Automatic Color Mixer

A method for automated color recognition and replication

MAX LINDGREN

MAX THIEL
Automated Color Mixer

A method for automated color recognition and replication

MAX LINDGREN
MAX THIEL

Bachelor’s Thesis at ITM
Supervisor: Pouya Mahdavipour
Examiner: Nihad Subasic

TRITA MMK 2017:32 MDAB 650
Abstract

Throughout history colors have been an important way for living organisms to signal something to their surroundings. For instance, the color of a plant’s leaves or petals have been the way with which the plant communicates with its surroundings for example to attract insects to spread its DNA, or to signal that it is poisonous. Animals have also been using colors to impress the other gender or to frighten other animals in order to stay alive and spread their DNA. One could argue that colors in some cases are vital and have played a major role in evolution. This may not be the case today for us humans, but the importance and use of color is still very significant in todays society.

The purpose of this project is to construct a demonstrator that can mix a color that it gets as input from a user. Using automation control the machine can first mix a color and then compensate for any errors just like a human would do to get the right result. To be able to do this a color sensor that reads color in RGB is used. The different levels are then translated to a another color model called HSV, which is then used for mixing the final color.

The performance of the system was tested by letting a sensor scan a sample color card, calculate how much of each primary color to pour into the mix. The mixed paint was then painted onto a piece of paper that was left to dry before being scanned. The difference in hue was noted after the first mix and the system continued adjusting for any deviations until a good result was achieved. The result of the tests was that it took a maximum amount of three mixes to get within 0.8% of the reference value. In half of the tests the demonstrator achieved a good result after the very first mix.
Sammanfattning

Automatisk färgblandare

Färg har genom historien varit ett effektivt sätt att signalera något till sin omgivning. Till exempel har färger på blad hos blommor och växter fungerat som växters sätt att kommunisera med sin omgivning. Antingen för att locka till sig insekter för att sprida sitt DNA eller för att signalera giftighet. Djur har använt sig av färger för att imponera på det andra könet och därigenom kunna föra sitt DNA vidare. Man skulle kunna argumentera för att färg har haft en viktig roll i evolutionen. För människan är det i dagsläget inte riktigt lika viktigt evolutionärt, men vikten av och användningen av färg har en stor betydelse i dagens samhälle. Numera finns det färgaffärer som tillhandahåller maskiner som kan låta kunden själv välja precis vilken färg den vill ha och med hjälp av maskinerna blanda upp denna.

Syftet med detta projekt är att konstruera en mekatronisk konstruktion som kan ta indata i form av färgval från en användare. Genom användandet av reglerteknik låter maskinen först blanda färgen en första gång och sedan kompensera för eventuella fel precis som en människa skulle. För att uppnå detta används en färgsensor som löser av färgen som RGB och sedan översätter detta till färgermodellen HSV som används för att bestämma blandingens volympropor tioner.

För att testa systemet läts en sensor läsa av ett färgkort och beräkna hur mycket av vardera primärfärg att hålla i blandningen. Den blandade färgen applicerades senare på ett papper och låts torka innan avläsning. Skillnaden i nyans efter första beredningen noterades och systemet fortsatte att korrigerar eventuella fel tills ett bra resultat erhölls. Resultatet av alla tester var att det tog högst tre blandningar för att komma inom 0,8% från referensnyansen. I hälften av testerna erhölls bra resultat redan efter den initierade blandningen.
Acknowledgements

We would like to thank Pouya Mahdavipour for supervision, guidance and feedback throughout this project. We would also like to thank Staffan Qvarnström and all other course assistants for help with components, machines and more. And last but not least, all other students in the lab for all the moral support and feedback throughout this semester.

Max Lindgren, Max Thiel
Royal Institute of Technology, Stockholm, May 2017
Contents

1 Introduction .............................................. 1
  1.1 Background ............................................ 1
  1.2 Purpose ............................................... 2
  1.3 Scope .................................................. 2
  1.4 Method ................................................ 3

2 Theory .................................................. 5
  2.1 Mixing colors ........................................... 5
    2.1.1 RGB ................................................. 6
    2.1.2 CMYK ............................................... 7
    2.1.3 HSV ................................................ 8
  2.2 Medium ............................................... 9
  2.3 Structural design ..................................... 10
  2.4 Controller .......................................... 11

3 Demonstrator ......................................... 13
  3.1 Assumptions and considerations ....................... 13
  3.2 Software ............................................. 14
  3.3 Electronics ......................................... 14
    3.3.1 Arduino UNO ....................................... 14
    3.3.2 Grove - I2C Color Sensor .......................... 15
    3.3.3 Valve electronics .................................. 15
  3.4 Mechanics .......................................... 16
    3.4.1 Valves ............................................ 16
    3.4.2 The Pentagon ...................................... 17

4 Results and Conclusions .............................. 19
  4.1 Results ............................................... 19
  4.2 Discussion .......................................... 21
    4.2.1 Input Issues ...................................... 21
    4.2.2 Possible Error Sources ............................ 21
    4.2.3 System Performance ............................... 22
  4.3 Conclusions .......................................... 23
5 Future Work and Improvements 25
  5.1 Immediate improvements .............................. 25
  5.2 Future Visions ........................................ 25

Bibliography 27

Appendices 27

A Arduino code 29

B Python code 43
Nomenclature

Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD</td>
<td>Computer-aided design</td>
</tr>
<tr>
<td>CMYK</td>
<td>Cyan, magenta, yellow, key</td>
</tr>
<tr>
<td>HSV</td>
<td>Hue, saturation, value</td>
</tr>
<tr>
<td>MOSFET</td>
<td>Metal–oxide–semiconductor field-effect transistor</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, green, blue</td>
</tr>
</tbody>
</table>

Symbols

≪ Left shift bit operator
List of Figures

2.1 Graphical representation of the RGB color model, created using Adobe Photoshop. ................................................................. 6
2.2 Graphical representation of the CMYK color model, created using Adobe Photoshop. ................................................................. 7
2.3 Graphical representation of the HSV color model, created using Autodesk Maya and Adobe Photoshop. ................................. 8
2.4 CAD model of the structural design, created using Solid Edge ST8. .... 10
3.1 Flow chart describing the main principle of the software, created using Adobe Illustrator. ......................................................... 14
3.2 Power switch circuit .................................................................. 15
3.3 Picture of one solenoid platform ................................................. 16
3.4 Setup of the final construction ................................................... 17
4.1 Color circle consisting of the color samples used as reference. ...... 19
4.2 Graph describing the error margin of the first mix for a series of samples, created using Microsoft Excel 2011. ....................... 20
4.3 Reference samples at the bottom of every pile. Closest to the reference is the final result. On top of every pile is the first mix. ...... 20
4.4 A chart showing the fraction of blue paint in the initial mix, dependent on the input hue, created using Microsoft Excel 2016. .... 22
Chapter 1

Introduction

This chapter aims to discuss the thoughts and ideas that serve as the basis of this mechatronics project as well as the purpose and overall goals of the project and the method used to achieve them.

1.1 Background

The will to create different colors is as old as time. The importance of color has throughout history been significant on many different levels, mostly it has served as a way for living organism to signal something to its surroundings. For instance, the color of a plants leaves or petals have been the way with which the plant communicates with its surroundings to attract insects and to spread its DNA, or to signal that it is poisonous. This strategy has also been useful in the same way within the animal kingdom. The color of an animal can serve as a way to impress the other gender, frighten other animals or be used as camouflage to avoid predators. In other words colors have served a major role in evolution throughout history. This may not have been the case for the human race in terms of biology, although colors have still played a vital role culturally and economically. The color and design of ones clothes has been, and still to some degree is, a statement and measure of ones wealth or standing in society. This may not be as apparent in todays society but color serves a big role as a way to express personalty and taste. This is not just tied to fashion and clothes, but to all kinds of visual expressions, for example art and furnishing.

Paint stores today often use advanced machines to color match paint. This however, is still a quite involved process requiring several advanced and expensive machines such as spectrophotometers[1], automatic dispensers and mixers[2]. The following are the research questions looked at in this project.
1.2 Purpose

This report aims to find a color model that works well when automating the mixing of colors and from that builds a method to do it in practice. It further aims to practically assess the performance of the chosen method. The concept of mixing colors has long seemed like a very practical problem, especially when talking about acrylic paint or other types of paint primarily used in small quantities for applications like art. The trend to miniaturize previously industrial machines for desktop use was an inspiration to see if something similar could be done for automatic color mixing. Applications, like the ones mentioned earlier, often use other primary colors than for example a printer. Therefore a more general model is preferable.

- What is the state of the art in terms of methodologies for color replication automation using different sets of primary colors?
- Which of the methodologies are cost-effective in practice?
- How can accuracy be defined and measured in the context of the chosen methodology?
- Given that a cost-effective methodology has been chosen, what is its accuracy in automatic color replication?

1.3 Scope

The demonstrator that will be constructed in this project is to be able to mix any color possible given the paint used. This means that most of the visible spectrum will be covered. However, due to the time constraints of this project the demonstrator is not intended to be a fully automated product. For example the sensor measurements are to be taken manually as well as the blending of the colors mixed by the demonstrator. This is to make sure enough focus is put into the construction of the method necessary to evaluate the performance of the demonstrator and the system as a whole.
1.4  Method

A demonstrator will be constructed with containers for each individual primary color. A sensor connected to an Arduino will read a color and calculate how much color to pour into the mix. To evaluate the performance of the system the RGB values from the color sensor are converted into three integers to express the color in the HSV color model. The demonstrator mixes the color, scans the mix and compensates for any deviations from the reference HSV data. If the data collected from the mixed color is close enough to the reference the process stops and the demonstrator is ready to read another color. In order to evaluate whether the demonstrator performs well enough the following requirements are set:

- The absolute difference from the reference in each HSV integer has to be within 5%.
- The number of adjustments has to be less than five.
Chapter 2

Theory

This chapter serves to talk about the fundamental theories needed to create a functioning demonstrator at the end of this project.

2.1 Mixing colors

There are many different models when it comes to mixing colors. Since the idea of the project is centered around mixing a given color from an arbitrary set of primary colors and also because the aim at desktop application implies the use of cheap and commonly available sensors and components, three different models were chosen and evaluated throughout the project.
2.1.1 RGB

RGB stands for Red, Green and Blue and is an additive color model. By adding different amounts of each individual color it is possible to cover a broad spectrum. A basic visual explanation of the principles is shown in figure 2.1. This is widely used in display technology like TVs, computer screens and mobile phones but also in devices aimed at capturing media like scanners and digital cameras[3]. Like the last mentioned devices the color sensor used in this project gives its result in RGB. However, RGB being additive means it applies to adding in wavelengths of light, mixing paint or pigments works on another principle. When mixing paint of two different colors what is changed is which wavelengths are absorbed. The wavelengths that are not absorbed i.e. what is reflected, is the color perceived.

Figure 2.1. Graphical representation of the RGB color model, created using Adobe Photoshop.
2.1. MIXING COLORS

2.1.2 CMYK

CMYK stands for Cyan, Magenta, Yellow and Key (Black) and is a subtractive color model. This model is mainly used in printing and assumes that white is used as a base to which the color is then applied. By adding in different pigments more wavelengths are absorbed[3]. Figure 2.2 illustrates how CMYK works. This color model would seem to work relatively well for this project but because of the choice of different primary colors the third method looked at is a better fit.

Figure 2.2. Graphical representation of the CMYK color model, created using Adobe Photoshop.
2.1.3 HSV

Neither the RGB or CMYK color models seem ideal for this project. Therefore another model was investigated. HSV stands for Hue, Saturation and Value, it is sometimes also referred to as HSB where the B stands for brightness instead of value. The model is a remapping of the RGB color model into cylindrical coordinates and was created to mimic how an artist mixes color[4]. This maps the hue as an angle on a color circle. The saturation becomes the distance from the center and the value becomes the height along the cylinder, this is visually represented in figure 2.3. This works very well for this project since it makes it easy to break the mixing process into different steps. Hue could first be matched by mixing of the primary colors, then saturation and value could be matched by adding in white and black respectively.

The conversion used in this project is an integer-based conversion designed by Vladimir Chernov, Jarmo Alander and Vladimir Bochko[5]. This model was chosen because of its optimization and loss less conversion. It also, because of keeping all values as integers, makes it easier to transfer over the Arduinos serial communication. This implementation takes 24-bit RGB as input (8 bits per channel). It outputs value as an integer from 0 to 255, saturation as 0 or in the span 257 to 65535 and hue as 0 to 393222. The conversion is done by the following nine steps.

1. Find the maximum \(M\), minimum \(m\) and middle \(c\) of Red \((R)\), Green \((G)\) and Blue \((B)\).

   \[
   M = \max(R, G, B) \tag{2.1}
   \]

   \[
   m = \min(R, G, B) \tag{2.2}
   \]

   \[
   c = \text{mid}(R, G, B) \tag{2.3}
   \]
2.2. MEDIUM

2. Assign Value \((V)\) with \(M\)
\[
V = M
\]  
(2.4)

3. Calculate the difference \((d)\) between \(M\) and \(m\).
\[
d = M - m
\]  
(2.5)

4. If \(d\) is equal to 0 then assign \(s\) with 0 and return. \(H\) is undefined in this case, we choose in our implementation to give it a default value of 0.

5. Determine sector index \((I)\).
\[
I = \begin{cases} 
0, & \text{if } M = R \text{ and } m = B \\
1, & \text{if } M = G \text{ and } m = B \\
2, & \text{if } M = G \text{ and } m = R \\
3, & \text{if } M = B \text{ and } m = R \\
4, & \text{if } M = B \text{ and } m = G \\
5, & \text{if } M = R \text{ and } m = G 
\end{cases}
\]  
(2.6)

6. Calculate saturation \((S)\).
\[
S = \frac{(d \ll 16) - 1}{V}
\]  
(2.7)

7. Calculate the fractional part of hue \((F)\).
\[
F = \frac{(c - m) \ll 16}{d} + 1
\]  
(2.8)

8. Invert \(F\) for odd sector indexes, this is done by subtracting \(F\) from the edge length constant \((E)\) that has a value of 65537.
\[
\text{if } I = 1 \text{ or } I = 3 \text{ or } I = 5 \text{ then } F = E - F
\]  
(2.9)

9. Calculate the hue \((H)\).
\[
H = E \cdot I + F
\]  
(2.10)

2.2 Medium

A number of different mediums were considered throughout the project. For example; food coloring, ink, oil based paint, and acrylic paint. Due to economic restrictions and tight time frame a simple and cheap solution was requested. The solution found was to use acrylic paint and thin it with water. This solution is both cheap and easy to use. The color dissolves quickly which makes it ideal for initial testing.
CHAPTER 2. THEORY

2.3 Structural design

The medium used in this project is acrylic hobby paint mixed with water. This is done to give the paint a lower viscosity in order to allow it to flow through low diameter silicone tubing. The idea of the construction is to have containers for each of the different colors used. At the bottom of each container a tube is attached which allows for the fluid to flow through. The containers are placed at the top of the construction to enable the paint to flow by gravity alone. To enable flow from the tube a solenoid is connected to a spring loaded arm in order to create a valve. The valve is then controlled from an Arduino. The tubes are led down to another container where the colors are mixed. A sample slide can then be prepared and measured by the color sensor. The sample measurement is compared to the reference color and adjustments to the mix are made. This process is then repeated until the result is within the error margin.

![Figure 2.4. CAD model of the structural design, created using Solid Edge ST8.](image-url)
2.4 Controller

The feedback controller that is used in the project follows a very simple principle. The reference signal is the color measured by the color sensor. This signal contains information about the different levels of RGB in the color measured. This is then converted to HSV. Depending these integers the base of the mixture is decided.

- If the value is low, the base of the mixture will be black.
- If the value is high and the saturation is low, the base of the mixture will be white.
- If the value is high and the saturation is high, the base will be the hue and will first be mixed from the primary colors.

An initial mix is poured and then measured, to compensate for any error the machine will add more of the color needed to bring it closer to the reference. The amount added depends on the difference in hue, saturation and value respectively. The intended controller is thereby a proportional controller.
Chapter 3

Demonstrator

This chapter aims to present the ingoing components necessary to construct a functioning color mixing machine.

3.1 Assumptions and considerations

The following points have been taken into consideration when constructing the demonstrator:

- The pouring of paint needs to be controlled individually per color.
- The economic restrictions on the project meant that a cheaper alternative to buying solenoid valves had to be found.
- The tubes used to transport the liquid paint need to have a suitable inner diameter and be elastic enough to squeeze shut in order to stop the flow.
- The placement of the paint containers have to be high enough to allow for an even flow.
3.2 Software

The heart and brains of the demonstrator is the Arduino UNO. The Arduino controls the pouring of the different colors into the mix. In figure 3.1 the flowchart of the Arduino based software is presented:

![Flow chart](image)

**Figure 3.1.** Flow chart describing the main principle of the software, created using Adobe Illustrator.

Another computer based program that captures the serial communication from the Arduino was written. It was used to help with debugging and to visually present the color captured by the sensor. This software is however not required for operation of the machine.

3.3 Electronics

This section aims to introduce and explain the different electronic components used in the construction of the demonstrator

3.3.1 Arduino UNO

Arduino UNO is a programmable microcontroller that can be used for all sorts of projects. It has several digital output and input pins as well as analog inputs[6].
3.3. ELECTRONICS

3.3.2 Grove - I2C Color Sensor

The Grove - I2C Color Sensor used in this project is an optical color sensor aimed at hobbyists and makers[7]. The central part of this product is the TCS3414 digital color sensor[8].

3.3.3 Valve electronics

Except from the Arduino and the sensor the system uses five solenoids controlled by five MOSFETs. These contain an internal flyback diode but since the change in current in the large coils of the solenoids when turned off can result in large voltage spikes another fly back diode was added to each circuit. The solenoids are powered by a different power source than the Arduino since they require both a much higher voltage and current than can be provided through the Arduino itself.

Figure 3.2. Power switch circuit
3.4 Mechanics

This section aims to talk about the construction of the ingoing mechanical components of the demonstrator.

3.4.1 Valves

The valves are made from wood and have been made by hand. A hole was drilled through the bottom part which the tube could fit through. The arm was constructed out of steel to assure it could take the strain caused by the spring and still be thin enough to squeezes the tube shut. The solenoid is locked in place by two M2 screws whose position is adjustable on the side board in the bottom of figure 3.3 below to allow for a smooth motion. The position of the spring that pulls the arm that squeezes the tube had to be adjusted. This to prevent any fluid passing while making sure the solenoid was still able to pull the arm allowing the paint to pass through.

![Figure 3.3. Picture of one solenoid platform](image-url)
3.4. MECHANICS

3.4.2 The Pentagon

Due to the initial design’s need for five valves and stability in the construction of the demonstrator a symmetrical solution felt like the simplest way to achieve a good result. First of all the components were created in CAD using extruded parts with the same thickness as the fiberboard. Using mortise and tenon joints the entire construction could be assembled and disassembled with ease. The final construction can be seen below in figure 3.4.

![Setup of the final construction](image)

**Figure 3.4.** Setup of the final construction
Chapter 4

Results and Conclusions

This chapter aims to talk about the results of this project and performance of the demonstrator. The results are also discussed in relation to the research questions posed for this project.

4.1 Results

To test performance of the system and demonstrator a series of sample cards were made to serve as reference signal for the system. The tests that where carried out followed the methodology described below. A sample card, as seen in figure 4.1, was scanned and the demonstrator calculated how much of each individual color to pour into then mix.

![Figure 4.1](image)

*Figure 4.1.* Color circle consisting of the color samples used as reference.
The result was then painted onto a piece of paper that was left to dry before being scanned. The difference in hue was noted after the first mix and the system continued working until a good result was achieved. The number of adjustments was also noted. The results are visible below in figure 4.2.

![First mix graph](image)

**Figure 4.2.** Graph describing the error margin of the first mix for a series of samples, created using Microsoft Excel 2011.

In figure 4.3 a visual representation of the test results is shown.

![Reference samples](image)

**Figure 4.3.** Reference samples at the bottom of every pile. Closest to the reference is the final result. On top of every pile is the first mix.


4.2 Discussion

This section serves to present some of the issues faced during this project as well as an overall discussion of the results in relation to posed research questions.

4.2.1 Input Issues

The color sensor used in this project had difficulties reading a relatively wide range of colors. This problem was most pronounced for colors with red influences. Since the color sensor did not give reliable results for some of the samples, the colors that were tested had to be narrowed down in order to evaluate the performance of the system. Since good sensor data were acquired for all colors ranging between the primary colors yellow and blue the tests were carried out in this spectrum.

4.2.2 Possible Error Sources

The only instrument used for collecting data in this project is the color sensor. The color that the sensor reads can differ depending water to paint ratio in the paint solutions used in the machine, irregularities in the paint applied to the sample and the light shining though it when scanning the color.
4.2.3 System Performance

The accepted deviation was decided upon by experimentally checking the range of hue gotten by multiple scans of color samples that looked visually uniform. The conversion method used gives hue with a range of 0 to 393222, making this a deviation of less than 0.8% of the entirety of the scale. A little less than half of this scale can be produced with the two primary colors used in this project. The accepted deviation is just over 1.8% of the hue range that is used.

At first strictly linear mixing proportions were used for the first mix. This seemed to work well for blue half of the yellow to blue spectrum. However, we found that for hues closer to the yellow end of the spectrum this ended up creating a much too green color and the amount of yellow paint needed to adjust it to the correct hue was much too great. To address this problem the formula for the proportions was adjusted and a chart of the final version is presented in figure 4.4.

![Figure 4.4](#)

Figure 4.4. A chart showing the fraction of blue paint in the initial mix, dependent on the input hue, created using Microsoft Excel 2016.

The overall performance of the demonstrator is quite pleasing. Looking at the results the accuracy after the first mix is lower than the allowed error margin percentage.
4.3 Conclusions

The following sections serve to discuss the results in relation to the research questions posed in section 1.2.

What is the state of the art in terms of methodologies for color replication automation using different sets of primary colors? At the current time color replication automation is still quite an involved process. Like stated under section 1.1 it generally requires several different expensive machines. When it comes to the question of using different sets of primary colors the HSV color model was created to mimic the way an artist mixes paint[4]. This model was therefore adopted in this project as the basis for the method developed.

Which of the methodologies are cost-effective in practice? When conducting the research for this project none of the found methodologies seem to be cost effective. The products in use are generally very expensive, a spectrophotometer used in a paint store, the analog for the sensor used in this project, can cost anywhere from a few hundred dollars up to several thousand. This project therefore attempts to use a much cheaper sensor solution. A sensor from Seeed Studio, aimed more at hobbyists and makers was chosen.

How accuracy can be defined and measured in the context of the chosen methodology? Given the chosen color model for this project, HSV, the accuracy of the model can be measured by the difference in hue value between the test and reference sample.

Given that a cost-effective methodology has been chosen, what is its accuracy in automatic color replication? The methodology used in this project is based on reading the color mixed by the demonstrator which means that the accuracy of this project is mainly based on the accuracy of the sensor. The accuracy of the sensor was experimentally found by conducting several measurements on the same sample to see how much the hue value varied. Deviation as low as 3000, less than 0.8% of the hue scale, from the reference hue was found to work.
Chapter 5

Future Work and Improvements

This chapter aims to talk about possible improvements to the demonstrator and visions of the project as a whole.

5.1 Immediate improvements

With a better sensor and/or a better conversion from raw sensor readings to RGB, it would be possible to get good results when scanning colors of any hue. Better in this case means that the transition in sensor data between different samples of similar hue is more continuous across the whole spectrum than with the sensor and conversion currently in use. As of now this mainly is required for colors with red influences. Smooth transitions in sensor data between visually similar samples is required to maintain the effectiveness of the method developed in this project.

When the hue part of the HSV color model is fully covered, saturation and value can be taken into consideration. If good readings for these values can be obtained the method could be expanded to the original scope of the project and almost any color should be possible to replicate.

5.2 Future Visions

Given time and resources more of what is currently manually operated could be automated, this could lead to both more precise results and a faster testing process that uses up less of the paint for each sample to scan. Another possible improvement would be for the machine to save data for successful results and alters the mixing method to allow for a better first result and faster performance.
Bibliography


Appendix A

Arduino code

/* ////////////////////////////////////////////
// Automated Color Mixer
// A mechatronics bachelor project
// Authors: Max Lindgren and Max Thiel
// TRITA MMK 2017:32 MDAB 650
////////////////////////////////////////*/

#include <Wire.h>
#include <GroveColorSensor.h>
#include <Registers.h>
#include <Utilities.h>
#include <math.h>

// Sensor connection.
// wire -> arduino pin
// YELLOW, SCL -> A5
// WHITE, SDA -> A4
// RED, VCC -> 5V
// BLACK, GND -> GND

// Hue value boundaries for yellow -> blue spectrum with the used setup.
#define YB_RANGE_LOWER_BOUND 62500
#define YB_RANGE_UPPER_BOUND 226000
#define YB_RANGE_SIZE (YB_RANGE_UPPER_BOUND - YB_RANGE_LOWER_BOUND)

// The accepted deviation of hue value.
#define WORKING_ERROR_MARGIN 3000

// How many milliseconds to pour in total for the initial mix.
# define INITIAL_FLUID_AMOUNT 4000
// The maximum allowed time to pour when adjusting a mix.
# define MAX_ADJUST_FLUID_AMOUNT 2000
// The minimum allowed time to pour when adjusting a mix,
// experimentaly found to consistently give of one drop.
# define MIN_ADJUST_FLUID_AMOUNT 75

// Used in hsv conversion,
// the hue spectrum is devided into six sections.
// This is the size of the value span for each section.
# define HEXAGONAL_EDGE_LENGTH 65537

// Button pin ids.
enum Buttons
{
    SCAN_PIN = 2,
    ACCEPT_PIN
};

// LED pin ids.
enum LEDPins
{
    GREEN_PIN = 4,
    YELLOW_PIN,
    RED_PIN
};

// Valve pin ids.
enum ValvePins
{
    YELLOW_VALVE_PIN = 9,
    BLUE_VALVE_PIN,
    RED_VALVE_PIN,
    BLACK_VALVE_PIN,
    WHITE_VALVE_PIN
};

// States of the machine.
enum MachineState
{
    STARTUP,
    AWAITING_COLOR,
    Initializing_MIX,
    MEASURING,
ADJUSTING_MIX

// Bit flags for which LEDs to light.
enum LEDBitFlags
{
    GREEN_LED = 1 << 0,
    YELLOW_LED = 1 << 1,
    RED_LED = 1 << 2,
    ALL_LEDS = GREEN_LED | YELLOW_LED | RED_LED
};

// Returns the distance hues distance from YB_RANGE_UPPER_BOUND (blue)
// as a fraction of YB_RANGE_SIZE.
float bluedist(unsigned long hue)
{
    float dist = ((float)(YB_RANGE_SIZE - (hue - YB_RANGE_LOWER_BOUND)))/YB_RANGE_SIZE;
    return dist;
}

// Returns the fraction of blue paint tho pour i.e. yellow fraction is:
// 1 - bluefract(hue)
float bluefract(unsigned long hue)
{
    double frac = -bluedist(hue);
    return min(frac+1,exp(frac*7+3));
};

// A template function that returns the middle of three values.
template <class T> const T& mid(const T& a, const T& b, const T& c)
{
    return (a<max(b,c))?max(a,min(b,c)):max(b,c);
};

// Function that converts a color in RGB to the same color in HSV
// (hue, saturation, value).
void rgbtohsv(int red, int green, int blue,
              unsigned long& hue, unsigned long& saturation, unsigned int& value)
{
    // Calculate max, min and mid of red green and blue.
    unsigned long M = max(red,max(green,blue));
    unsigned long m = min(red,min(green,blue));
    unsigned long c = mid(red,green,blue);
// Assign value.
value = M;

// Calculate the difference between max and min.
unsigned long diff = M - m;
if (diff == 0)
{  
saturation = 0;
  hue = 0;
  return;
}

// Determine sector index.
unsigned long I = 0;
if (M == red) I = (m == blue) ? 0 : 5;
else if (M == green) I = (m == blue) ? 1 : 2;
else I = (m == red) ? 3 : 4;

// Calculate saturation.
saturation = ((diff << 16) - 1) / value;

// Calculate the fractional part of hue.
unsigned long F = ((c - m) << 16) / diff + 1;

// Inverse F for odd sector indexes.
if (I == 1 || I == 3 || I == 5)
  F = HEXAGONAL_EDGE_LENGTH - F;

// Calculate integer based hue.
hue = HEXAGONAL_EDGE_LENGTH * I + F;

// The main class of the machine.
class ColorMixer
{
public:
  // Public functions.

  // Constructor.
  ColorMixer();
  // Destructor.
  ~ColorMixer();
// Called in the arduino main loop.
void run();

private:
// Private functions.

// Sets the state of the machine.
void setState(MachineState state);
// Takes bit flags and turns the LEDs on and off accordingly.
void setLEDs(unsigned int LEDBitFlags);
// Does a scan using the sensor.
// Gives the raw results as an array of ints, 8 in length
// and the hue as the given unsigned long.
void scanColor(int* raw, unsigned long& hue);

// State functions, each called from run dependant on current state.
void initialize();
void awaitColor();
void initializeMix();
void measure();
void adjustMix();

// Private members
// The machine state.
MachineState m_state;
// Pointer to the sensor class.
GroveColorSensor* m_pColorSensor;
// Flag to keep track of if a scan has been made.
bool m_read;
// Flag used to iterate through all valves for
// evacuating air at startup.
int m_initializingColor;

// Holds the raw sensor data of the scanned reference color.
int m_raw[8];
// Holds the hue of the scanned reference color.
unsigned long m_hue;
// Holds the hue of the scanned mixed color.
unsigned long m_measuredhue;
// Holds the amount of milliseconds the valves have been
// opened for in total for the current mix.
unsigned long m_mixedamount;
// Constructor, set default values for all members and
// initialize arduino pins, serial and wire (IC2).
ColorMixer::ColorMixer()
: m_state(STARTUP)
, m_pColorSensor(0)
, m_read(false)
, m_initializingColor(YELLOW_VALVE_PIN)
, m_raw{0,0,0,0,0,0,0,0}
, m_hue(0)
, m_measuredhue(0)
, m_mixedamount(0)
{
    pinMode(GREEN_PIN, OUTPUT);
    pinMode(YELLOW_PIN, OUTPUT);
    pinMode(RED_PIN, OUTPUT);
    pinMode(SCAN_PIN, INPUT);
    pinMode(ACCEPT_PIN, INPUT);
    pinMode(YELLOW_VALVE_PIN, OUTPUT);
    pinMode(BLUE_VALVE_PIN, OUTPUT);
    pinMode(RED_VALVE_PIN, OUTPUT);
    pinMode(BLACK_VALVE_PIN, OUTPUT);
    pinMode(WHITE_VALVE_PIN, OUTPUT);
    Serial.begin(9600);
    Wire.begin();
}

// Destructor, free up any allocated memory.
ColorMixer::~ColorMixer()
{
    // Free the allocated memory
    if (m_pColorSensor)
        delete m_pColorSensor;
}

// State function, gets the machine ready to scan a color to mix.
// Additional functionality was bit to go directly to MEASURING state
// in order to make it possible to resume mixing an already started mix.
void ColorMixer::initialize()
{
    // Setup the sensor
    if (!m_pColorSensor)
    {
        setState(STARTUP);
        m_pColorSensor = new GroveColorSensor();
    }
m_pColorSensor->ledStatus = 1;
delay(1000);
// Set status LEDs
setLEDs(GREEN_LED | RED_LED);

// Print instructions to serial.
Serial.println("Evacuate air from system");
Serial.println("Open valve on pin:");
Serial.println(m_initializingColor);
}

// Open valves by the push of a button to evacuate all
// the air in the silicone tubes.
if (m_initializingColor <= WHITE_VALVE_PIN)
{
    // Open if button is pressed, else close.
    // Signal with LEDs.
    int open = !digitalRead(SCAN_PIN);
    if (open)
    {
        digitalWrite(m_initializingColor, HIGH);
        setLEDs(ALL_LEDS);
    }
    else
    {
        digitalWrite(m_initializingColor, LOW);
        setLEDs(GREEN_LED | RED_LED);
    }
}

// If other button is pressed move on to next valve,
// or if done, choose what state to go to.
// m_read is used to make sure that this is not triggered
// more than once without releasing the button in-between.
int accept = !digitalRead(ACCEPT_PIN);
if (accept && !m_read)
{
    m_read = true;
    digitalWrite(m_initializingColor, LOW);
    setLEDs(GREEN_LED | RED_LED);
    m_initializingColor++;
    Serial.println("Air evacuated!");
    if (m_initializingColor <= WHITE_VALVE_PIN)
    {

APPENDIX A. ARDUINO CODE

```cpp
Serial.println("Moving on to valve on pin:");
Serial.println(m_initializingColor);
}
else
{
    Serial.println("Initialization done!");
    Serial.println("Choose state: [L] MEASURING / [R] AWAITING_COLOR");
    m_read = false;
}
delay(500);
}
else if (!accept)
    m_read = false;
}
// Here what state to move into is chosen
// ([L] MEASURING / [R] AWAITING_COLOR)
// If [L] is pressed the sensor does a scan and saves that
// as the reference, sequential presses will redo the scan.
// After this [R] has to be pressed to accept and
// set the state to MEASURING.
else
{
    int left = !digitalRead(SCAN_PIN);
    int right = !digitalRead(ACCEPT_PIN);

    if (right)
    {
        if (m_read)
            setState(MEASURING);
        else
            setState(AWAITING_COLOR);
    }
    else if (left)
    {
       (scanColor(m_raw, m_hue);
        m_mixedamount = INITIAL_FLUID_AMOUNT;
        m_read = true;
    }
}
// State function, left button is pressed to
// scan a color to be used as a reference.
```
void ColorMixer::awaitColor()
{
    int scan = !digitalRead(SCAN_PIN);
    if (scan)
    {
        scanColor(m_raw, m_hue);
        m_read = true;
    }

    int accept = !digitalRead(ACCEPT_PIN);
    if (accept && m_read)
    {
        setState(INITIALIZING_MIX);
    }
}

void ColorMixer::initializeMix()
{
    if (m_hue >= YB_RANGE_LOWER_BOUND && m_hue <= YB_RANGE_UPPER_BOUND)
    {
        // Determine the ratio of blue to yellow paint to pour.
        float bluequota = bluefract(m_hue);
        float yellowquota = 1.0f - bluequota;
        char printbuffer[64] = "";
        // Pour yellow
        int pourtime = round(INITIAL_FLUID_AMOUNT*yellowquota);
        sprintf(printbuffer, "Pouring yellow for %d milliseconds.", pourtime);
        Serial.println(printbuffer);
        digitalWrite(YELLOW_VALVE_PIN, HIGH);
        delay(pourtime);
        digitalWrite(YELLOW_VALVE_PIN, LOW);
        m_mixedamount += pourtime;
        // Pour blue
        pourtime = round(INITIAL_FLUID_AMOUNT*bluequota);
        sprintf(printbuffer, "Pouring blue for %d milliseconds.", pourtime);
        Serial.println(printbuffer);
    }
APPENDIX A. ARDUINO CODE

digitalWrite(BLUE_VALVE_PIN, HIGH);
delay(pourtime);
digitalWrite(BLUE_VALVE_PIN, LOW);
m_mixedamount += pourtime;

// Move on to state MEASURING.
setState(MEASURING);
} else {
    Serial.println("Scanned color not within working range!");
    setState(AWAITING_COLOR);
}

// State function, measure how close the
// poured mix is from the reference hue.
// If it is within accepted deviation go to
// state AWAITING_COLOR to begin a new mix.
// If it is not within accepted deviation go to
// state ADJUSTING_MIX to pour more paint and adjust the hue.
void ColorMixer::measure() {
    // Scan color when left button is pressed.
    int scan = !digitalRead(SCAN_PIN);
    if (scan) {
        int raw[8];
        scanColor(raw, m_measuredhue);
        Serial.println(m_measuredhue);
        m_read = true;
    }

    // If there is a color scanned when right button is pressed,
    // check deviation and move on to state
    // AWAITING_COLOR or ADJUSTING_MIX accordingly.
    int accept = !digitalRead(ACCEPT_PIN);
    if (accept && m_read) {
        long diff = m_hue - m_measuredhue;
        diff = abs(diff);
        bool done = diff <= WORKING_ERROR_MARGIN;
        char printbuffer[64] = "";
        sprintf(printbuffer, "Looking for hue: %lu. Got hue: %lu.",

m_hue, m_measuredhue);
Serial.println(printbuffer);
delay(500);
sprintf(printbuffer, "Color comparison: %ld off. %s", diff,
(done ? "Done!" : "Adjust!");
Serial.println(printbuffer);
delay(500);
if(done)
  setState(AWAITING_COLOR);
else
  setState(ADJUSTING_MIX);
}

// State function, pour more paint into the mix.
// How much is dependant on the deviation in hue from the reference,
// where in the spectrum the reference is and
// what color of paint is poured.
void ColorMixer::adjustMix()
{
  long diff = m_hue - m_measuredhue;
  // Determine number of milliseconds to pour,
  // with upper bound MAX_ADJUST_FLUID_AMOUNT.
  int poutime = min(round(float(abs(diff))
    / YB_RANGE_SIZE * m_mixedamount), MAX_ADJUST_FLUID_AMOUNT);
  int pin = YELLOW_VALVE_PIN;
  if (diff >= 0)
  {
    pin = BLUE_VALVE_PIN;
    // Blue is being poured
    // If reference is close to the yellow end of the spectrum,
    // use less paint small amounts of blue with alter hue a lot here.
    poutime *= 0.2f + min(1.f, (1.f - bluedist(m_hue))*4.f) * 0.8f;
  }
  // Minimum poutime, make sure at least one drop is poured.
  poutime = max(poutime, MIN_ADJUST_FLUID_AMOUNT);
  char printbuffer[64] = "";
  sprintf(printbuffer,
    "Adjusting mix, pouring for %d milliseconds from valve on pin %d.",
    poutime, pin);
  Serial.println(printbuffer);
  digitalWrite(pin, HIGH);
delay(poutime);
}
APPENDIX A. ARDUINO CODE

digitalWrite(pin, LOW);
m_mixedamount += pourtime;

// Move back to state MEASURING.
setState(MEASURING);
}

// Machines main function, called from arduino loop.
// Calls corresponding state function given current state.
void ColorMixer::run()
{
    switch (m_state) {
        case STARTUP: initialize(); break;
        case AWAITING_COLOR: awaitColor(); break;
        case INITIALIZING_MIX: initializeMix(); break;
        case MEASURING: measure(); break;
        case ADJUSTING_MIX: adjustMix(); break;
    }
}

// Sets the machine to the given state,
// sets LEDs correspondingly and resets any needed members.
void ColorMixer::setState(MachineState state)
{
    switch (state) {
        case STARTUP:
            setLEDs(ALL_LEDS);
            Serial.println("State set: STARTUP");
            break;
        case AWAITING_COLOR:
            setLEDs(GREEN_LED);
            m_read = false;
            m_mixedamount = 0;
            Serial.println("State set: AWAITING_COLOR");
            break;
        case INITIALIZING_MIX:
            setLEDs(YELLOW_LED);
            Serial.println("State set: INITIALIZING_MIX");
            break;
        case MEASURING:
            setLEDs(GREEN_LED | YELLOW_LED);
            m_read = false;
            m_measuredhue = 0;
            Serial.println("State set: MEASURING");
            break;
    }
}
break;
    case ADJUSTING_MIX:
        setLEDs(YELLOW_LED);
        Serial.println("State set: ADJUSTING_MIX");
        break;
    }
    m_state = state;
}

// Turns on or off LEDs corresponding to given bit flags.
void ColorMixer::setLEDs(unsigned int LEDs)
{
    digitalWrite(GREEN_PIN, LEDs & GREEN_LED);
    digitalWrite(YELLOW_PIN, (LEDs & YELLOW_LED) >> 1);
    digitalWrite(RED_PIN, (LEDs & RED_LED) >> 2);
}

// Reads color using the color sensor.
void ColorMixer::scanColor(int* raw, unsigned long & hue)
{
    int red, green, blue = 0;
    unsigned int value = 0;
    unsigned long saturation = 0;
    m_pColorSensor->readRGB(&red, &green, &blue, raw);
    delay(300);
    red = raw[2] + 256*raw[3];
    green = raw[0] + 256*raw[1];
    blue = raw[4] + 256*raw[5];
    // Tweak of values, made to make yellow -> blue spectrum work better.
    red = red * 1.35;
    blue = blue + pow(max(blue-130, 0), 1.02);

    // Factor values down to make highest value at most 255.
    int cmax = max(red,green);
    cmax = max(cmax,blue);
    if (cmax > 255)
    {
        float factor = 255.0f / cmax;
        red = round(red*factor);
        green = round(green*factor);
        blue = round(blue*factor);
    }

    // Convert RGB color to HSV.
rgbtohsv(red, green, blue, hue, saturation, value);

    // Transfer read data over serials.
    char printbuffer[80] = "";
    sprintf(printbuffer,
"color,%d,%d,%d,\lu,%lu,%u,%d,%d,%d,%d,%d,%d",
red, green, blue,
hue, saturation, value,
raw[0], raw[1], raw[2], raw[3], raw[4], raw[5], raw[6], raw[7]);
    Serial.println(printbuffer);
    m_pColorSensor->clearInterrupt();

    // Pointer to the main class.
    ColorMixer* g_pColorMixer = NULL;

    void setup() {
        // Create instance of the main class.
        if (!g_pColorMixer)
        {
            g_pColorMixer = new ColorMixer();
        }
    }

    void loop()
    {
        // Call the run function on the instance of the main class.
        g_pColorMixer->run();
    }
Appendix B

Python code

""" ///////////////////////////////////////////////////////////////////////////////////////////
// Automated Color Mixer
// A mechatronics bachelor project
// Authors: Max Lindgren and Max Thiel
// TRITA MMK 2017:32 MDAB 650
///////////////////////////////////////////////////////////////////////////////////////////
""

import serial
import time
from tkinter import *

# Function that returns the middle value out of three inputs.
def mid(a,b,c):
    return max(a, min(b,c)) if (a< max(b,c)) else max(b,c)

# Conversion function for a color given in RGB to a color in HSV.
def rgbtohsv(red, green, blue):
    # Calculate max, min and mid of red green and blue.
    M = max(red,max(green,blue))
    m = min(red,min(green,blue))
    c = mid(red,green,blue)

    # Assign value.
    value = M

    # Calculate the difference between max and min.
    diff = M - m
    if (diff == 0):
        return 0, 0, value
# Determine sector index
I = 0
if (M == red):
    I = 0 if (m == green) else 5
elif (M == green):
    I = 1 if (m == blue) else 2
else:
    I = 3 if (m == red) else 4

# Calculate saturation
saturation = ((diff << 16) - 1) // value

# Calculate the fractional part of hue
F = ((c - m) << 16) // d + 1

# E is the hexagonal edge length
E = 65537

# Inverse F for odd sector indexes
if (I == 1 or I == 3 or I == 5):
    F = E - F

# Calculate integer based hue
hue = E * I + F
return hue, saturation, value

# Conversion function for a color given in HSV to a color in RGB.
def hsvtorgb(hue, saturation, value):
    if saturation == 0 or value == 0:
        return 0, 0, 0

    # Calculate diff
diff = ((saturation*value) >> 16) + 1

    # Calculate m, the minimal value of R, G and B
    m = value - diff

    # E is the hexagonal edge length
    E = 65537

    # Determine hue index sector and calculate the middle component.
    if hue < E:
        return value, ((hue * diff) >> 16) + m, m
    elif hue >= E and hue < 2*E:
return ((2*E - hue) * diff) >> 16) + m, value, m
elif hue >= 2*E and hue < 3*E:
    return m, value, (((hue - E*2) * diff) >> 16) + m
elif hue >= 3*E and hue < 4*E:
    return m, (((E*4 - hue) * diff) >> 16) + m, value
elif hue >= 4*E and hue < 5*E:
    return ((hue - E*4) * diff) >> 16) + m, m, value
elif hue >= 5*E:
    return value, m, (((E*6 - hue) * diff) >> 16) + m

# Main class, holds the tkinter and the serial interface.
class ColorDisplay:
    def __init__(self):
        # Init all members
        self.root=None
        self.rgbframe=None
        self.rawframe=None
        self.hsvframe=None
        self.hueframe=None
        self.ser = None

    def __del__(self):
        # Make sure serial is closed.
        if self.ser:
            self.ser.close()
            self.ser = None

    def show(self):
        # Display tkinter, run mainloop
        self.root = Tk()
        # Add update to the tkinter loop.
        self.root.after(0, self.update)
        self.root["width"] = 960
        self.root["height"] = 540
        self.root.resizable(width=False, height=False)
        self.createframes();
        self.root.mainloop()

    def createframes(self):
        # Create the four colored frames.
        self.rgbframe = Frame(self.root, bg = "orange")
        self.rgbframe["width"] = self.root["width"] / 4
        self.rgbframe["height"] = self.root["height"]
        self.rgbframe.pack(side=LEFT)
APPENDIX B. PYTHON CODE

```python
self.tweakframe = Frame(self.root, bg = "blue")
self.tweakframe["width"] = self.root["width"] / 4
self.tweakframe["height"] = self.root["height"]
self.tweakframe.pack(side=LEFT)
self.hsvframe = Frame(self.root, bg = "green")
self.hsvframe["width"] = self.root["width"] / 4
self.hsvframe["height"] = self.root["height"]
self.hsvframe.pack(side=LEFT)
self.hueframe = Frame(self.root, bg = "yellow")
self.hueframe["width"] = self.root["width"] / 4
self.hueframe["height"] = self.root["height"]
self.hueframe.pack(side=LEFT)

def setcolor(self, red, green, blue,
    hue, saturation, value,
    tweakr, tweakg, tweakb):
    # Set the color of rgbframe.
    rgbcolor = " #%02 x %02 x %02 x" % (red, green, blue)
    self.rgbframe["bg"] = rgbcolor

    # Set the color of tweakframe.
    tweakcolor = " #%02 x %02 x %02 x" % (tweakr, tweakg, tweakb)
    self.tweakframe["bg"] = tweakcolor

    # Calculate and set color of hsvframe.
    r, g, b = hsvtorgb(hue, saturation, value)
    hsvcolor = " #%02 x %02 x %02 x" % (r, g, b)
    self.hsvframe["bg"] = hsvcolor

    # Calculate and set color of hueframe.
    r, g, b = hsvtorgb(hue, 65535, 255)
    huecolor = " #%02 x %02 x %02 x" % (r, g, b)
    self.hueframe["bg"] = huecolor

    # Print values to the console.
    print("tweak: " + tweakcolor + " rgb: " + rgbcolor +
        " HSV in rgb: " + hsvcolor + " Hue: " + str(hue) +
        " -- Hue in rgb: " + huecolor)

def initSerial(self):
    # Try to establish serial connection to the arduino.
    if not self.ser:
        try:
            self.ser = serial.Serial(port='COM4',\
```
baudrate=9600,
parity=serial.PARITY_NONE,
stopbits=serial.STOPBITS_ONE,
bytesize=serial.EIGHTBITS,
timeout=0)
print("connected to: " + self.ser.portstr)
except serial.SerialException as e:
    if self.ser:
        self.ser.close()
        self.ser = None

# Read data from the serial connection.
def readSerial(self):
    seq = []

    if self.ser:
        try:
            startread = time.time()
            for char in self.ser.read():
                seq.append(chr(char))
            while len(seq) > 0:
                for char in self.ser.read():
                    seq.append(chr(char))

            # End acquisition when line break is received.
            if seq[-1] == '\n':
                # Join data into a single string and
                # strip the line break.
                joined_seq = ''.join(str(v) for v in seq)
                joined_seq.strip()

                # Split the string at every comma.
                data = joined_seq.split(',
            # Read the data.
            self.readData(data)
            break
        elif startread - time.time() > 10:
            print("readSerial timeout, took more than 10 seconds."
        print("data gotten: " + ''.join(str(v) for v in seq)
        break

47
APPENDIX B. PYTHON CODE

# If the serial connection fails,
# close it and set the member to None.
# This will make the program try
# to re-establish the connection.
except serial.SerialException as e:
    if self.ser:
        self.ser.close()
        self.ser = None

def readData(self, data):
    # If a color "package" is received, read the color,
    # else print to console.
    if data[0] == "color":
        print(''.join(str(v) + ',' for v in data))
        self.readColor(data[1:])
    else:
        print(''.join(str(v) for v in data))

# Reads color received over serial and
# sets colors of frames accordingly.
def readColor(self, rgbhue):
    if len(rgbhue) == 14:
        # Combine the raw color values for visual representation.
        r = int(rgbhue[8]) + 256*int(rgbhue[9])
        g = int(rgbhue[6]) + 256*int(rgbhue[7])
        b = int(rgbhue[10]) + 256*int(rgbhue[11])

        cmax = max(r,g,b)
        if cmax > 255:
            factor = 255 / cmax
            r = r*factor
            g = g*factor
            b = b*factor
        r = int(r)
        g = int(g)
        b = int(b)

        # Set the colors of the frames.
        self.setcolor(int(rgbhue[0]),
                      int(rgbhue[1]),
                      int(rgbhue[2]),
                      int(rgbhue[3]),
                      int(rgbhue[4]),
                      48
int (rgbhue[5]),
  r, g, b)

# Print some of the data to console.
print("Raw data: \%s\ % rgbhue[3]
print(rgbhue[8])
print(rgbhue[9])
print(rgbhue[6])
print(rgbhue[7])
print(rgbhue[10])
print(rgbhue[11])
print(rgbhue[12])
print(rgbhue[13])

def update(self):
    # Try to establish serial connection.
    self.initSerial()

    # Try to read data over the serial connection.
    self.readSerial()

    # Make sure update is called again next loop.
    self.root.after(1, self.update)

def main():
    # Start the program.
    CD = ColorDisplay()
    CD.show()

if __name__ == '__main__':
    # Call main!
    main()