[COLOR ID DOCUMENTATION] V1.0

WHAT IS IT:

The Color Identifier Project is an ongoing project that takes color input via an RGB sensor, and outputs an assigned pitch in the Key of C Major.

HOW IT WORKS:

Most of the design was done from scratch for this project. The main components to consider are the **Arduino Uno**, the **RGB Sensor**, and the **Piezzo Buzzer.**

When the Arduino is plugged into its power source, it immediately begins taking in input from the sensor. **CAUTION: Don't look directly into the RGB Sensor upon plugging in the Arduino. The sensor has a bright LED light and may cause eye discomfort if looked directly at.**

The **RGB Sensor** measures four dimensions: red, green, blue, and brightness. The code has converted the RGB space to HSV space, and relies on the **hue** value to make a pitch determination.

The small blue knob is a **potentiometer**, and controls the amount of signal that's allowed to pass. It essentially functions as a volume knob for this project.

Code for this project was developed in the Arduino IDE, available at this website: [http://arduino.cc]. Please look in the associated folder for the code. The language is "C-Like", but it is not exactly C.

Since there's currently no enclosure for the device, measurements vary greatly, and may require some recalibration based on the space you're in. Calibration depends on ambient light and desired proximity to the sensor. Future implementations may wish to explore different projections or dimensional reductions of the color spaces.

COMPONENTS:

Arduino Uno – This is the microcomputer that determines the association of color and pitch. Code is uploaded to the device via the Arduino IDE.

RGB Sensor – This is the main input component.

Piezzo Buzzer – This is the main output component.

Potentiometer - This component controls the amount of signal that is allowed to pass. In this project, it functions as a volume knob.

WIRING:

COMPONENTS:
-ARD (Arduino Uno)
-RGB (RGB Sensor)
-PZB (Piezzo Buzzer [or desired Audio Out])
-PTN (Potentiometer)

PIN OUT FROM ARDUINO UNO:

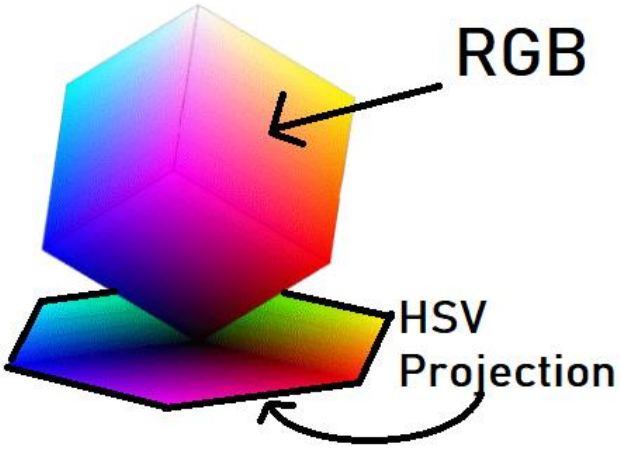
ARD1 → RGB LED
ARD 5V → RGB VIN
ARD GND → RGB GND

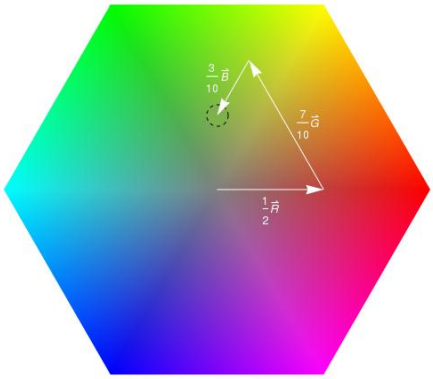
ARD 10 → PZB ⊕
PZB ⊖ → PTN ⊕
PTN ⊖ →

HSV SPACE:

HSV is based on a circle – the hue is the degrees away from a fixed radius (positioned at RED = 0 degrees).
Since this is coming from RGB input, the HSV space will be represented by a hexagon. Conversion between the spaces can be seen within the Arduino code.

Each vector in the right-most image represents movement in the RGB cube that eventually determines the HSV coordinates.





FUTURE DEVELOPMENT:

- Explore dimensional reduction techniques to reduce variance/sensitivity of pitches.

- Add a button/switch that delays the start of the device past power-on.

CHANGE LOG:

12/27/19 – Initial Documentation Developed.

CODING:

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"

//Chosing the pins for the colors
#define redpin 3
#define greenpin 5
#define bluepin 6

// set to false if using a common cathode LED
// For an anode pin, connect to +5V
// For cathode pin, connect the CMN to ground.
#define commonAnode true

//Chosing pin for the buzzer
#define speakerPin 10

//Two lines from "colorview" code from RGB sensor
//Assumed purpose is to interpret/convert to HEX; not too sure.
byte gammatable[256];
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);


void setup() {

  //Set up for the buzzer starts here
  pinMode(speakerPin, OUTPUT);
  //Set the output pin for the speaker

  //Set up for RGB sensor starts here
  Serial.begin(9600);
  Serial.println("Color View Test!");

  if (tcs.begin()) {
    Serial.println("Found sensor");
  } else {
    Serial.println("No TCS34725 found ... check your connections");
    while (1); // halt!
  }

  //RGB LED outputs can be put here, but we don't have an RGB LED light yet.

  // turns on LED, set to TRUE to turn it off.
  tcs.setInterrupt(false);

// This creates our gamma table
// it helps convert RGB colors to what humans see
// Creates gamma values which is used for an LED light interpretation.
  for (int i=0; i<256; i++) {
    float x = i;
    x /= 255;
    x = pow(x, 2.5);
    x *= 255;

    if (commonAnode) {
      gammatable[i] = 255 - x;
    } else {
      gammatable[i] = x;
    }
    //Serial.println(gammatable[i]);
  }

}

void loop() {
  uint16_t clear, red, green, blue;

  // takes 50ms to read
```

```
  // uncomment to create "strobe" effect
  //delay(60);

  //Input from RGB sensor
  tcs.getRawData(&red, &green, &blue, &clear);

  /* THIS IS FOR THE GAMMA TABLE
     Creates the values that are 1000+ for visualization
  Serial.print("C:\t"); Serial.print(clear);
  Serial.print("\tR:\t"); Serial.print(red);
  Serial.print("\tG:\t"); Serial.print(green);
  Serial.print("\tB:\t"); Serial.print(blue);
  */

  // Calculates the HEX color codes
  uint32_t sum = clear;
  float r, g, b;
  r = red; r /= sum;
  g = green; g /= sum;
  b = blue; b /= sum;

  float hue, sat, val, max_rgb, min_rgb;

  max_rgb = max(r, g);
  max_rgb = max(max_rgb, b); // max_rgb is also value -- which is effectively brightness!
use for different octaves
  min_rgb = min(r, g);
  min_rgb = min(min_rgb, b);


  if (max_rgb == r) {
    hue = ((g - b)/(max_rgb - min_rgb))/6;
  }
  else if (max_rgb == g) {
    hue = ((b - r)/(max_rgb - min_rgb)+2)/6;
  }
  else {
    hue = ((r - g)/(max_rgb - min_rgb)+4)/6;
  }
  if (hue < 0) {
    hue += 1;
  }

   r *= 256; g *= 256; b *= 256;

  //Displays color values in serial monitor
  Serial.print("C:\t"); Serial.print(clear);
  Serial.print("\tR:\t"); Serial.print(r);
  Serial.print("\tG:\t"); Serial.print(g);
  Serial.print("\tB:\t"); Serial.print(b);
  Serial.print("\t");
  Serial.print((int)r, HEX); Serial.print((int)g, HEX); Serial.print((int)b, HEX);
  Serial.print("\tHue:\t"); Serial.print(hue);
  Serial.println();

//Decision tree to decide pitches
if(clear < 1000) {
     //Very Red
     if (((hue >= 0.00) && (hue <= 0.02)) || (hue == 1.00))
     {
         //RED
         play('a', 1);
     }
}
else if ((clear > 1000) && (clear < 3500)) {
     if ((hue >= 0.01) && (hue <= 0.02))
     {
        //ORANGE
        play('g', 1);
     }
     else if ((hue >= 0.10) && (hue <= 0.13))
     {
```

```
        //YELLOW
        play('C', 1);
     }
     else if ((hue >= 0.24) && (hue <= 0.30))
     {
        //LIGHT GREEN
        play('D', 1);
     }
     else if ((hue >= 0.38) && (hue <= 0.46))
     {
        //DARK GREEN
        play('E', 1);
     }
     else if ((hue >= 0.55) && (hue <= 0.56))
     {
        //LIGHT BLUE
        play('F', 1);
     }
     else if (((hue >= 0.58) && (hue <= 0.61)))
     {
          //PURPLE
          play('A', 1);
     }
     else if ((hue >= 0.57) && (hue <= 0.58))
     {
          //DARK BLUE
          play('G', 1);
     }
    // else if ((hue >= 0.62) && (hue <= 0.70))
    // {
        //PURPLE
      //  play('G', 1);
     //}
     else if ((hue >= 0.97) && (hue <= 0.98))
     {
        //MAGENTA
        play(' ', 1);
     }
     else if ((hue >= 0.06) && (hue <= 0.08))
     {
        //BROWN
        play(' ', 1);
     }
}
else if ((clear >= 3500) && (clear < 13000))
{
     if ((hue >= 0.01) && (hue <= 0.02))
     {
        //ORANGE
        play('b', 1);
     }
     else if ((hue >= 0.10) && (hue <= 0.13))
     {
        //YELLOW
        play('C', 1);
     }
     else if ((hue >= 0.53) && (hue <= 0.56))
     {
        //LIGHT BLUE
        play('F', 1);
     }
}
else if(clear > 13000)
{
     if ((hue >= 0.53) && (hue <= 0.56))
     {
        //LIGHT BLUE
        play('F', 1);
     }
     else{
     //WHITE
     play('B',1);
```

```
    }
}

}


void play( char note, int beats)
{
  int numNotes = 14;  // number of notes in our note and frequency array (there are 15 values, but arrays start at 0)

  //Note: these notes are C major (there are no sharps or flats)

  //this array is used to look up the notes
  char notes[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C', 'D', 'E', 'F', 'G', 'A', 'B', ' '};
  //this array matches frequencies with each letter (e.g. the 4th note is 'f', the 4th frequency is 175)
  int frequencies[] = {131, 147, 165, 175, 196, 220, 247, 262, 294, 330, 349, 392, 440, 494, 523};

  int currentFrequency = 0;    //the frequency that we find when we look up a frequency in the arrays
  int beatLength = 150;   //the length of one beat (changing this will speed up or slow down the tempo of the song)

  //look up the frequency that corresponds to the note
  for (int i = 0; i < numNotes; i++)  // check each value in notes from 0 to 14
  {
    if (notes[i] == note)             // does the letter passed to the play function match the letter in the array?
    {
      currentFrequency = frequencies[i];   // Yes! Set the current frequency to match that note
    }
  }

  //play the frequency that matched our letter for the number of beats passed to the play function
  tone(speakerPin, currentFrequency, beats * beatLength);
  //wait for the length of the tone so that it has time to play
  //a little delay between the notes makes the song sound more natural
}
```