Department of Electronic & Telecommunication Engineering

University of Moratuwa

EN1093 Laboratory Practical – I

PROJECT REPORT

RGB COLOR SENSOR

Group Members:	ABEYSINGHE A.L.R.	190012X
	ABEYSINGHE W.A.M.S.Y.	190014F
	ABEYWICKRAMA K.C.S.	190018V
	ADIKARI A.M.A.D.	190021A

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

05th January 2018

RGB COLOUR SENSOR

Lakdilu Abeysinghe, Yohan Abeysinghe, Chamod Abeywickrama, Ashen Adikari

Department of Electronics and Telecommunication, University of Moratuwa

***Abstract*— A colour sensing device is made using the RGB model approach. This device has three operational modes. Calibration mode, Sensor mode and Given value RGB mode. The calibration mode is to calibrate the device before the test run and this is done in real-time using only three colours. The goal of the Sensor mode is to identify the red, green, blue content in the colour of a given surface and display them using an LCD. In given value RGB mode a keypad is used to get user inputs and light is illuminated according to those inputs. This colour sensor is developed without using any third-party libraries and only using one PWM pin in the microcontroller to control an RGB LED. We explored different algorithms to achieve high accuracy and finally ended up with a simple endpoint method algorithm. An end product is made including the custom PCB design and a user-friendly 3D enclosure.**

***Index Terms*—** RGB color sensor, ATmega328p, PWM

1. INTRODUCTION

1.1. Objectives

There are several types of colour sensor models namely RGB, CMYK and Grayscale. The RGB model is the most famous and frequently used one. These RGB colour sensors play a vital role in image processing, digital signal processing, object detection and these are used in security, biomedical and mechanical industries. Here we were expecting to achieve the following objectives.

- Determining the RGB composition of the colour of a given surface and lighting an RGB LED to match that colour.
- Lighting an RGB LED using imputed red, green and blue values.
- Calibrating the device before the test using only three colours.
- Using only one PWM pin to operate all the three colours in RGB LED.
- Programming is done only using C in AVR studio and without using any third-party libraries

1.2. Purpose

A device is made according to the above criteria using the Atmega328P as the micro-controller. A keypad is used to get the inputs. An LCD is used to output the values. A custom PCB is designed using Altium designer to our requirements. A 3D enclosure that houses all these components is designed using Solidworks.

2. METHOD

2.1. List of Components Used

2.1.1. Sensor Part

- 3 LEDs of Red, Green and Blue - These LEDs are

  1 error6 warnings used to light up the measuring surface in a specific colour at a time (Red, Green or Blue). They are used instead of a one RGB since in a RGB the lights from separate colors are focused to separate points. But since we are using one LDR we need to spot all the light at the same point.

- LDR sensor - To measure the intensity of the colour reflection received from the measuring surface, when it is lighted with respective colours using LEDs. Here the photodiodes were used for some testings instead of LDR, but finally LDR is used due to its easy manipulations.

2.1.2. Main Circuit

- ATmega328P - This is a commonly used microcontroller in the Electronics field. It is used since it is easy to do the testing part by uploading the code to Arduino without having to buy a microcontroller beforehand.
- 16*2 LCD Display - This is used as the interface between user and the programme.
- 4*3 Keypad - To get the inputs.

- RGB LED - To show the respective colours of the user input.
- A733 Transistors - This is used to light up RGB LED by using only one PWM pin. Here we used 2 transistors to build a simple NOR gate..
- 100pF Capacitors - One is used in the AREF pin of the Microcontroller to remove noise in ADC. Another is used for 9V to 5V regulator.
- 22pF Capacitors - Used for the Crystal.
- 9V Battery

2.2. Our Approach for Sensing

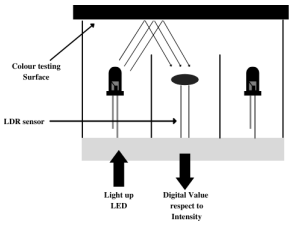


Fig. 1

An LDR and three separate LEDs of red, green and blue colour are housed in the Sensing part as shown in the Fig.1. One LED is illuminated at a time, and the intensity of that reflected light from the surface of measuring is read by the LDR. Since the Sensor housing does not allow any other light to come in, it is considered that all the measured intensity is only due to the reflections. Then these LDR readings are used for all the work throughout the project.

2.3. Algorithm

2.3.1. Linear Regression Method

First, 12 frequently used colours were selected as our dataset. Out of those, 5 colours (red, green, blue, black, white) were chosen as calibration colours. The LDR readings for these colours with respect to all three Red, Green and Blue were taken. These values were plotted on a graph (Fig.2). Then these data points for each colour were separately selected and a common line for these data points of only the 5 calibration colours was drawn using the linear regression method.

As you can see in Fig.3 this method resulted in high errors in both calibration colours as well as in the other 7 colours in our data set. So we thought there might be some correlation between the three colours and we might not be able to consider them separately. .

So it was decided to test this by considering all three colours at once using a machine learning approach.

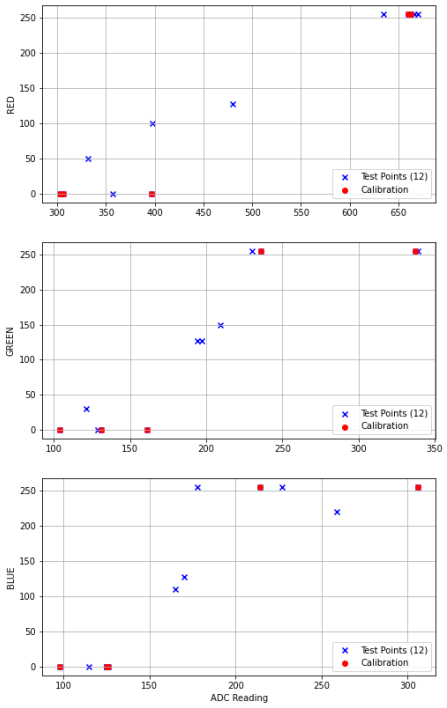


Fig. 2

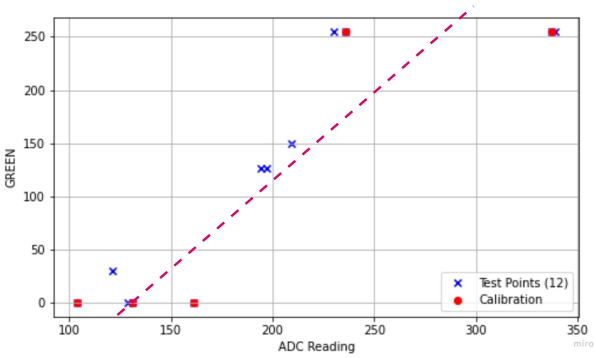


Fig. 3

2.3.2. Using Scikit Learn Library and Polynomial Regression

Then the dataset was increased to 36 colours and the values were plotted in a graph as shown in Fig.4. We trained both linear regression model and a polynomeial regression model by using only 5 colours (Since we are allowed only to use 5 colours for calibration). Then this trained model was tested on the rest of the 31 colours. As these models tend to work better with larger data sets, this approach was not successful.

Although this is generally a good method, we decided it is not suitable for this project with the constraints that we have to adhere to. Even if we are able to successfully implement this method, it is harder to implement this model in a low-level micro-controller like Atmega328P in real-time. So it is harder to do real-time calibration at the testing ground using this. It

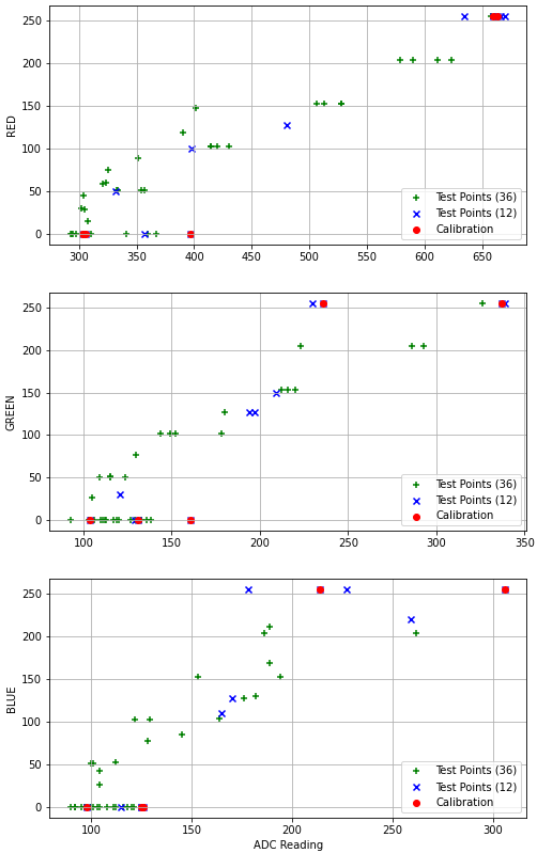


Fig. 4

was decided that the compromise we have to make for a slight increase in accuracy is too much.

2.3.3. *Correlation Analysis*

*	r	g	b
R	0.945	0.511	0.148
G	0.665	0.927	0.057
B	-0.218	-0.011	0.896

Fig. 5

Then the correlation among the colours was checked using the Pandas library. In Fig. 5 r,g,b refers to the readings of the LDR and R,G,B refers to the expected values. As shown in the correlation coefficients table in Fig. 5, we found out that the correlation among the colours is very low (The values that are not in the diagonal of the table is very low). So, it is again decided that it is possible to consider that these colours behave independently.

As seen from the Fig. 5, the correlation between the reading for the 3 colours and the respective expected value is close to 1 (values in the diagonal of the table). Therefore we again considered that we should go for a linear model.

2.3.4. *End Point Method*

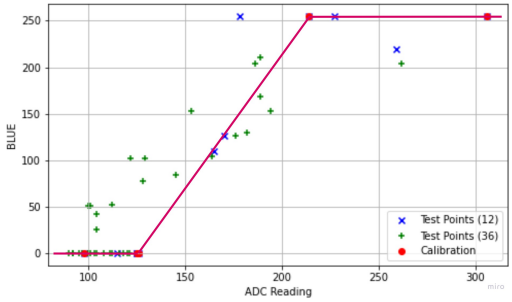


Fig. 6

Then again a graph was plotted using only 5 calibration colours. Then we tried to draw a line that gives the highest accuracy for the most commonly used colours (common colours mostly have values near 0,127 and 255). The accuracy of the other colours was not prioritized very much, since the overall accuracy depends mostly on the common colours. To accomplish this it was seen that the line was drawn by connecting the highest reading in the lower end of the plot (closer to the 0 colour value) and the lowest reading in the higher end of the plot (closer to the 255 colour value) is very suitable. The values for the colours below this considered highest reading in the lower end is taken as 0 and the values for the colours above this considered lowest reading in the highest end is taken as 255. So it was decided to use this line for the algorithm. Then after the testing is done later it is found that the white and black colours are not needed for this process. So we used only 3 colours for our calibration thereafter. The graph with the line used in this process for the green colour is shown in Fig.6.

2.3.5. *Error Correction*

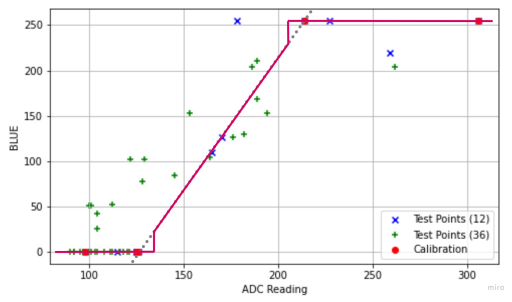


Fig. 7

After the latter testing, we found out that some errors for the readings are happening near the endpoints

(closer to colour values 0 and 255) due to the imperfections in the LDR. Since this region is crucial for the common colours we decided to do an error correction.

As seen in Fig.7 the values of the colours that are below 15 were made 0 by force and the values of the colours above 240 were made 255 by force. This finalized version resulted in great accuracy for commonly used colours and good accuracy in other colours.

2.4. One PWM Implementation

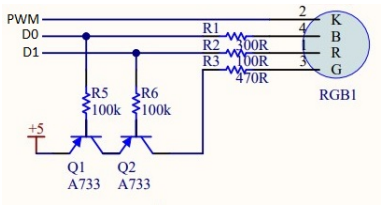


Fig. 8

D1	D0	Output
0	0	Green
0	1	Blue
1	0	Red

Fig. 9

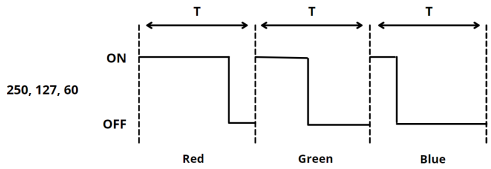


Fig. 10

The objective of this is to control the RGB LED using only one PWM pin in the micro-controller by sending signals to the 3 pins in the RGB LED. But in the approach we used, we send only one PWM signal to the common pin of the RGB LED and only one colour is lighted up at a time. Here the normal PWM behaviour of the micro-controller is used to control this common pin of the RGB LED. The timer overflow interrupts were used to switch between the colours while using only this common pin. Since our IC runs at a frequency of 16MHz, this switching operation is very fast. So, although only one colour is lighted at an instance, it is effectively seen as the RGB LED is lighting all the three red, green and blue colours at the same time. Due to this, this method has no visible difference to the eye or to a camera when compared to using 3 PWM pins.

In this process, the PWM pin is connected to the common pin. To light up one colour in the RGB at a time we used a NOR gate instead of a prebuilt Demux IC. This NOR gate is built using 2 PNP transistors. Here 2 Digital IO pins are used to control this NOR gate.

Due to the 16MHz frequency in our micro-controller, the clock cycle is 62.5ns (T). The timer is configured so the timer counter increments by clock cycle (62.5ns, no prescaling) and the timer overflow interrupt is enabled. Since a common cathode RGB is used, this PWM pin operates in inverting mode. The normal PWM behaviour is used to light up a colour within one cycle as shown in Fig.10. When the timer overflow interrupt occurs, the colour is changed to the next one and the PWM register gets updated to the next colour.

2.5. AVR Implementation

First, all the algorithms and the codes were tested on the selected hardware using Arduino. After confirming the process, they were moved to C using AVR studio. Here we developed separate custom core libraries to control the hardware components in our device (LCD, Keypad, PWM and Sensor) and to implement a calibration algorithm.

- Sensor core - Controls the ADC and reads the analogue value of the LDR sensor.
- LCD core - controls the LCD
- Keypad core - controls the Keypad
- Calib core - Implementation of the calibration algorithm.
- PWM core - Control RGB LED using 1 PWM pin

Appendix I represents this core and the main code workflow in a flow chart

2.6. Schematic

The schematics for all the circuits were designed using Altium software. The total schematic diagram is shown in Appendix II. It has 2 parts.

1. Main Circuit - (Power Supply Unit, IC, RTL NOR gate, Headers)
2. Sensor Circuit - (LDR, 3 LEDs)

2.7. PCB Design

There were two separate PCBs to ensure maximum space utilization and ease of use.

1. Main PCB (Fig. 11)

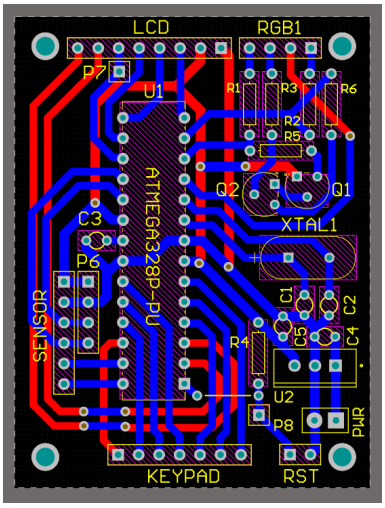


Fig. 11

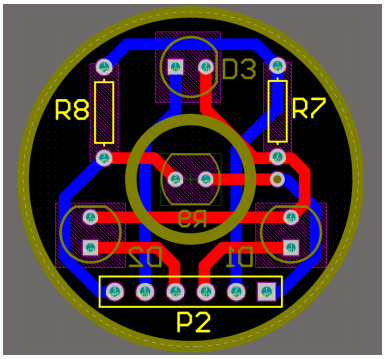


Fig. 12

Fig. 13

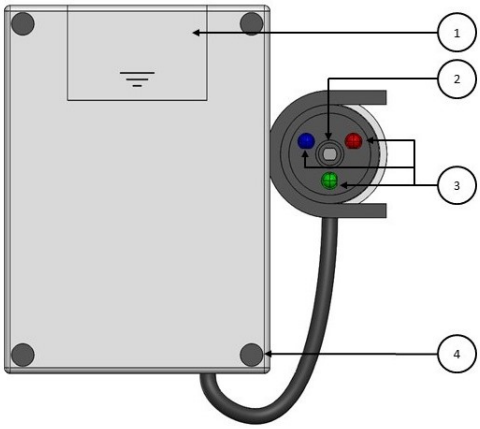


Fig. 14

2. Sensor part PCB - Sensor with the three LEDs were fixed as a separate unit to the enclosure. This is made in circular shape to match the enclosure of the sensor part (Fig. 12).

They were designed using the Altium designer. Double layer PCBs were used for compactness. 30mill path widths are used for normal nets and 50 mill path widths are used for power paths.

2.8. Enclosure Design

Enclosure Design has two main parts, the Main box and the Sensor part..

1. Main Box - This has the top part that can be separated, the bottom part that can be removed by removing 4 screws and the middle part. A hanger is attached to the middle part which acts as the holder to the Sensor part. LCD Display, power button, RGB LED and the keypad are all mounted on the top face. Due to this, it is easier for the user to interact with the device. This part has four rubber feet attached to the bottom part, which helps for the proper grip with the ground

2. Sensor Part - Sensor parts house the Sensor PCB, three LEDs and one LDR. It is attached to the main part using a single large wire. The sensor part can be hung and locked into the Main box using the hanger.

This enclosure has no visible screws and the dimensions are as follows. Main box: (130m x 94cm x 60mm) and Sensor: (diameter – 5.5mm / height – 4cm)

2.9. GIT Repository

GIT is used throughout the project for allocating the tasks among four members. There were four separate branches for each member, so anyone could see each member's contribution. Then the final work is merged with the main branch. Each one has separate folders named simulations, src (for Arduino and AVR work). Then there were separate branches for PCB and enclosure designs. When converting the code from Arduino to AVR, we build our own libraries named "cores". Each member was responsible for a core. These cores were pushed into respective branches and they were later merged with the main branch. All the work we did can be viewed at our git hub repository using the following link. `https://github.com/KCSAbeywickrama/rgb-g2`

3. RESULTS

Fig. 15

We achieved all three expected outcomes of this project.

1. The calibration is done in real-time before the test using only 3 colours.
2. The accuracy of the given value RGB mode is very high. It is able to light up the given input values to the exact colour. (Fig.15)
3. There was very high accuracy in the sensing mode too. The reading for the colours with the values near the endpoint was almost 100%. The accuracy for the values in the middle points was good as well. (Fig.16)

The following table shows the deviations of the expected value with the actual reading in our product. So, it is evident that the errors are very low and the accuracy is very good. And most of the errors are also due to the testing colours being not very accurate since our printers do not have perfect colours.

4. DISCUSSION

We were able to achieve all the expected outcomes of this project. The calibration is done using only 3 colours before the test. An RGB was lit according to the values taken from the keypad. The colour composition of a surface is identified with great accuracy and the composition was shown in an LCD display. An RGB was lit to match the colour of the surface.

Apart from these, all the programming parts were done using the C language in AVR studio and no third party libraries were used. Only one PWM pin is used to control the RGB LED.

PCB design was done using Altium software and it had no rule violations. A 3D enclosure was developed using Solidworks which has proper dimensions to house all the components including the PCB. Although we were unable to print the PCB and make the final enclosure due to the current situation in the country, all the algorithms and codes were implemented on a breadboard. So the project can be considered a success.

Fig. 16

5. ACKNOWLEDGEMENT

There were many people behind the success of this project. First, our gratitude goes to our supervisor Mr Ashwin De Silva, who always encouraged us and gave us guidance and insight in each and every one of our weekly meetings. Then our sincere gratitude goes to all the lecturers and instructors who were always there to help us whenever needed. The ENTC family is a place where everyone shares everything. We would like to help our batchmates with all the help they gave. This project is the outcome of our four group members working together and giving their maximum effort for four consecutive months.

6. REFERENCE

[1] T. N. Ghorude, T. and A. D. Shaligram, A., 2012. Microcontroller Based Color Measurement Using Rgb Leds. International Journal of Scientific Research, 1(3) , pp.93-95.

[2] author:, M., 2021. Simple RGB Color Detector Using Arduino. [online] Instructables. Available at: https://www.instructables.com/Simple-RGB-Color-Detector-Using-Arduino [Accessed 5 August 2021].

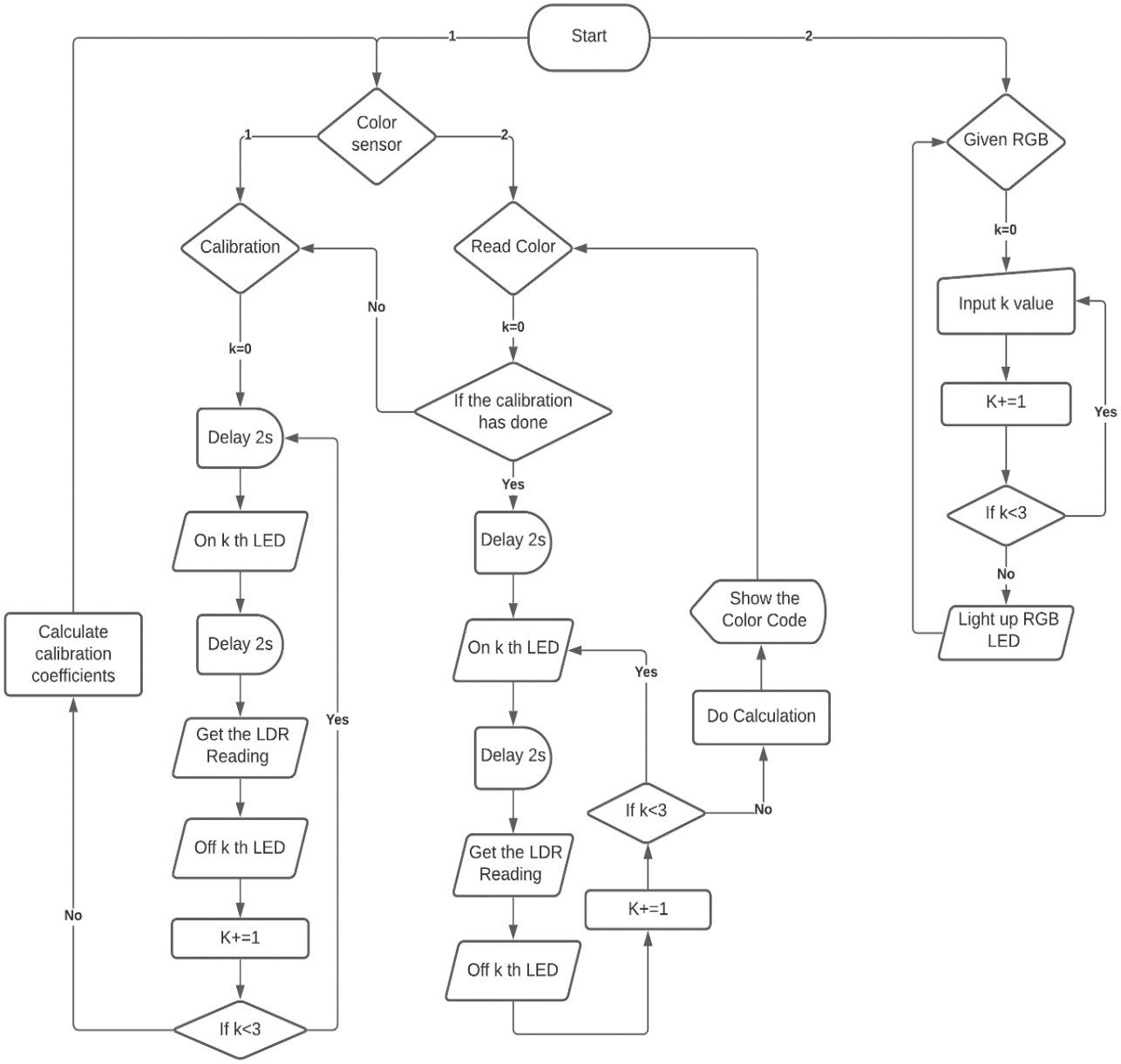
[3] Instructables.com. 2021. [online] Available at: https://www.instructables.com/Using-an-RGB-LED-to-Detect-Colours [Accessed 29 July 2021].

[4] it?, L. and Ooijen, W., 2021. LDR + RGB Led = Color sensor. How to calibrate it?. [online] Electrical Engineering Stack Exchange. Available at: https://electronics.stackexchange.com/questions/173414/ldr-rgb-led-color-sensor-how-to-calibrate-it [Accessed 30 June 2021].

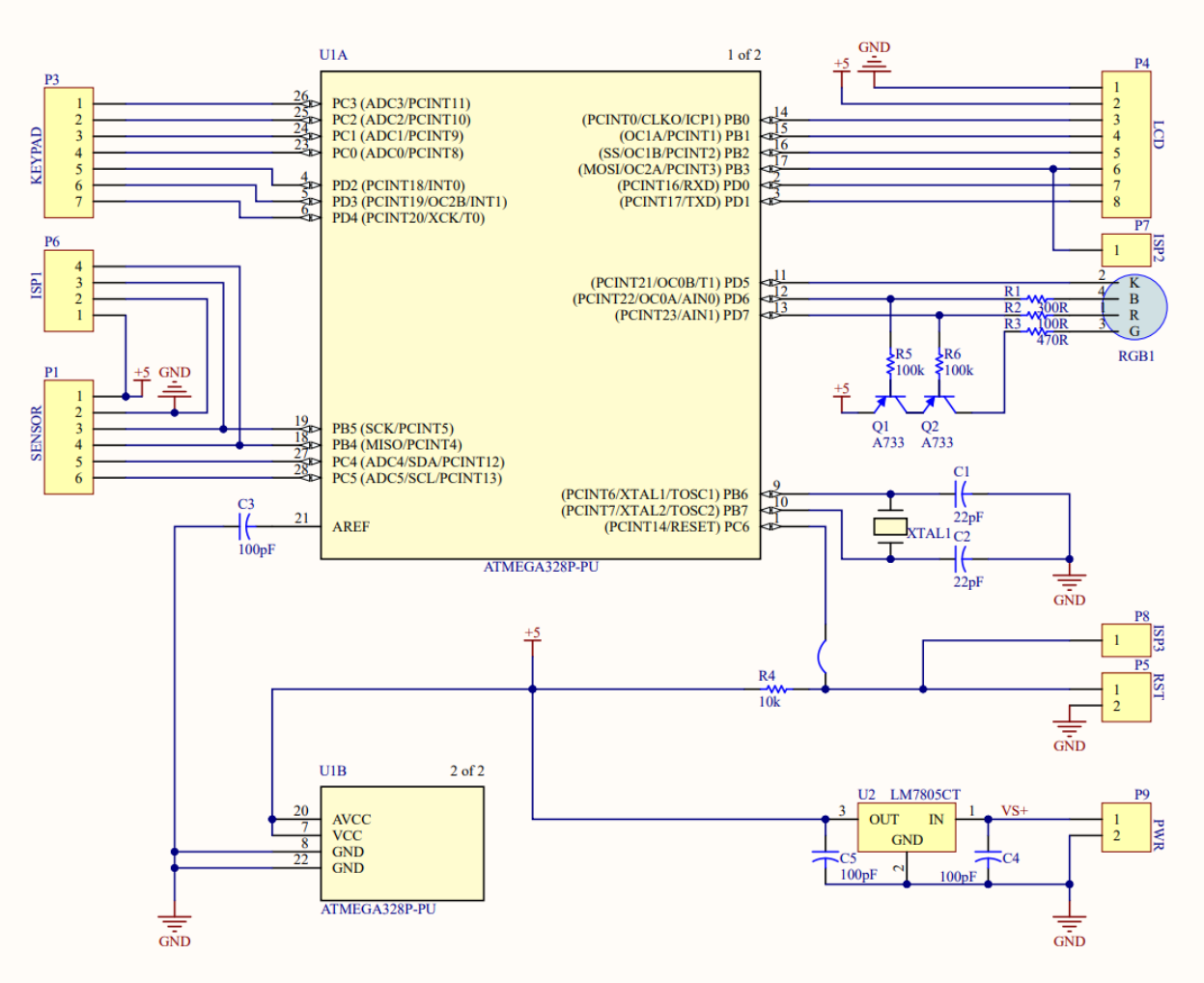
[5] Heble, S., Heble, S., Heble, S., Papuc, I., Heble, S., Heble, S., Heble, S. and Papuc, I., 2021. maxEmbedded » a guide to robotics, embedded electronics and computer vision. [online] maxEmbedded. Available at: https://maxembedded.com [Accessed 9 August 2021].

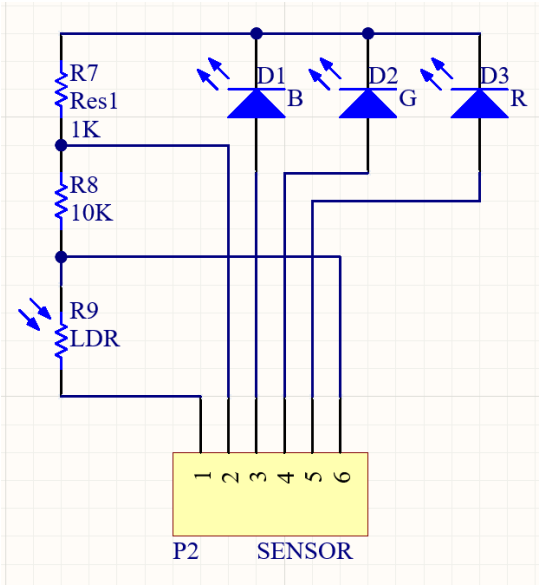
Appendix

1. Appendix I – Flow Chart of the AVR Implementation.



2. Appendix II – Schematic Diagrams. (Main circuit and the Sensor Circuit)





3. Appendix III – AVR C code.

main.c File

```
/*
* main.c
*
* Created: 6/12/2021 11:27:28 PM
* Author : Dulanjana
*/

#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include "cores/lcd.h"
#include "cores/keypad.h"
#include "cores/pwm.h"
#include "cores/calib.h"
#include "cores/sensor.h"

uint16_t reading[3]; //sensor readings
uint8_t rgb[3];      //rgb values
char *rgb_str[3]={"RED = ","GREEN = ","BLUE = "};

//declaring fucnitons for modes
void sensor_mode();
void calib_mode();
void read_mode();
void rgb_mode();


int main(void)
{
    //init cores
    lcd_init();
    keypad_init();
    sensor_init();
    pwm_init();

    //display a startup message
    lcd_clear();
    lcd_string("RGB Color Sensor");
    lcd_set_cursor(1,4);
    lcd_string("Group 2");
    _delay_ms(3000);


    while (1){

        //display the main menu
        lcd_clear();
        lcd_set_cursor(0,0);
```

```
        lcd_string("1 - Color Sensor");
        lcd_set_cursor(1,0);
        lcd_string("2 - Given RGB");

        //get the input key to select options from main menu
        char mainmenu_key=keypad_get_key();

        //key1: goto color sensor mode
        if (mainmenu_key=='1')sensor_mode();

        //key2: goto given value RGB mode
        else if (mainmenu_key=='2')rgb_mode();

        //debounce delay
        _delay_ms(300);
    }
}

void calib_mode(){
    lcd_clear();
    lcd_set_cursor(0,2);
    lcd_string("Calibration");
    lcd_set_cursor(1,4);
    lcd_string("Started");
    _delay_ms(2000);

    //start calibration for colors
    calib_start();

    lcd_clear();
    lcd_set_cursor(0,2);
    lcd_string("Calibration");
    lcd_set_cursor(1,5);
    lcd_string("Done");
    _delay_ms(2000);
}

void read_mode(){

    while (1)
    {
        lcd_clear();
        lcd_string("Place on color");
        lcd_string_blink("...Waiting...",2,1,1);

        //get readings and calculate the color
        sensor_read(reading);
        calib_calc(reading,rgb);

        //display color code
        lcd_clear();
        lcd_set_cursor(0,2);
        lcd_string("@ ");
        lcd_uint8_arr(rgb);
```

```
        lcd_set_cursor(1,0);
        lcd_string("Menu=<-  Read=->");

        //ligt up RGB led with respective color
        pwm_set(rgb);
        pwm_start();

        //ask for continue or stop
        while (1){
            char con_key=keypad_get_key();
            if (con_key==OK_KEY){
                pwm_stop();
                break;
            }
            if(con_key==BACK_KEY){
                pwm_stop();
                _delay_ms(300);
                return;
            }
        }
    }

}

void sensor_mode(){

    lcd_clear();
    lcd_set_cursor(0,2);
    lcd_string("Color Sensor");
    _delay_ms(2000);

    while (1){

        //display a menu for calibrating or reading color values
        lcd_clear();
        lcd_set_cursor(0,0);
        lcd_string("1-Calib   2-Read");
        lcd_set_cursor(1,5);
        lcd_string("Menu=<-");

        char value='\0';
        while(value=='\0'){
            value=keypad_get_key();

            if (value==BACK_KEY){
                _delay_ms(300);
                break;
            }

            //key1: goto calibraiton mode
            else if (value=='1') calib_mode();

            //key2: goto color reading mode
            else if (value=='2') read_mode();
```

```
        }

        //back key: exit from sensor mode
        if (value==BACK_KEY){
            break;
        }
    }
}

void rgb_input(){
    //display a message to enter values
    lcd_clear();
    lcd_set_cursor(0,0);
    lcd_string("Enter values");

    //get RGB values
    for (int color=0;color<3;color++){
        lcd_clear_line(1);
        lcd_string(rgb_str[color]);     //display the color name


        int rgb_value=0;    //given value
        int len_value=0;    //length of the input value

        while(1){
            char key=keypad_get_key();

            //break and go to next color
            if (key==OK_KEY){
                _delay_ms(300);
                break;
            }

            //delete a wrong input character
            else if (key==BACK_KEY){
                if (len_value){
                    lcd_delete();
                    _delay_ms(300);
                    rgb_value/=10;
                    len_value-=1;
                }
            }

            else{
                //display pressed key on lcd display
                lcd_char(key);
                _delay_ms(300);

                //calculate the integer value from pressed keys
                rgb_value=rgb_value*10+(key-'0');
                len_value+=1;
            }
        }
```

```
        //check whether the entered value is valid
        if (rgb_value>255){
            lcd_clear();
            lcd_set_cursor(0,2);
            lcd_string("Invalid input");
            _delay_ms(2000);

            //if the value is invalid allow to enter that value again
            lcd_clear();
            lcd_set_cursor(0,0);
            lcd_string("Enter values");
            color-=1;
        }

        //if inputs are OK, then add them to the rgb array
        else{
            rgb[color]=rgb_value;
        }
    }
}


void rgb_mode(){

    lcd_clear();
    lcd_set_cursor(0,2);
    lcd_string("Given Value");
    lcd_set_cursor(1,6);
    lcd_string("RGB");
    _delay_ms(2000);

    while(1){

        //get rgb color code input
        rgb_input();

        //display the entered color values
        lcd_clear();
        lcd_set_cursor(0,2);
        lcd_string("@ ");
        lcd_uint8_arr(rgb);
        lcd_set_cursor(1,0);
        lcd_string("Menu=<-   RGB=->");

        //ligt up RGB led with the enterd color
        pwm_set(rgb);
        pwm_start();

        //ask for continue with this mode or go back to menu
        char con_key;
        while (1){
            con_key=keypad_get_key();
            if (con_key==OK_KEY || con_key==BACK_KEY){
                break;
```

```
            }
        }

        //exit from given value rgb mode
        if(con_key==BACK_KEY){
            pwm_stop();
            break;
        }

        //debounce delay
        _delay_ms(300);
    }
}
```

Calibration Core

```
/*
* calib.c
*
* Created: 01-Jun-21 6:45:02 PM
*  Author: K.C.S. Abeywickrama
*/
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include "calib.h"
#include "lcd.h"
#include "sensor.h"

uint16_t readings[3][3];
float coffs[3][2];
char *colors[5]={"RED","GREEN","BLUE"};

uint16_t max(uint16_t v1,uint16_t v2){
    if(v1>v2) return v1;
    return v2;
}

uint8_t trim(float v){
    if(v<15) return 0;
    if(v>240) return 255;
    return (uint8_t)v;
}

void calib_start(){

    for(uint8_t i=0;i<3;i++){
        lcd_clear();
        lcd_string("Place on ");
        lcd_string(colors[i]);
```

```
        lcd_string_blink("...Waiting...",5,1,1);
        sensor_read(readings[i]);
        lcd_set_cursor(1,0);
        lcd_uint16_arr(readings[i]);
        _delay_ms(2000);
    }


    for(uint8_t i=0;i<3;i++){
        uint16_t h=readings[i][i];
        uint16_t l=max(readings[(i+1)%3][i],readings[(i+2)%3][i]);

        float m=255.0/(h-l);
        float c=-m*l;

        coffs[i][0]=m;
        coffs[i][1]=c;
    }

}

void calib_calc(uint16_t *reading,uint8_t *rgb){

    for(uint8_t i=0;i<3;i++){
        uint8_t value=trim(coffs[i][0]*reading[i]+coffs[i][1]);
        rgb[i]=value;
    }
}
```

Sensor Core

```
/*
 * sensor.c
 *
 * Created: 19/05/2021 03:18:54
 *  Author: Lakdilu
 */

#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include "sensor.h"
#include "lcd.h"

#define LDR_PIN 5

volatile uint8_t *led_ddrs[3] = {&DDRC, &DDRB, &DDRB};
volatile uint8_t *led_ports[3] = {&PORTC, &PORTB, &PORTB};
uint8_t led_pins[3] = {PORTC4, PORTB4, PORTB5};
```

```
void sensor_init()
{
    for (uint8_t i = 0; i < 3; i++)
        *led_ddrs[i] |= 1 << led_pins[i]; //set led_pins as output

    DDRC &= ~(1 << LDR_PIN); // set LDR_PIN as a input pin

    ADMUX |= (1 << REFS0); //REFS = 1 : External 5V voltage with cpasitor in AREF pin

    ADCSRA |= (1 << ADEN) | (1 << ADPS0) | (1 << ADPS1) | (1 << ADPS2); // Enable ADC,  Freque
ncy is 125kHz

    DIDR0 = (1 << LDR_PIN); // Digital input buffer is disabled
}
uint16_t _adc_read(uint8_t pin)
{
    pin &= 0b111; // convert pin value to binary

    ADMUX |= pin; // Initialize pin which is going to ADC (active the pin which we are going t
op get analog reading)

    ADCSRA |= (1 << ADSC); // begin the ADC

    // delay the function till ADC complete (ADSC = 1 until ADC circule complete)
    while (ADCSRA & (1 << ADSC));

    return (ADC); // return ADC value
}

void sensor_read(uint16_t *reading)
{
    lcd_clear();
    lcd_set_cursor(0,0);
    lcd_string("Reading");
    for (int i = 0; i < 3; i++)
    {
        *led_ports[i] |= 1 << led_pins[i];    // Turn on LED
        lcd_char('.');
        _delay_ms(2000);                      // Wait until LDR reading is stable
        reading[i] = _adc_read(LDR_PIN);      // Get LDR reading (corresponding function in se
nsor.c)
        *led_ports[i] &= ~(1 << led_pins[i]); // Turn off LED
    }                                         // LED pins ---
> RED = C4 / GREEN = B4 / BLUE = B2

}
```

PWM Core

```
/*
* pwm.c
*
* Created: 15-Jun-21 6:37:17 PM
*  Author: K. C. S. Abeywickrama
*/

#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "pwm.h"

#define PWM_PIN DDD5
#define PWM_DDR DDRD
#define PWM_PORT PORTD
#define PWM_OCR OCR0B

#define DEMUX_PORT PORTD
#define DEMUX_DDR DDRD

volatile uint8_t pin=0;
volatile uint8_t gbr[3]={0,0,50};

ISR(TIMER0_OVF_vect){

    DEMUX_PORT = (DEMUX_PORT & 0b00111111) | pin<<6 ;
    pin=(pin+1)%3;
    PWM_OCR=gbr[pin];

}

void pwm_init(){
    //define output pins
    PWM_DDR |= 1<< PWM_PIN;
    DEMUX_DDR |= 1<< DDD6 | 1<< DDD7;
    PWM_PORT|=1<<PWM_PIN;
}

void pwm_start(){
    //timer0 B inverting mode fast PWM | TOV on 0xff
    TCCR0A |= 1<<COM0B1 | 1<<COM0B0 | 1<< WGM01 | 1<< WGM00;

    //enable overflow interrupt
    TIMSK0 |= 1<<TOIE0;
    sei();

    //start timer2
```

```
    TCCR0B |= 1<<CS00; //(no prescaler)
    TCCR0B |= 1<<CS01 | 1<<CS00; //(64 prescaler)
    //TCCR0B |= 1<<CS02; //(256 prescaler)
    //TCCR0B |= 1<<CS02 | 1<<CS00; //(1024 prescaler)
}

void pwm_stop(){
    cli();

    TCCR0B=0;
    TIMSK0 =0;
    TCCR0A =0;

    PWM_PORT|=1<<PWM_PIN;

}

void pwm_set(uint8_t *rgb){
    gbr[0]=rgb[1];
    gbr[1]=rgb[2];
    gbr[2]=rgb[0];
}

void pwm_set_args(uint8_t red,uint8_t green,uint8_t blue){
    gbr[0]=green;
    gbr[1]=blue;
    gbr[2]=red;
}

void pwm_check(){
    uint8_t _colors[7][3]={{255,0,0},{0,255,0},{0,0,255},
    {255,255,0},{0,255,255},{255,0,255},{255,165,0}};

    pwm_start();

    for(uint8_t i=0;i<7;i++){
        pwm_set(_colors[i]);
        _delay_ms(3000);
    }

    pwm_stop();
}
```

LCD Core

```
/*
* lcd.c
*
* Created: 5/19/2021 12:49:00 AM
*  Author: Yohan Abeysinghe 1
*/

#define F_CPU 16000000UL


#include <avr/io.h>
#include <util/delay.h>
#include "lcd.h"

#define DATA_PORT0 PORTB
#define DATA_PORT1 PORTD
#define DATA_DDR0 DDRB
#define DATA_DDR1 DDRD
#define EN_DDR DDRB
#define LCD_EN PORTB
#define LCD_RS PORTB
#define RS PORTB0               /* Define Register Select pin */
#define EN PORTB1               /* Define Enable signal pin */

void lcd_command( unsigned char cmnd )
{
    DATA_PORT0 = (DATA_PORT0 & 0b11110011) | (cmnd & 0b110000)>>2;
    DATA_PORT1 = (DATA_PORT1 & 0b11111100) | (cmnd & 0b11000000)>>6;
    LCD_RS &= ~ (1<<RS);        /* RS=0*/
    LCD_EN |= (1<<EN);
    _delay_us(1);
    LCD_EN &= ~ (1<<EN);

    _delay_us(200);

    cmnd<<=4;
    DATA_PORT0 = (DATA_PORT0 & 0b11110011) | (cmnd & 0b110000)>>2;
    DATA_PORT1 = (DATA_PORT1 & 0b11111100)| (cmnd & 0b11000000)>>6;
    LCD_EN |= (1<<EN);
    _delay_us(1);
    LCD_EN &= ~ (1<<EN);
    _delay_ms(2);
}


void lcd_char( unsigned char data )
{
    DATA_PORT0 = (DATA_PORT0 & 0b11110011) | (data & 0b110000)>>2;
    DATA_PORT1 = (DATA_PORT1 & 0b11111100)| (data & 0b11000000)>>6;
    LCD_RS |= (1<<RS);      /* RS=1 */
    LCD_EN|= (1<<EN);
    _delay_us(1);
```

```
    LCD_EN &= ~ (1<<EN);

    _delay_us(200);

    data<<=4;
    DATA_PORT0 = (DATA_PORT0 & 0b11110011) | (data & 0b110000)>>2;
    DATA_PORT1 = (DATA_PORT1 & 0b11111100)| (data & 0b11000000)>>6;
    LCD_EN |= (1<<EN);
    _delay_us(1);
    LCD_EN &= ~ (1<<EN);
    _delay_ms(2);
}

void lcd_init (void)            /* LCD Initialize function */
{
    DATA_DDR0 |= 0b1111;
    DATA_DDR1 |= 0b0011;
    EN_DDR |= 1<<EN;
    _delay_ms(20);          /* Delay to power on the LCD */

    lcd_command(0x02);              /* send for 4 bit initialization  */
    lcd_command(0x28);              /* 2 line, 5*7 matrix in 4-bit mode */
    lcd_command(0x0c);              /* Display on cursor off*/
    lcd_command(0x06);              /* Increment cursor (shift cursor to right)*/
    lcd_command(0x01);              /* Clear display screen*/
    _delay_ms(2);
}


void lcd_string (char *str)     /* Send string to LCD function */
{
    int i;
    for(i=0;str[i]!=0;i++)      /* Send each char */
    {
        lcd_char (str[i]);
    }
}

void lcd_clear()
{
    lcd_command (0x01);     /* Clear display */
    _delay_ms(2);
    lcd_command (0x80);     /* Cursor at home position */
}

void lcd_set_cursor(int row_index,int col_index){
    if (row_index==0){
        lcd_command(0x80);
    }
    if (row_index==1){
        lcd_command(0xC0);
    }
    for (int temp=0;temp<col_index;temp++){
        lcd_command(0x14);
```

```
    }
}

void lcd_clear_line(int row){
    lcd_set_cursor(row,0);
    for (int position=0;position<16;position++){
        lcd_char(' ');
    }
    lcd_set_cursor(row,0);
}

void lcd_int(int value){
    int length;
    int divider;
    if (value>=100){
        length=3;
        divider=100;
    }
    else if(value>=10){
        length=2;
        divider=10;
    }
    else if(value>=0){
        length=1;
        divider=1;
    }
    for (int bit=0;bit<length;bit++){
        int quocient=value/divider;
        lcd_char(quocient+'0');
        value%=(quocient*divider);
        divider/=10;
    }
}

void lcd_float(float value){
    int a=value;
    int b=(value-a)*1000;
    lcd_int(a);
    lcd_char('.');
    lcd_int(b);
}

void lcd_delete(){
    lcd_command(0x10);
    lcd_char(' ');
    lcd_command(0x10);
}

void lcd_string_blink(char *word,int iter,int row,int column){
    lcd_clear_line(row);
    for (int blink_iter=0;blink_iter<iter;blink_iter++){
        lcd_set_cursor(row,column);
        lcd_string(word);
        _delay_ms(500);
```

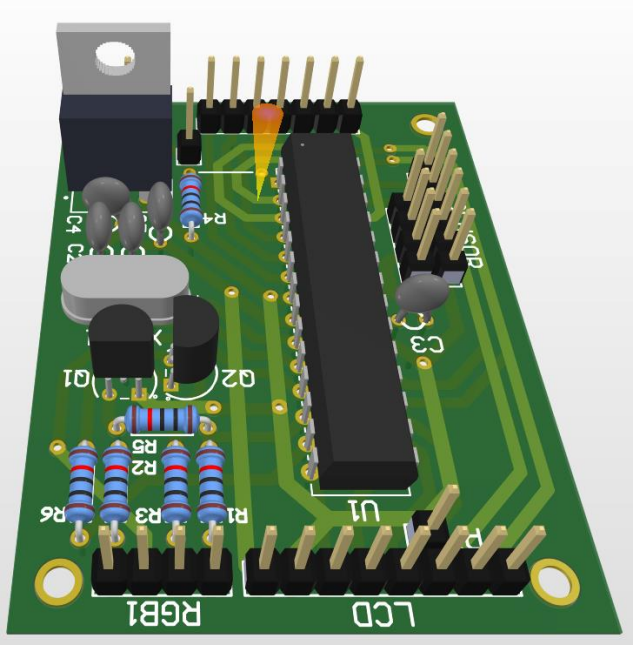
```
        lcd_clear_line(row);
        _delay_ms(500);
    }
}

void lcd_uint8_arr(uint8_t *values){
    for (int color=0;color<2;color++){
        lcd_int(values[color]);
        lcd_char(',');
    }
    lcd_int(values[2]);
}

void lcd_uint16_arr(uint16_t *values){
    for (int color=0;color<2;color++){
        lcd_int(values[color]);
        lcd_char(',');
    }
    lcd_int(values[2]);
}

void lcd_check(){
    lcd_clear();
    lcd_set_cursor(0,5);
    lcd_string("LCD is");
    lcd_set_cursor(1,7);
    lcd_string("OK");
    _delay_ms(2000);
}
```

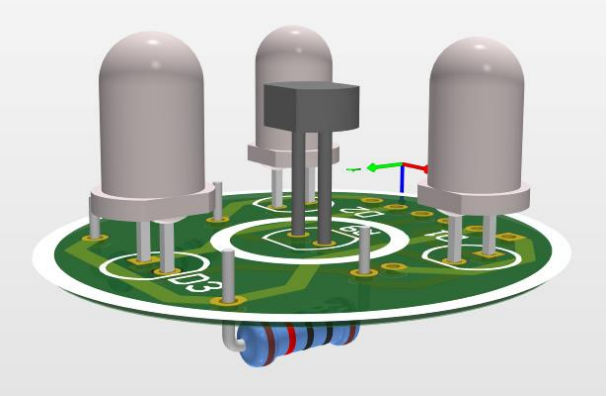
Keypad Core

```
/*
 * keypad.c
 *
 * Created: 5/19/2021 12:22:11 AM
 *  Author: Dulanjana
 */

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include "keypad.h"
#include "lcd.h"

#define ROW_DDR DDRC
#define ROW_PORT PORTC
#define ROW_PIN PINC
#define COL_DDR DDRD
#define COL_PORT PORTD
#define COL_PIN PIND



const uint8_t ROW_PINS[4]={PORTC0,PORTC1,PORTC2,PORTC3};
const uint8_t COL_PINS[4]={PORTD2,PORTD3,PORTD4};

const char KEYS[4][3]={{'1','2','3'},{'4','5','6'},{'7','8','9'},{BACK_KEY,'0',OK_KEY}};

void keypad_init(){
   COL_DDR &= ~(0x1c);
   COL_PORT |= 0x1c;
   ROW_DDR |= 0x0f;
}

void set_port(char *str){
   for (int bit=0;bit<4;bit++){
       if (str[bit]=='1'){
           ROW_PORT |= 1<<ROW_PINS[bit];
       }
       else{
           ROW_PORT &= ~(1<<ROW_PINS[bit]);
       }
   }
}

char keypad_get_key(){
   char value='\0';
   while (value=='\0'){
       for (int col=0;col<3;col++){
           set_port("0111");
           if(~COL_PIN & 1<<COL_PINS[col]){
               value=KEYS[0][col];
           }
```

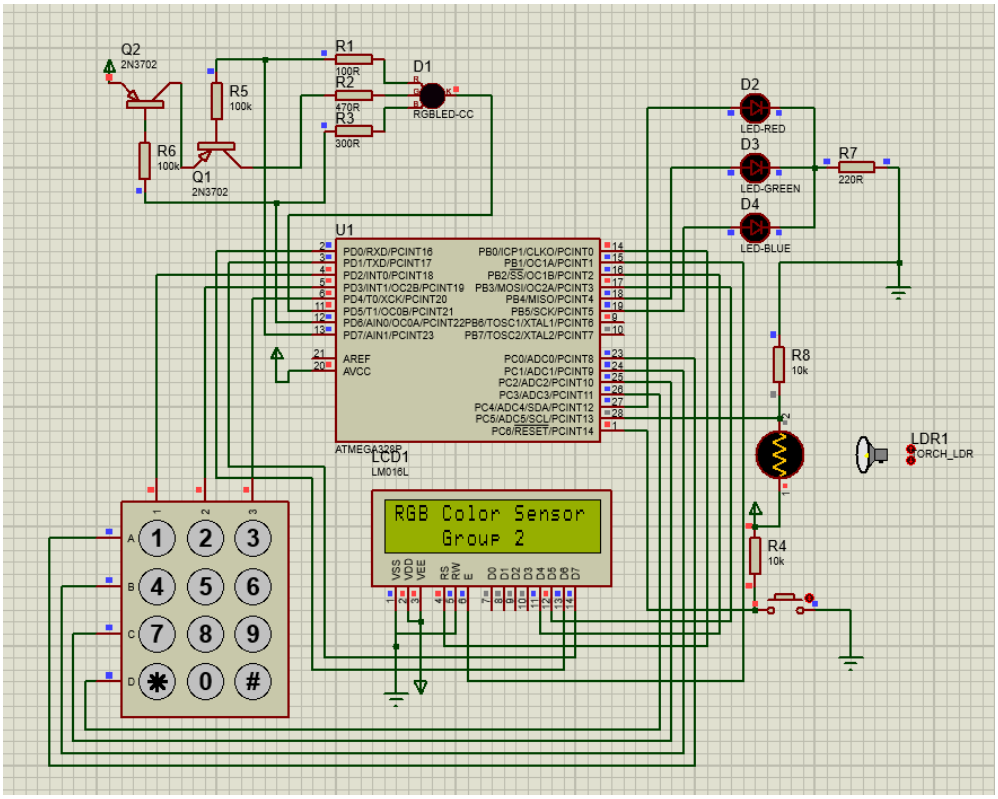
```
            set_port("1011");
            if(~COL_PIN & 1<<COL_PINS[col]){
                value=KEYS[1][col];
            }
            set_port("1101");
            if(~COL_PIN & 1<<COL_PINS[col]){
                value=KEYS[2][col];
            }
            set_port("1110");
            if(~COL_PIN & 1<<COL_PINS[col]){
                value=KEYS[3][col];
            }
        }
    }
    return value;
}

void keypad_check(){
    lcd_clear();
    lcd_set_cursor(0,0);
    lcd_string("Keypad checking");
    lcd_set_cursor(1,0);
    char value;
    while (1){
        value=keypad_get_key();

        lcd_char(value);
        _delay_ms(300);

        if (value==OK_KEY){
            _delay_ms(1000);
            break;
        }
    }
}
```

4. Appendix IV – 3D Renderings of the PCBs

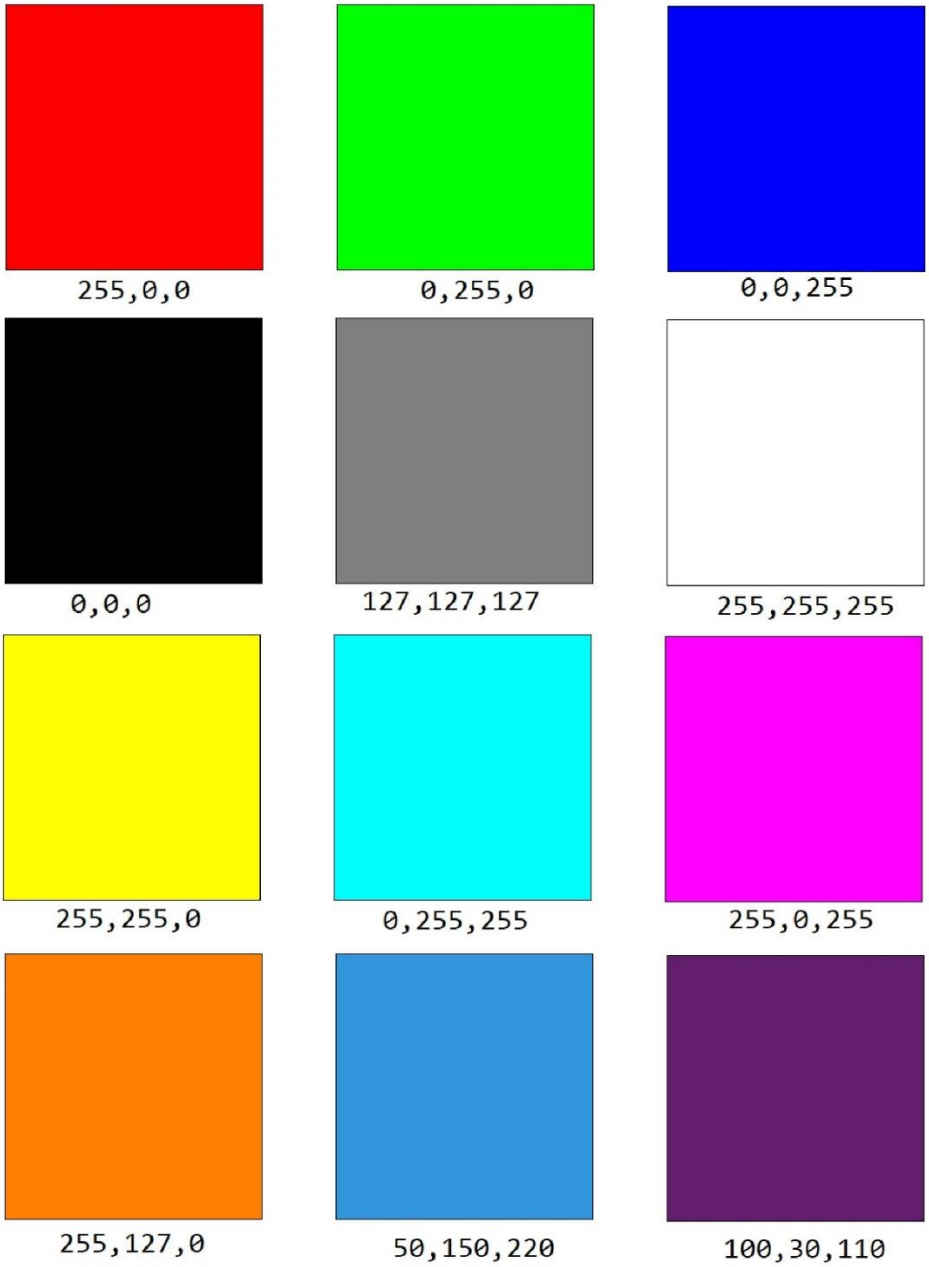


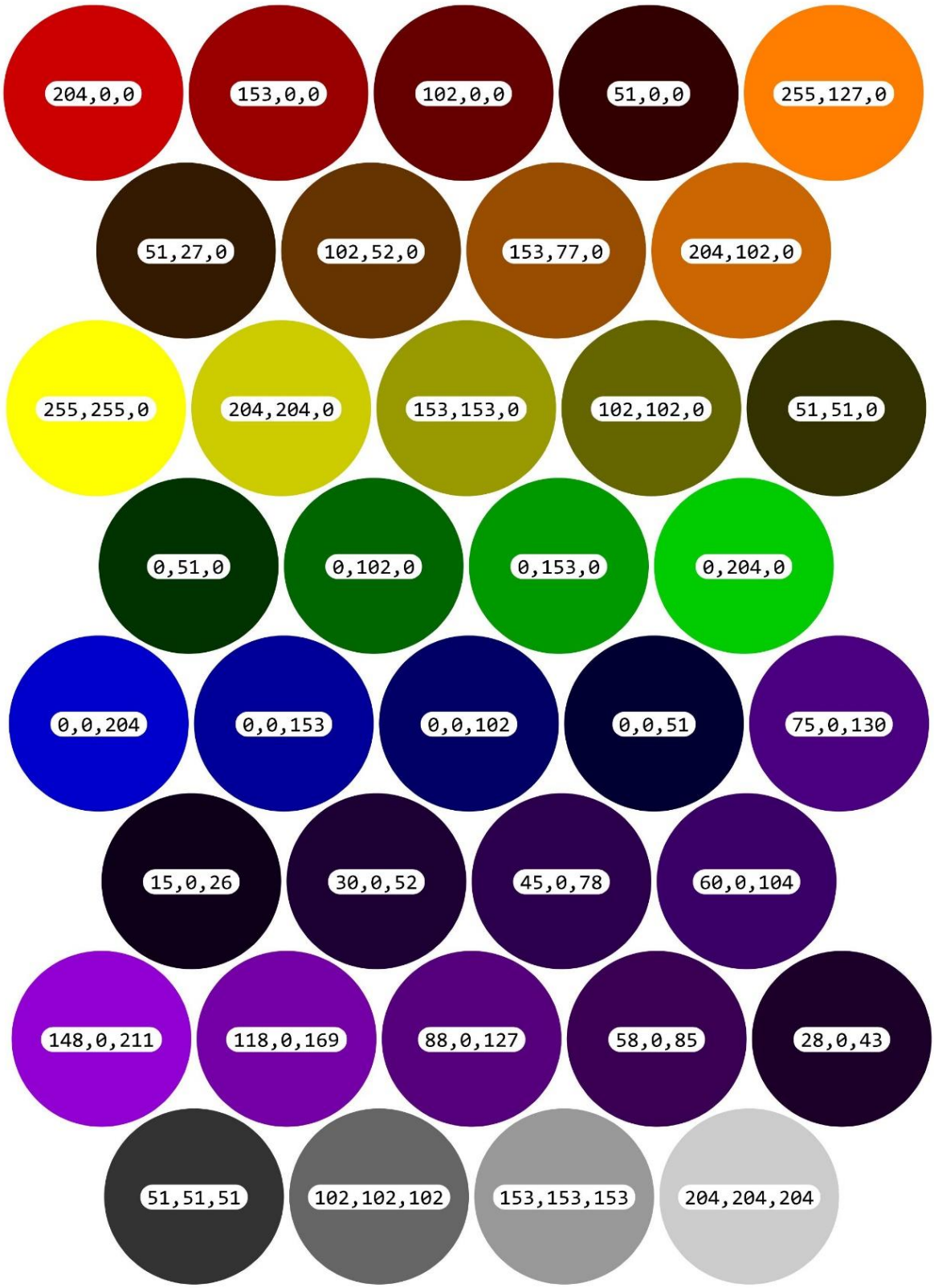


5. Appendix V – Proteus Simulation



6. Appendix IV – Color Codes Used.




204,0,0
153,0,0
102,0,0
51,0,0
255,127,0
51,27,0
102,52,0
153,77,0
204,102,0
255,255,0
204,204,0
153,153,0
102,102,0
51,51,0
0,51,0
0,102,0
0,153,0
0,204,0
0,0,204
0,0,153
0,0,102
0,0,51
75,0,130
15,0,26
30,0,52
45,0,78
60,0,104
148,0,211
118,0,169
88,0,127
58,0,85
28,0,43
51,51,51
102,102,102
153,153,153
204,204,204