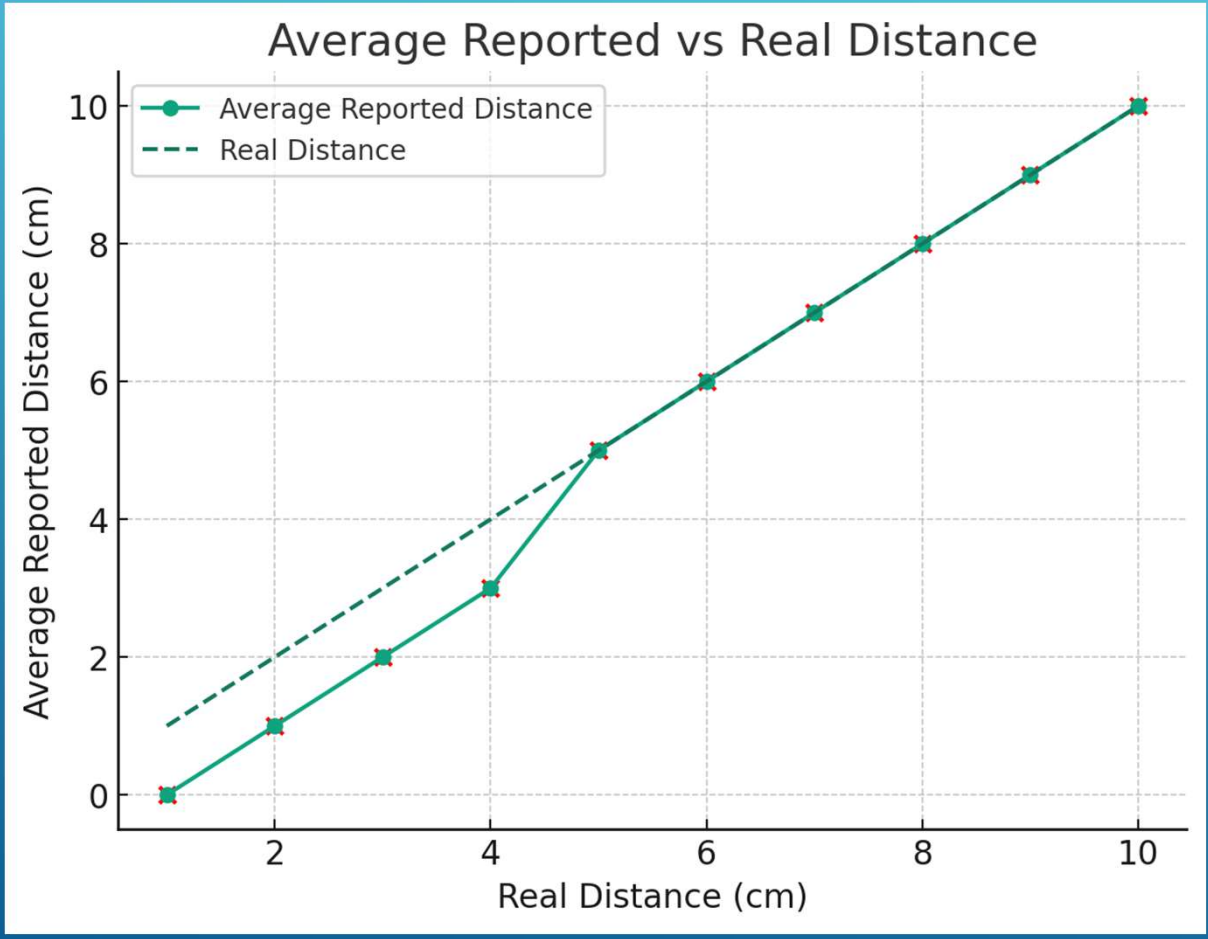




The accuracy graph shows an increasing trend in accuracy as the real distance increases, reaching 100% and leveling off from 5 cm onwards. It means that the measurement method becomes more accurate at higher distance values.

The error percentage graph shows a steep decline as the distance increases, which complements the accuracy graph. The highest error percentage is at the lowest distance, which quickly reduces to 0% by 5 cm and remains at this low level for greater distances. This indicates that the error in measurements is significantly reduced as the distance increases, leading to high accuracy for distances beyond 5 cm.

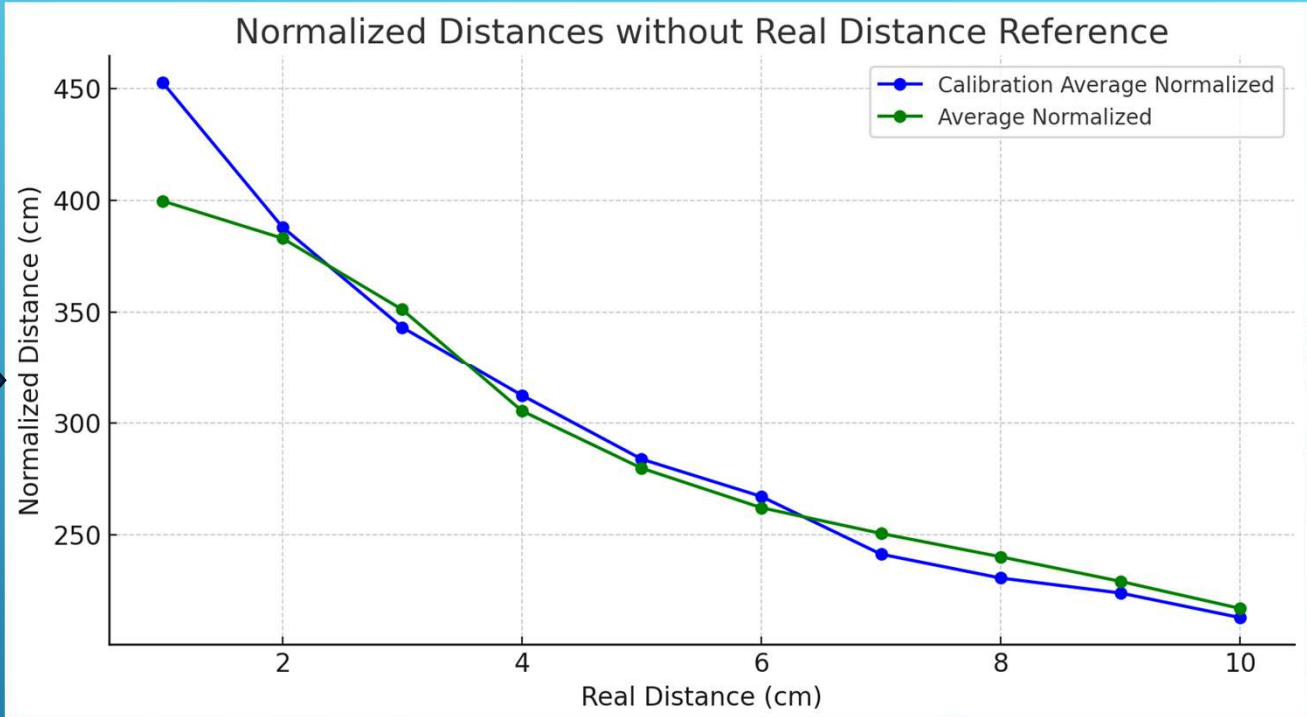
limits of the system – recommendations for using and accuracy measurements-part 9- data presentation – distance in [cm] reported vs real measures



The line representing the average reported distance closely follows the dashed line that represents the real distance.
The data points (marked in red) are very close to the line of equality, indicating that the average reported distances are quite close to the real distances.
This graph confirms the high accuracy of the reported measurements compared to the actual values. (+- 1 cm accuracy achieved)

limits of the system – recommendations for using and accuracy measurements-part 10-data presentation – normalized data in measurement and in calibration

real distance in [cm]	calibration avg normlized	avg normlized
1	452.6325	399.665
2	388.01	382.995
3	343.145	351.1075
4	312.3375	305.34
5	283.74	279.675
6	267.02	261.9925
7	241.1925	250.475
8	230.5175	240.02
9	223.8175	229.025
10	212.8075	216.905



Final conclusion – we succeed to perform good and accurate measurement and our calibration of the system brings accurate distance measurements results!

The graph compares the calibration average normalized distances and the average normalized distances:

- The blue line represents the calibration average normalized distances.
- The green line represents the average normalized distances.

The results are unified with the previews data we archived!

HW02.16+- REAL DEMONSTRATIONS OF MEASUREMENT ON VARIOUS DISTANCES – IN YOUTUBE LINKS-PART 2

4 cm test –one can see that the measurement is accurate (stands in +- 1 cm accuracy)

This is an interactive demonstration – please click the link to get to my YouTube channel-the demonstration is stored there:

https://youtu.be/l9LLH9PaXp4

HW02.16+- REAL DEMONSTRATIONS OF MEASUREMENT ON VARIOUS DISTANCES – IN YOUTUBE LINKS-PART 3

5 cm test –one can see that the measurement is accurate (stands in +- 1 cm accuracy)

This is an interactive demonstration – please click the link to get to my YouTube channel-the demonstration is stored there:

https://youtu.be/sOmdC_ljs98

HW02.16+- REAL DEMONSTRATIONS OF MEASUREMENT ON VARIOUS DISTANCES – IN YOUTUBE LINKS-PART 4

8 cm test –one can see that the measurement is accurate (stands in +- 1 cm accuracy)

This is an interactive demonstration – please click the link to get to my YouTube channel-the demonstration is stored there:

https://youtu.be/JL4XoK0PwQo

HW02.16+- REAL DEMONSTRATIONS OF MEASUREMENT ON VARIOUS DISTANCES – IN YOUTUBE LINKS-PART 5

9 cm test –one can see that the measurement is accurate (stands in +- 1 cm accuracy)

This is an interactive demonstration – please click the link to get to my YouTube channel-the demonstration is stored there:

https://youtu.be/thezoXSZJQ0

HW02.16+- REAL DEMONSTRATIONS OF MEASUREMENT ON VARIOUS DISTANCES – IN YOUTUBE LINKS-PART 6

10 cm test –one can see that the measurement is accurate (stands in +- 1 cm accuracy)

This is an interactive demonstration – please click the link to get to my YouTube channel-the demonstration is stored there:

https://youtu.be/thezoXSZJQ0