University of Moratuwa

EN 1093 – Laboratory Practice I

RGB Colour Sensor

Loshanan M.	190363X
Luxshan S.	190364C
Mathotaarachchi M.M.	190388D
Mokeeshan V.	190395V

Supervisor- Achintha Wijesinghe

This is submitted as a partial fulfillment for the module
EN1093: Laboratory Practice I
Department of Electronic and Telecommunication Engineering
University of Moratuwa

16th August 2021

Abstract

Coloris- the RGB colour sensor is device that can detect the colour of any given surface. And there is a mode to light the colour using RGB LED with user given R, G, B values. The key idea is coloured objects reflect their own colour and absorb all others. This project is done by lighting a colour light source and analysing the reflected colour light from the given surface. This colour sensor is designed using ATMega microcontroller. For an input of unknown coloured surface, the result would be in the format of RGB colour model. And the colour of that surface is detected as the most dominating colour among red, green, and blue. Also, the result of RGB colour mixture can be visualised from this product using a single RGB LED and with 16*2 LCD Display.

Table of Contents

List of Tables

List of Figures

1. Introduction

Colour detection is one of the main principles which are used to evaluate the properties of a material. Colour detection sensors are used in many industries such as food and beverage, automotive and manufacturing. In this project *Coloris*-the RGB colour sensor is designed using three LEDs and an LDR. It detects the colour of any surface and can light up the colour that a user wants.

The fundamental, used in this project is, objects appear coloured because they absorb some colours and reflect some other colours into human eyes. In this project colour sensor is developed by measuring the density of reflected colour light by the given-coloured object.

The most common RGB colour model is used throughout this project.

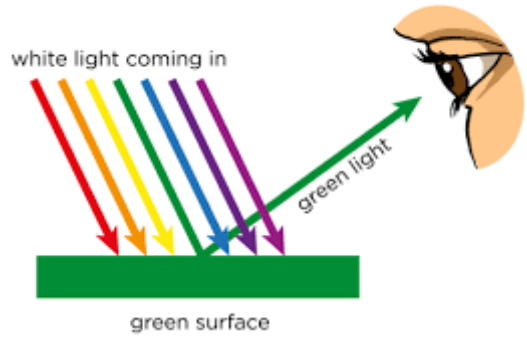


Figure 1.1:theory of colour detection

1.1 Task

- The colour sensor should sense and output the colour of a given surface.
- A calibration stage is allowed before sensing.
- There should be a mode to light a RGB LED for user given red, green, and blue values.

1.2 Specifications

- The sensor should contain
    - a/ many LDR
    - a keypad
    - an LCD display
    - PCBs
    - enclosure
    - RGB LED
- Product should be able to handle alternations in lighting conditions
- ATMega microcontrollers are recommended to use

1.3 Scientific Literature

After the problem was identified clearly a flowchart was developed to show objectives of the project. After that a circuit is designed using Arduino and it was coded properly. Then the circuit was developed by using physical simulations to have better results.

Then the project moved from Arduino to single ATMega microcontroller circuit. The basics of coding a microcontroller, were learned before moving the simulation to ATMega. Both ATMega and Arduino simulations were done by Proteus 8 software. Required PCB is designed using Altium software. Then using the dimensions of the PCB, the 3D enclosure was designed by Solidworks.

2. Methods

2.1 Arduino

At the beginning this project is designed using Arduino Uno. Circuits were designed in this stage and the simulations were done using Proteus software.

First LCD display is connected to Arduino and coded for displaying texts in that LCD display. Then the keypad is connected to Arduino for the purpose of changing the modes and entering the inputs for user input mode. Keypad library is used to code Arduino for using this matrix style keypad.

Next the main core part is developed using red, green, and blue LEDs and a LDR for the purpose of doing calibration and sensing. Since the pin connected to LDR should be an input pin, it was connected to an Analog pin of the Arduino.

Then there are no more free pins in Arduino to connect an RGB LED for user input mode. So, an I2C module is used to connect LCD display for minimising the required number of pins. Only two pins are enough for the working of the LCD display if the I2C is used, otherwise it requires 6 pins to connect. Then the RGB LED was attached with three PWM pins in Arduino, and the Arduino code was fully finalized.

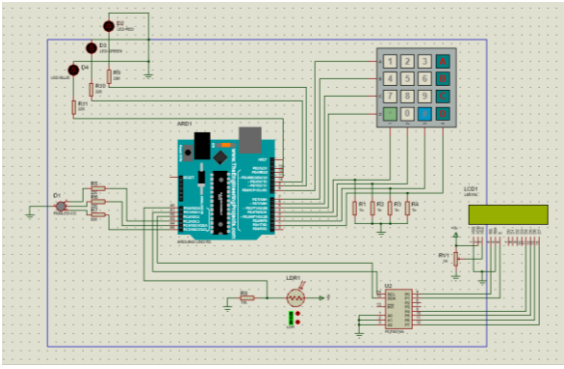


Figure 2.1:LCD Display Views

2.2 Coding ATMEGA microcontroller

As ATMEGA328P is simpler to use, with the usage of 8bit and 16bit instead of 32/64bit which are more complicated. So, here the ATMega microcontroller was used as the final design and programmed it using C.

basic libraries in C that used here.

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
```

2.2.1 Creating a function for the Keypad:

- The concept behind this task is when a key is pressed the relevant circuit is completed. First the pin connected to 1st row is made high. Then it'll check if any of the pins connected to the columns are at the high level. After a delay of 40 ms it'll do the same for the 2nd row. Just like this process takes place in all 4 rows continuously. If it recognizes that a column is high, then we get the corresponding key as input.

2.2.2 Creating functions for the LCD display and the I2C

- Used datasheets and made functions for LCD and I2C separately.

2.2.3 Creating functions for different tasks of the algorithm

- As this code must do various number of tasks, here several numbers of functions are created earlier part of the code and used in main part. The advantages of using functions are
    - Code became more simple and easily understandable.
    - reduce the length of the main part of the code.
    - Detecting the errors was easy.

There are 2 important parts.

- PWM Coding
  - PWM pins are used for control RGB LED. We needed ATMEGA328P datasheet for this part and used only 8-bit Timer/Counter0 with PWM. For this part we created a separate function.

```
void rgbLED(char x, int num); //Function to control RGB LED
```

- Analog-to-Digital Conversion
  - This is used for getting values from LDR. We needed ATMEGA328P datasheet for this part also. We used ADC0 for connect LDR.

```
ADMUX = (1 << REFS0);
ADCSRA = (1 << ADEN) | (1 << ADIE) | (1 << ADPS0) | (1 << ADPS1) | (1 << ADPS2);
DIDR0 = (ADC0D);
int revalue(); //Function to read value of LDR
```

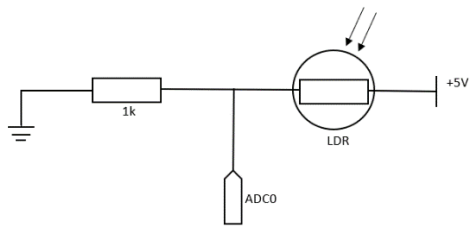


Figure 2.2:LDR reading principle

V_{in} = Input Voltage to ADC0
V_{ref}= Reference voltage(5 v)
ADC = $(1024*V_{in})/V_{ref}$
V_{in} = $(ADC*V_{ref})/1024$
LDR = $(1/Vin)*(V_{ref}-V_{in})$
LDR = (1024/ADC)-1

2.3 PCB Design

Main PCB does handle the ATmega328p microcontroller, Power supply circuit, and Sensor core part.

2.3.1 Electronic components

- ATmega328p microcontroller and Holder
- 1 k * 1 Resistor
- 10 k * 1 Resistor
- 330 ohms * 9 Resistors
- 22pF * 2 Capacitors
- 100nF * 3 Capacitors
- 330nF * 1 Capacitor
- Crystal Oscillator 16 MHz * 1
- 7805 Voltage Regulator
- Push Button * 1
- LDR * 1
- LED * 4
- RGB LED * 1

2.3.2 Connectors

- Header 2x_Male_Pin_2.54mm * 1
- Header 5x_Male_Pin_2.54mm * 1
- Header 8x_Male_Pin_2.54mm * 1
- Header 4x_Male_Pin_2.54mm * 1

2.3.3 PCB information

- Size – 110mm X 65 mm
- Layers – 2 (Top & Bottom)
- Holes – 4
- Components – 32
- Pads – 106
- Nets - 35
- Routing size – 2.540mm (5V routing line is only 5mm)
- Copper pouring is used.

2.3.4 Main Circuit Board

This mainly includes the ATmega328p microcontroller, Power supply (5V Power regulator), and Sensor core part. ATmega328p microcontroller takes signals from LDR and executes appropriate color in the RGB led.

In this PCB power supply unit, first, we connect a 9V battery to the 2x_Male header pin. Then the regulating circuit converts this 9V to 5V in this purpose we use a 7805-voltage regulator, 330nF, 100nF capacitors, LED, and 330-ohm resistor. In the ATmega328p microcontroller part, there is a 22pF two capacitors, one 16 MHz crystal oscillator, one 100nF capacitor 10k resistor, and a reset button for the working purpose of this microcontroller. This microcontroller should be programmed by the FTDI connector with the Atmel studio programming Code for the working conditions. 8x_Male_Pin header is used to connect the keypad and the 4x_Male_Pin header is used to connect the LCD display I2C module with the main PCB board.

In the Sensor core part, here used red, green, blue led lights separately, a LDR, four 330 ohms resistors, and one 1K resistor. It located as LEDs in the circumference of the circle (1mm radius) in equal gaps. LDR is in the centre part of the circle, in the core part. two 330 ohms resistors cascade in the red LED. Because this RED color LED voltage drops (1.7V – 2V) is lower than other two-color LEDs. Also,330-ohm is connected resistors series with each green and blue LED. When the working time, the LEDs blink one by one, and the data from the color sheet is captured to the LDR in each LED blink time. And this analog data sends through the Atmega328p Microcontroller for identifying purposes. In the Output, RGB single LED, similarly connect their colors with 330 ohms resistors and this LED indicates the correct color of the object in the programming time.

2.4 Enclosure design

As a product to present in the market it is important to design an enclosure which contains the availability of fixing all the parts in an attractive and economical manner. Here for the color sensor, the enclosure was designed using Solidworks. As PCB contains both sensor part and main circuit in on a single circuit board, enclosure is in a step-down manner which will help to increase the accuracy of color sensor. After the development of body, the downloaded build-up components of 16*2 LCD Display, 4*4 Keypad and 9V battery holder and the relevant measures and spaces are added to the body. And at finally by using assembly feature of solidworks all the components were assembled as one product-Coloris Color Sensor.

The key features of the enclosure are:

- Length=210mm
- Width=120mm
- Hight=49mm
- Thickness= 2mm-5mm
- Density = 1.30 grams per cubic centimetre
- Mass = 569.99 grams
- Volume = 438.46 cubic centimetres
- Surface area = 1789.41 square centimetres
- material –PVC rigid
- Body color-Blue

Here as a color sensor enclosure, it must be a resistant for UV light reflection to avoid unnecessary destruction for LDR reading. So here PVC rigid material is used which is highly resistant for UV light. And more than that at the surrounding of sensor part there is black color to reduce the effect of light reflection in the reading of LDR. And finally, to present as a good outlooking device blue color is applied to the enclosure.

3. Results

In the working condition of the project first, LCD display shows a welcome message and then it shows operation Modes. The keypad keys show the corresponding operating modes

A – Calibration Mode

B – Sensing Mode

C – Input Keypad Values Mode

D – Reset Mode

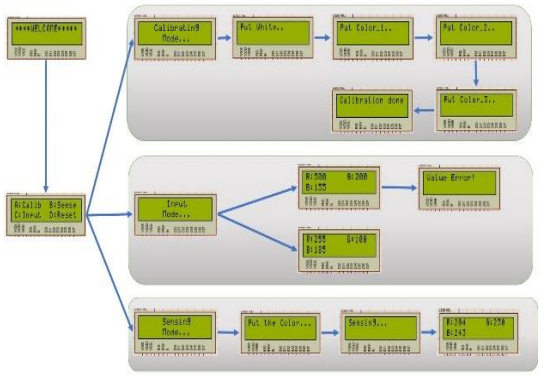


Figure 3.1:LCD Display Views

First, when Press "A" in the Keypad, the calibration mode will on.

Here it'll calibrate the color cards. First, with the white color card, the sensor takes red, green, blue values to the system. Similarly, with the sequence of red color card, green color card, and the blue color card the red, green, blue values will be deducted by the system in the early stage. Then we linearly mapping (Voltage α light density) these values to get white array values and black array values. Here the below picture shows our mapping graph diagram.

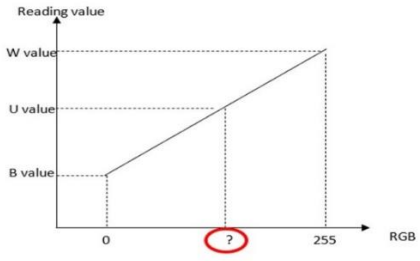


Figure 3.2: Linear mapping concept

According to the reading, white array red, green, and blue values are **781, 792, 625**, and black array red, green, and blue values are **259, 389, 227**. Then our calibration part is done.

After that press **B** in the Keypad to go the Sensing Mode.

Here the device will run with various of color cards to check the output. First, check the red color card in this time Lcd display shows the corresponding red, green, blue values are **223,27,41** in the 8-bit form. And the output RGB led blinks red color to indicate that color card. Similarly, check is done with green color card, blue color card, and violet color card their details are below in the table.

Table 3.1:Sensor part Readings

Color card	Red value	Green value	Blue value	RGB led blink color
Red	223	27	41	Red
Green	108	191	50	Green
Blue	0	43	169	Blue
White	239	255	255	White
Violet	107	25	108	Violet

Then press **C** in the Keypad to go to the Input Keypad Values Mode.

Here the red, green, and blue values are input in the form of 8-bit to get the corresponding output color in the RGB led. this RGB led shows the color regarding the input value and the LED screen will display the most dominating color in the respective color. For example, if we press red, green, and blue values are 255,100,50 then the output RGB led shows orange color and as most dominating value is red the screen display "red". Sometimes if values greater than 255 (8-bit) or any alphabet letters in the keypad are input then the LCD display shows the Value Error message.

Then press **D** in the Keypad to go to the Reset Mode. And do again our prefer works.

The pictures in the Appendix IV shows the **output** result of our RGB led. The accuracy of our sensor results is presented in the pictures. The results were cross checked using the values

from the below website https://csfieldguide.org.nz/en/interactives/rgb-mixer . The values are mostly closer to that online tool values. Therefore, the sensor is in high accuracy, and it will provide a good experience for the user.

4. Discussion

While doing this project, first we individually designed a device with an Arduino board. And then chose the best design and started developing it. For that, we divided the tasks and did our best to complete our tasks successfully.

Since we were beginners, we had to face a lot of problems while doing the project. Lack of experience was a main problem for us. So, we had to learn everything from basics: Arduino coding, AVR coding, Proteus Simulating, PCB designing and Enclosure designing.

The problems we encountered in the project are as follows.

- Due to the Covid-19 situation we had to do everything online, but we were able to keep good communication among group members and manage our project works.
- We were unable to find some components due to the pandemic situation.
- We had not enough pins to connect all the components to Arduino/ATMEGA328P. So, we decided to connect LCD display through I2C module.
- In Proteus we had no way to give different colors to the LDR, we only had the torch LDR to get some values from the LDR.

Although we got correct values from our initial physical device with the Arduino board RGB LED did not give the corresponding color. We finally found that we must use higher resistant to red pin since the voltage of red pin is less than blue and green pins.

- Initially the simulation did not work as we expected for analog-to-digital converting (LDR) and PWM (RGB LED) parts. So, we had to modify the code many times.

The weekly review sessions were very helpful in clarifying our doubts and making sure whether we were moving in the right direction.

5. Acknowledgement

In this pandemic situation it is not an Easy task for us to finish this RGB color sensor to this far. We were novices to all the concepts of Designing and coding ideas and we faced a lot of challenges since this is our first project. There are many personnel behind the accomplishment of this project.

First, we would like to pay our gratitude to our supervisor who always encouraged self-studying and guided us whenever we needed help. The progress meetings held by our supervisor through zoom were very helpful to clear our doubts and discuss issues regarding our project.

In addition, we would like to convey our sincere gratitude to all the lecturers and instructors who were always willing to share their knowledge with us.

It is our duty to thank our ENTC family, without whom we could not have accomplished this project. Everyone in our batch helped each other and learnt everything together.

We developed team spirit working together in this project for about 3 months and all our group members gave their maximum contribution to succeed the given project.

Further, we would like to thank all the people who are not mentioned here for the great support provided even in these pandemic situations.

6. References

[1] https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSA-FBS-3.3v.pdf

[2] https://www.electronicsforu.com/technology-trends/learn-electronics/16x2-lcd-pinout-diagram

[3] https://www.exploreembedded.com/wiki/Basics_of_I2C_with_AVR

[4] http://web.csulb.edu/~hill/ee346/Lectures/20%20C++%20ATmega%20I2C%20Serial%20Comm.pdf

[5] https://circuitdigest.com/microcontroller-projects/light-intensity-measurement-using-ldr-and-atmega8

[6] https://www.youtube.com/watch?v=POaJqnMHofo&t=409s

[7] https://easyeda.com/join?type=project&key=406c4e8896c8eca25e4760e5306c90ac

[8] https://csfieldguide.org.nz/en/interactives/rgb-mixer

7. Appendices

7.1 Appendix I-Circuit Diagram

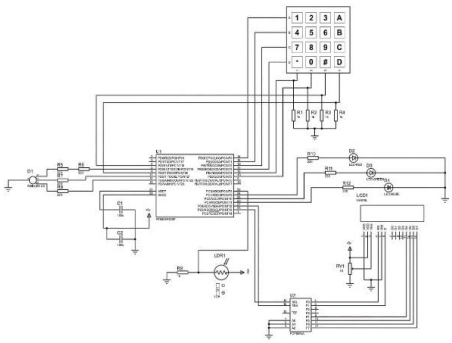


Figure 7.1:Circuit diagram with ATMega328

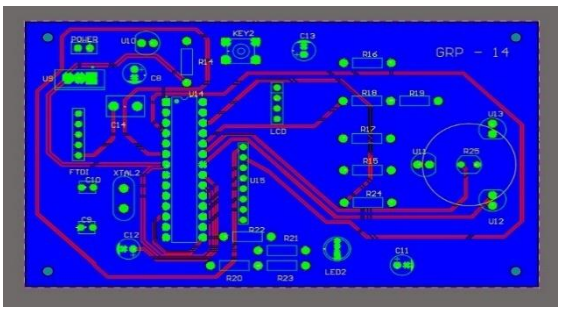


Figure 7.4:PCB top layer

7.2 Appendix II- PCB design

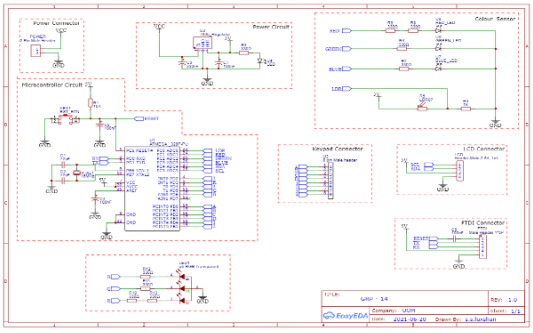


Figure 7.2:Schematic of the main PCB

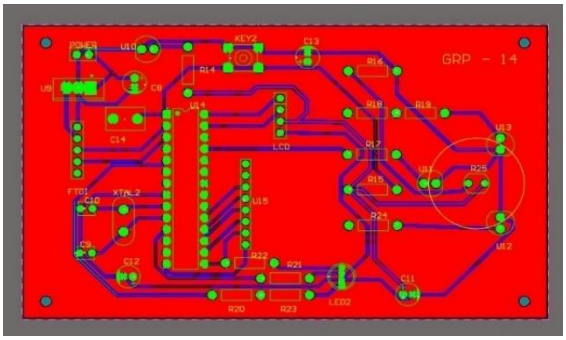


Figure 7.5:PCB bottom layer

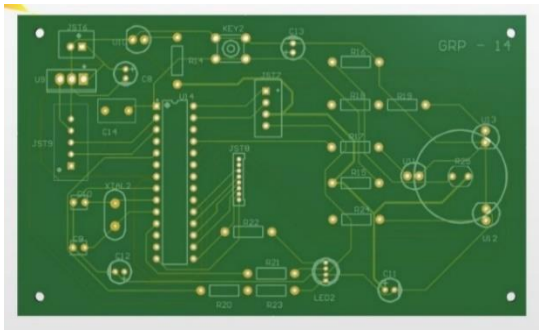


Figure 7.3:Main PCB top view (in Altium software)

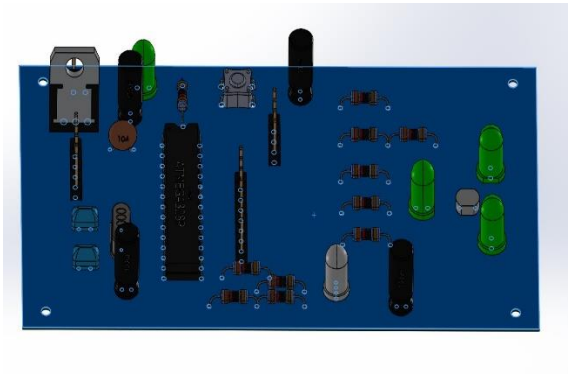


Figure 7.6:3D view of PCB

7.3 Appendix III- Enclosure Design

Figure 7.7:Enclosure Top view

Figure 7.10:Enclosure Interior top part

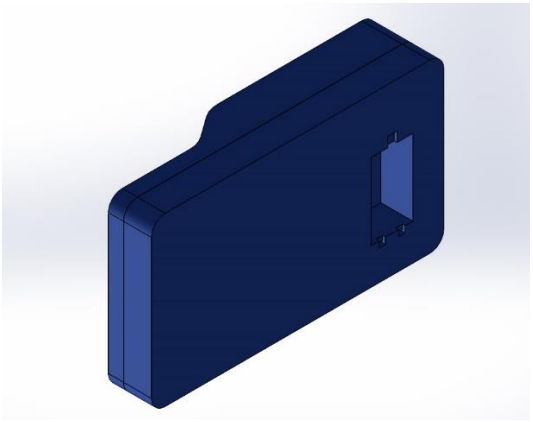


Figure 7.8:Enclosure back view

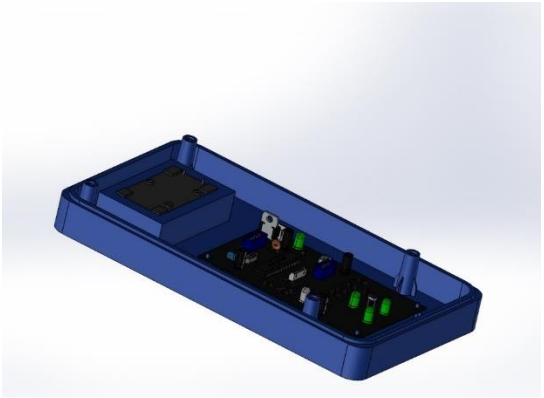


Figure 7.9:Enclosure Interior-bottom part

7.4 Appendix IV- Output

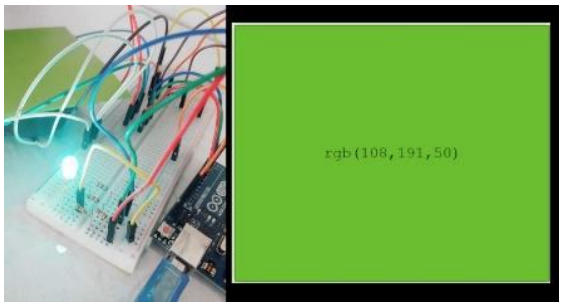


Figure 7.11:Results- Green color

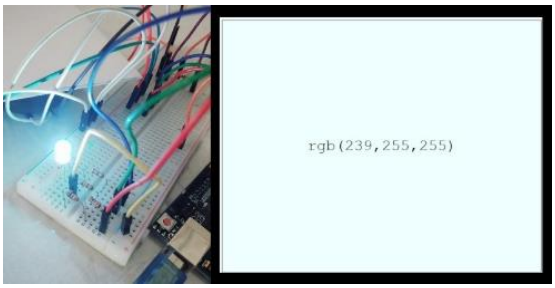


Figure 7.14:Results- White Color

Figure 7.12:Results- Blue color

Figure 7.15:Results- Violet Color

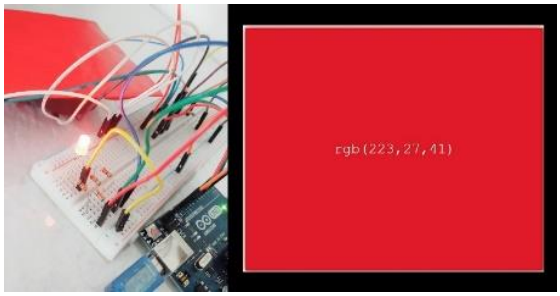
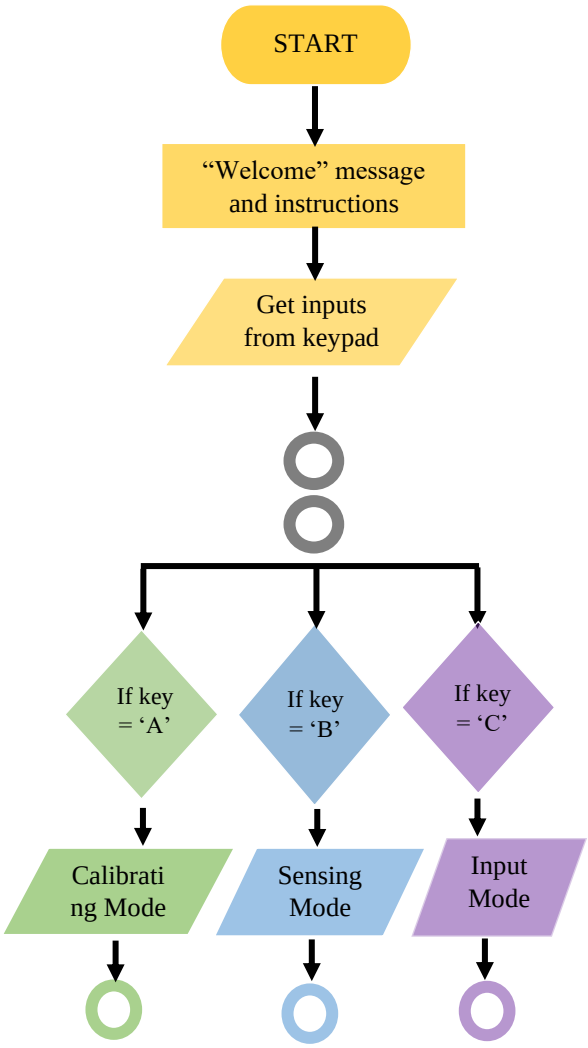
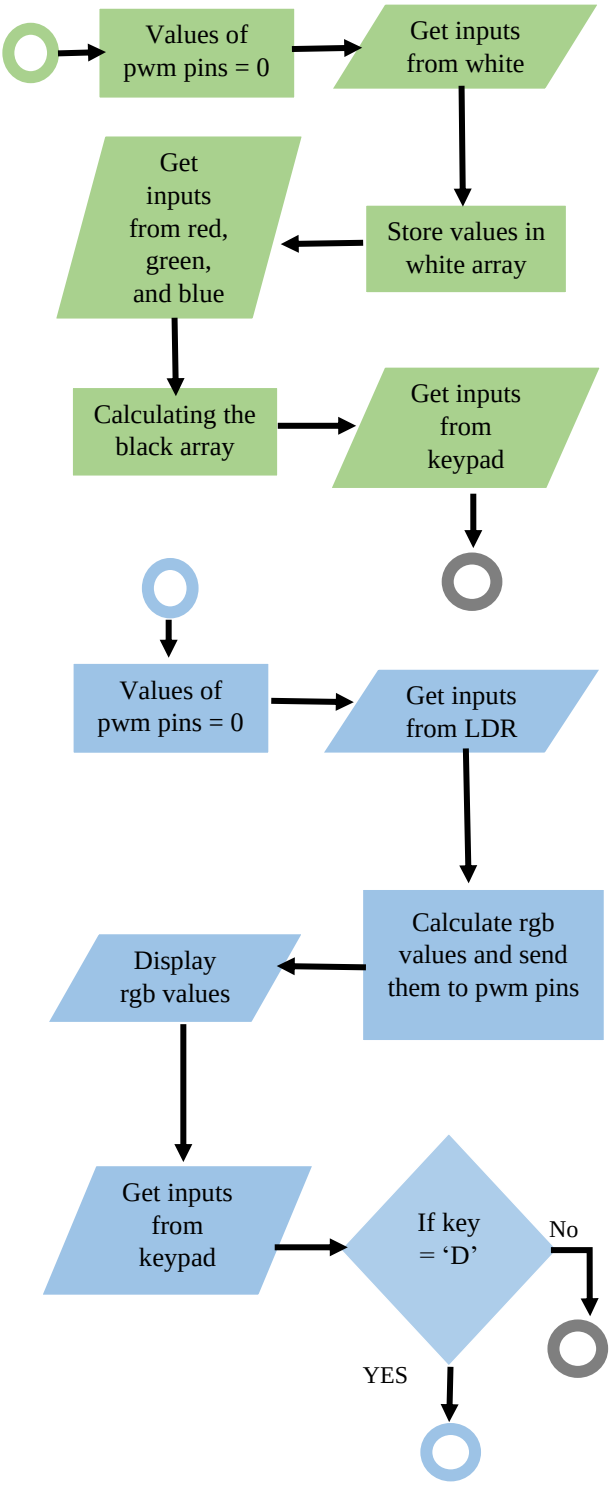
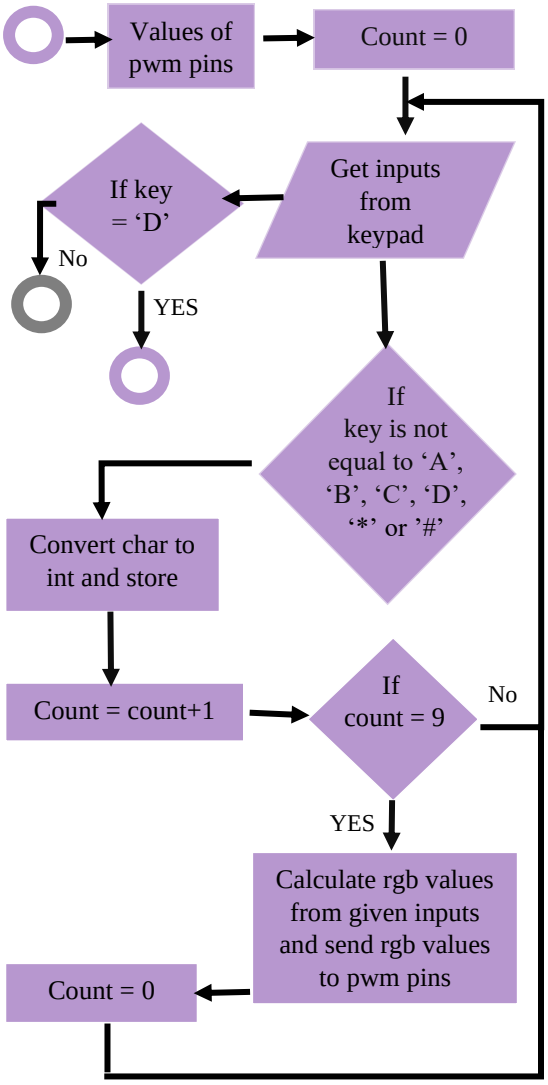


Figure 7.13:Results- Red Color

7.5 Appendix V-Algorithm







7.6 GitHub repository link- RGB Sensor project

https://github.com/MovindiM/RGB-Colour-Sensor

7.7 Appendix VI-Atmega code

```
#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#define Vref 5 //reference voltage for ADC
#define Res 1 //Resistor connected to LDR and GND (1k)

char* getKey(); //Function to get input from keypad

void i2cInit(); //i2c initializing Function
void i2cStart();
void i2cWrite(char x);
char i2cRead();

void lcdInit();          //LCD Initializing Function
void toggle();           //Latching function of LCD
void lcdCmd_hf(char v1); //Function to send half byte command to LCD
void lcdCmd(char v2); //Function to send Command to LCD
void lcdDwr(char v3); //Function to send data to LCD
void lcdString(char *c); //Function to Send String to LCD
void lcdString_xy(char row_1, char position, char *string_00); //Function to type string at x row y column

void inputFunc(int count, char* key, int Arr[9]); //Function to convert char inputs to int and store
void getReading(); //Function to light up r, g, and b led and take values
void sensor(int rgbArr[3]); //Function of sensing Mode
void calibrate(); //Function of calibrating Mode
void rgbCalc(int inputs[9], int rgb[3]); //Function to get RGB values from given input keys
int ldrValue(); //Function to read value of LDR
int getMax(int num_1,int num_2,int num_3); //Function to get maximum value
int mapValue(int value, int maximum, int minimum); //Function to map LDR value to integer number
void rgbLED(char x, int num); //Function to control RGB LED
int checkError(int Arr[3]); //Function to check input rgb values are less than 255
void sensorOutput(int whiteArr[3], int blackArr[3], int rgbArr[3]); //Function of output (sensor Mode)
void pwmLow(); //Function of making the PWM pins(connected to rgb led) low
int inputCount = 0; //global variable to count inputs in input Mode
int main(void)
{
        DDRB = 0XCF; //PORTB4 and PORTB5 are inputs
        DDRD = 0XE8; //PORTD0, PORTD1, PORTD2 and PORTD4 are inputs
        DDRC = 0XFE; //PORTC0 is input others are output

        ADMUX = (1 << REFS0); // setting the reference of ADC  (connected to ADC0 --> 0000)
        ADCSRA = (1 << ADEN) | (1 << ADIE) | (1 << ADPS0) | (1 << ADPS1) | (1 << ADPS2); //ADC enable  ADC interrupt enable ADC pre-scaler selection to 128
        DIDR0 = (ADC0D); //Digital input disable

        int inputMode = 0; //Used to know whether input mode is selected
        int startInput = 0; //Used to stop take numbers as inputs after getting 9 numbers in input mode
        int sensingMode = 0; //Used to know whether sensing mode is selected
```

```
	int inputArr[9] = {0,0,0,0,0,0,0,0,0}; //Array to store input values
	int whiteArr[3] = {0,0,0}; //Array to store highest values(calibrating)
	int blackArr[3] = {0,0,0}; //Array to store lowest values(calibrating)
	int rgbArr_1[3] = {0,0,0}; //Array to store RGB values for sensing mode
	int rgbArr_2[3] = {0,0,0}; //RGB values for input giving mode

	i2cInit();
	i2cStart();
	i2cWrite(0X70);
	lcdInit();

	lcdString("****WELCOME*****");//welcome message
	_delay_ms(2500);
	lcdCmd(0x01);//clear LCD
	lcdString_xy(0,0,"A:Calib");
	lcdString_xy(0,9,"B:Sense");
	lcdString_xy(1,0,"C:Input");
	lcdString_xy(1,9,"D:Reset");

	while (1)
	{
		char* key = getKey(); //get key from key pad
		if (strcmp(key,"NO_KEY")!=0) //checking key has been given or not
		{
			if (strcmp(key,"A")==0) //Calibrating Mode
			{
				pwmLow();
				int inputMode = 0;
				int sensingMode = 0;

	lcdCmd(0x01);//clear LCD

	lcdCmd(0x0C);//cursor off

	lcdString_xy(0,3,"Calibrating");

	lcdString_xy(1,5,"Mode...");
				_delay_ms(1000);

	calibrate(whiteArr,blackArr);//calibrating
			}
			else if (strcmp(key,"B")==0) //Sensing Mode
			{
				pwmLow();
				inputMode = 0;
				sensingMode = 1;
				lcdCmd(0x01);

	lcdCmd(0x0C);//cursor off

	lcdString_xy(0,5,"Sensing");

	lcdString_xy(1,5,"Mode...");
				_delay_ms(1000);

	sensor(rgbArr_1);//sensing

	sensorOutput(whiteArr,blackArr,rgbArr_1);//output
			}
			else if (strcmp(key,"C")==0) //Input Mode (select the input Mode)
			{
				pwmLow();
				inputMode = 1;
				startInput = 1;
				sensingMode = 0;
				inputCount = 0;
				lcdCmd(0x01);
				lcdCmd(0x0C);

	lcdString_xy(0,6,"Input");

	lcdString_xy(1,5,"Mode...");
				_delay_ms(1000);
				lcdCmd(0x01);
				lcdString("R:");

	lcdCmd(0X0E);//cursor on
			}

			else if (strcmp(key,"D")==0) //Used to reset values in sensing and input modes
			{
				pwmLow();
				if (sensingMode == 1)//reset sensor values
				{

	sensor(rgbArr_1);//sensing

	sensorOutput(whiteArr,blackArr,rgbArr_1);//output
				}
				else if (inputMode==1)//reset inputs
				{

	startInput = 1;
	inputCount = 0;
	lcdCmd(0x01);
	lcdString("R:");
	lcdCmd(0X0E);//Cursor on
				}
			}

			else if ((strcmp(key,"#")!=0) && (strcmp(key,"*")!=0) && (inputMode==1) && (startInput==1)) //if input mode selected and key is a number
			{
				//get inputs

	inputFunc(inputCount,key,inputArr);
				inputCount++;
				if (inputCount==9)
				{

	inputCount = 0;

	startInput = 0;

	lcdCmd(0x0C); //cursor off

	_delay_ms(1000);

	rgbCalc(inputArr,rgbArr_2);//calculate rgb values
					int Error = checkError(rgbArr_2);//check error
					if (Error == 0)
					{

	rgbLED('r',rgbArr_2[0]); //Red

	rgbLED('g',rgbArr_2[1]); //Green

	rgbLED('b',rgbArr_2[2]); //Blue
					}
					else
					{
```

```
		lcdCmd(0x01);

		lcdString("Value Error!"); //if value>255
                                                        }
                                                }
                                        }
                                }
                        }
}

//get inputs from keypad
char* getKey()
{
		char* key = "NO_KEY";
		PORTB = 0X01; //1st row
		if (PINB == 0X11)
		{
                        key = "1";
		}
		if (PINB == 0X21)
		{
                        key = "2";
		}
		if ((PINB == 0X01) && ((PIND & 0X04) == 0X04))
		{
                        key = "3";
		}
		if ((PINB == 0X01) && ((PIND & 0X10) == 0X10))
		{
                        key = "A";
		}
		_delay_ms(40);

		PORTB = 0X02; //2nd row
		if (PINB == 0X12)
		{
                        key = "4";
		}
		if (PINB == 0X22)
		{
                        key = "5";
		}
		if ((PINB == 0X02) && ((PIND & 0X04) == 0X04))
		{
                        key = "6";
		}
		if ((PINB == 0X02) && ((PIND & 0X10) == 0X10))
		{
                        key = "B";
		}
		_delay_ms(40);

		PORTB = 0X04; //3rd row
		if (PINB == 0X14)
		{
                        key = "7";
		}
		if (PINB == 0X24)
		{
                        key = "8";
		}
		if ((PINB == 0X04) && ((PIND & 0X04) == 0X04))
		{
                        key = "9";
		}
		if ((PINB == 0X04) && ((PIND & 0X10) == 0X10))
		{
                        key = "C";
		}
		_delay_ms(40);

		PORTB = 0X08; //4th row
		if (PINB == 0X18)
		{
                        key = "*";
		}
		if (PINB == 0X28)
		{
                        key = "0";
		}
		if ((PINB == 0X08) && ((PIND & 0X04) == 0X04))
		{
                        key = "#";
		}
		if ((PINB == 0X08) && ((PIND & 0X10) == 0X10))
		{
                        key = "D";
		}
		_delay_ms(40);
		return key;
}

void i2cInit(){
		TWBR = 0x62; //	Baud rate is set by calculating
		TWCR = (1<<TWEN); //Enable I2C
		TWSR = 0x00; //Pre-scaler set to 1
}

//Start condition
void i2cStart(){
		TWCR = (1<<TWINT) | (1<<TWEN) |
(1<<TWSTA);		//start condition
		while (!(TWCR & (1<<TWINT))); //check for start
condition

}

//I2C stop condition
void i2cWrite(char x){
		TWDR = x; //Move value to I2C
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C
and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
}

char i2cRead(){
		TWCR  = (1<<TWEN) | (1<<TWINT);		//Enable
I2C and clear interrupt
		while (!(TWCR & (1<<TWINT))); //Read successful
with all data received in TWDR
		return TWDR;
}

void toggle()
{
		TWDR |= 0x02; //PIN En de la LCD en = 1;
Latching data in to LCD data register using High to Low signal
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C
and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		_delay_ms(1);
		TWDR &= ~0x02;	//PIN del Enable de la LCD en
= 0;
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C
and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
}

void lcdCmd_hf(char v1)
{
		TWDR &=~0x01; //PIN RS de la LCD rs = 0;
Selecting register as Command register
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C
and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		TWDR &= 0x0F; //clearing the Higher 4 bits
		TWCR = (1<<TWINT) | (1<<TWEN);		//Enable
I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
```

```
		TWDR |= (v1 & 0xF0); //Masking higher 4 bits and sending to LCD
		TWCR = (1<<TWINT) | (1<<TWEN);	//Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		toggle();
}

//Send command to LCD
void lcdCmd(char v2)
{
		TWDR&=~0x01; //rs = 0; Selecting register as command register
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		TWDR &= 0x0F; //clearing the Higher 4 bits
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		TWDR |= (v2 & 0xF0); //Masking higher 4 bits and sending to LCD
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		toggle();

		TWDR &= 0x0F; //clearing the Higher 4 bits
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		TWDR |= ((v2 & 0x0F)<<4); //Masking lower 4 bits and sending to LCD
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		toggle();
}

//send data to LCD
void lcdDwr(char v3)
{
		TWDR|=0x01;		//rs = 1; Selecting register as command register
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		TWDR &= 0x0F; //clearing the Higher 4 bits
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		TWDR |= (v3 & 0xF0); //Masking higher 4 bits and sending to LCD
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		toggle();

		TWDR &= 0x0F; //clearing the Higher 4 bits
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		TWDR |= ((v3 & 0x0F)<<4); //Masking lower 4 bits and sending to LCD
		TWCR = (1<<TWINT) | (1<<TWEN); //Enable I2C and clear interrupt
		while  (!(TWCR &(1<<TWINT)));
		toggle();
}

//initiating LCD
void lcdInit()
{
		lcdCmd_hf(0x30); //Sequence for initializing LCD
		lcdCmd_hf(0x30); //Sequence for initializing LCD
		lcdCmd_hf(0x20); //Sequence for initializing LCD
		lcdCmd(0x28); //Selecting 16 x 2 LCD in 4Bit mode
		lcdCmd(0x0C); //Display ON Cursor OFF
		lcdCmd(0x01); //Clear display
		lcdCmd(0x06); //Cursor Auto Increment
		lcdCmd(0x80); //1st line 1st location of LCD
}

//send string to LCD
void lcdString(char *c)
{
		while(*c != 0) //Wait till all String are passed to LCD
		lcdDwr(*c++); //Send the String to LCD
}

//sent string to x row y column of LCD
void lcdString_xy(char row_1, char position, char *string_00)
{
		if ((row_1 == 0) && (position<16))
		{
			lcdCmd((position & 0x0F) | 0x80);
			lcdString(string_00);
		}
		else if ((row_1 == 01) && (position<16))
		{
			lcdCmd((position & 0x0F) | 0xC0);
			lcdString(string_00);
		}
}

//convert char input to int and store(used in input mode)
void inputFunc(int count, char* key, int Arr[9])
{
		lcdString(key);
		_delay_ms(300);
		Arr[count] = atoi(key);
		if (count==2)
		{
			lcdString_xy(0,10,"G:");
		}
		else if (count==5)
		{
			lcdString_xy(1,0,"B:");
		}
}

//getting LDR resistant
int ldrValue()
{
		//float Vin = voltage at ADC pin
		float LDR; //resistant of LDR
		ADCSRA |= (1 << ADSC); // ADC start conversion
		_delay_ms(100);
		/*ADC = (Vin*1024)/Vref
		  Vin = (ADC*Vref)/1024
		  LDR = (1/Vin)*(Vref-Vin)
		  LDR = (1024/ADC)-1*/
		LDR = (1024/ADC)-1;
		int x = round(LDR);
		return x;
}

//light up r, g, b led and take values(used in both calibration and sensor mode)
void getReading(int rgbArr[3])
{
		PORTC |= 0X02; //light red led
		_delay_ms(300);
		//get readings
		rgbArr[0] = ldrValue();
		PORTC &= 0XFD; //off red led
		_delay_ms(1000);
		PORTC |= 0X04; //light green led
		_delay_ms(300);
		//get readings
```

```
		rgbArr[1] = ldrValue();
		PORTC &= 0XFB; //off green led
		_delay_ms(1000);
		PORTC |= 0X08; //light blue led
		_delay_ms(300);
		//get readings
		rgbArr[2] = ldrValue();
		PORTC &= 0XF7; //off blue led
}

//Sensing Mode
void sensor(int rgbArr[3])
{
		lcdCmd(0x01); //clear LCD display
		lcdString("Put the Color...");
		_delay_ms(2000);
		lcdCmd(0x01);
		lcdString("Sensing...");
		getReading(rgbArr);
		_delay_ms(50);
}

//get maximum
int getMax(int num_1,int num_2,int num_3)
{
		int Maximum = 0;
		int Arr[3] = {num_1,num_2,num_3};
		for (int i=0;i<3;i++)
		{
			if (Arr[i]>Maximum)
			{
				Maximum = Arr[i];
			}
		}
		return Maximum;
}

//Function of calibrating mode
void calibrate(int whiteArr[3],int blackArr[3])
{
		int Arr_1[3] = {0,0,0};
		int Arr_2[3] = {0,0,0};
		int Arr_3[3] = {0,0,0};
		lcdCmd(0x01);
		lcdString("Put White..."); //white
		_delay_ms(2000);
		lcdCmd(0x01);
		lcdString("Calibrating...");
		_delay_ms(10);
		getReading(whiteArr);//lighting up LEDs and
sensing
		lcdCmd(0x01);
		lcdString("Put Color_1.."); //Color1
		_delay_ms(2000);
		lcdCmd(0x01);
		lcdString("Calibrating...");
		_delay_ms(10);
		getReading(Arr_1);//lighting up LEDs and sensing
		lcdCmd(0x01);
		lcdString("Put Color_2..");
		_delay_ms(2000);
		lcdCmd(0x01);
		lcdString("Calibrating...");
		_delay_ms(10);
		getReading(Arr_2);//lighting up LEDs and sensing
		lcdCmd(0x01);
		lcdString("Put Color_3..");
		_delay_ms(2000);
		lcdCmd(0x01);
		lcdString("Calibrating...");
		_delay_ms(10);
		getReading(Arr_3);//lighting up LEDs and sensing
		lcdCmd(0x01);
		lcdString("Calibration done");
		//light intensity and resistant of LDR are inversely
proportional. to get darkest values maximum resistant should be
taken
		blackArr[0] = getMax(Arr_1[0],Arr_2[0],Arr_3[0]);
		blackArr[1] = getMax(Arr_1[1],Arr_2[1],Arr_3[1]);
		blackArr[2] = getMax(Arr_1[2],Arr_2[2],Arr_3[2]);
}

//Function to calculate RGB values from given inputs
void rgbCalc(int inputs[9], int rgb[3])
{
		rgb[0] = (inputs[0]*100)+(inputs[1]*10)+inputs[2];
		rgb[1] = (inputs[3]*100)+(inputs[4]*10)+inputs[5];
		rgb[2] = (inputs[6]*100)+(inputs[7]*10)+inputs[8];
}

//Function to map LDR value to integer number
int mapValue(int value, int maximum, int minimum)
{
		float x=255.00;
		int a = (maximum-minimum);
		int b = (maximum-value);
		float c = x*b/a;
		int y = round(c);
		if (y<0)
		{
			y = 0;
		}
		return y;
}

//Function to control RGB LED
void rgbLED(char x, int num)
{
		TCCR0B = (1 << CS00) | (1 << CS01); //pre scaler
64 ( for OC0A and OC0B)
		TCCR2B = (1 << CS20) | (1 << CS21); //pre scaler
64 ( for OC2B)

		if (x == 'r')
		{
			TCCR2A = (1 << WGM21) | (1 <<
WGM20) | (1 << COM2B1); // fast PWM non inverting
			OCR2B = num;
		}
		else if (x == 'g')
		{
			TCCR0A |= (1 << WGM01) | (1 <<
WGM00) | (1 << COM0B1); // fast PWM non inverting
			OCR0B = num;
		}
		else if (x == 'b')
		{
			TCCR0A |= (1 << WGM01) | (1 <<
WGM00) | (1 << COM0A1); //fast PWM non inverting
			OCR0A = num;
		}
}

//Function to check input rgb values are less than 255
int checkError(int Arr[3])
{
		int Error = 0;
		for (int i = 0; i<3; i++)
		{
			if (Arr[i] > 255)
			{
				Error = 1;
				break;
			}
		}
		return Error;
}

//Function of output (sensor Mode)
```

```
void sensorOutput(int whiteArr[3], int blackArr[3], int
rgbArr[3] )
{
		int value;
		char str[20];
		lcdCmd(0x01); //clear LCD display
		//Red
		value =
mapValue(rgbArr[0],blackArr[0],whiteArr[0]);
		lcdString("R:");
		sprintf(str, "%d", value);
		lcdString(str);
		rgbLED('r',value);
		//Green
		value =
mapValue(rgbArr[1],blackArr[1],whiteArr[1]);
		lcdString_xy(0,10,"G:");
		sprintf(str, "%d", value);
		lcdString(str);
		rgbLED('g',value);
		//Blue
		value =
mapValue(rgbArr[2],blackArr[2],whiteArr[2]);
		lcdString_xy(1,0,"B:");
		sprintf(str, "%d", value);
		lcdString(str);
		rgbLED('b',value);
}

//making the PWM pins(connected to rgb led) low
void pwmLow()
{
		OCR2B = 0;
		OCR0B = 0;
		OCR0A = 0;
}
```