in progress

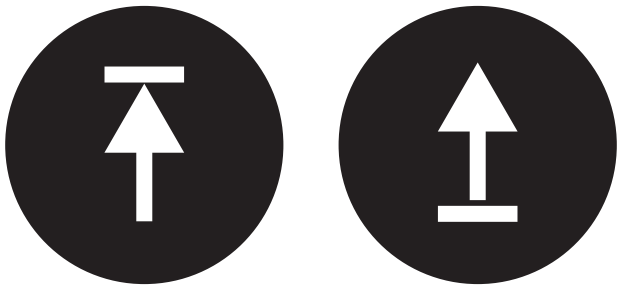
eric li | vis216

“i do, however, believe that the spoon is continually changing because we haven’t yet found its true form”

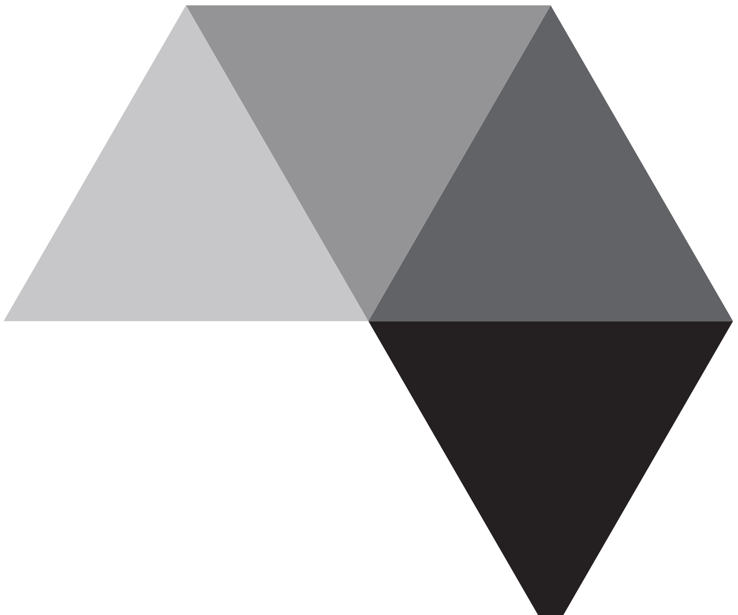
max bill, *continuity and change*

“what i advocate for … is a search for the constant, a search for the valid *gestalt*. *gestalt*, in this sense, is distinguished by its essential simplicity – not an artifical simplification, not stylisation, but simple and correct function.”

stop / go



in process



in progress

rgb vs cmyk

in modern media, there are two primary color spaces: the digital rgb color space and the print cmyk color space. briefly, rgb is an additive color space consisting of the colors red, gree, and blue that involves using visible light to create colors. this method forms the basis for all modern electronic displays. conversely, cmyk is a subtractive color space consisting of cyan, magenta, yellow, and black that forms the basis for all printing.

the differences between these color spaces is extremely important as people often will be working with both concurrently during a project. it is therefore crucial for one to understand the fundamental differences in the way that these color spaces operate and how they relate to one another.

back to basics

when it comes to illustrating the differences between these two color spaces, it is useful to distill each down to its fundamental unit. whereas atoms are the fundamental unit of matter, pixels are the fundamental unit of the rgb color space and ink drops are the fundamental unit of the cmyk color space.

by distilling each color space down to its fundamental unit, particularly with rgb, we are able to eliminate any discrepencies that arise between manufacturers such as color temperatures and screen calibrations. therefore, i chose to represent the rgb color space using the bare minimum: an rgb light emitting diode (led). i also chose to represent each space in the medium that it is meant for, digital for rgb and print for cmyk.

the concept

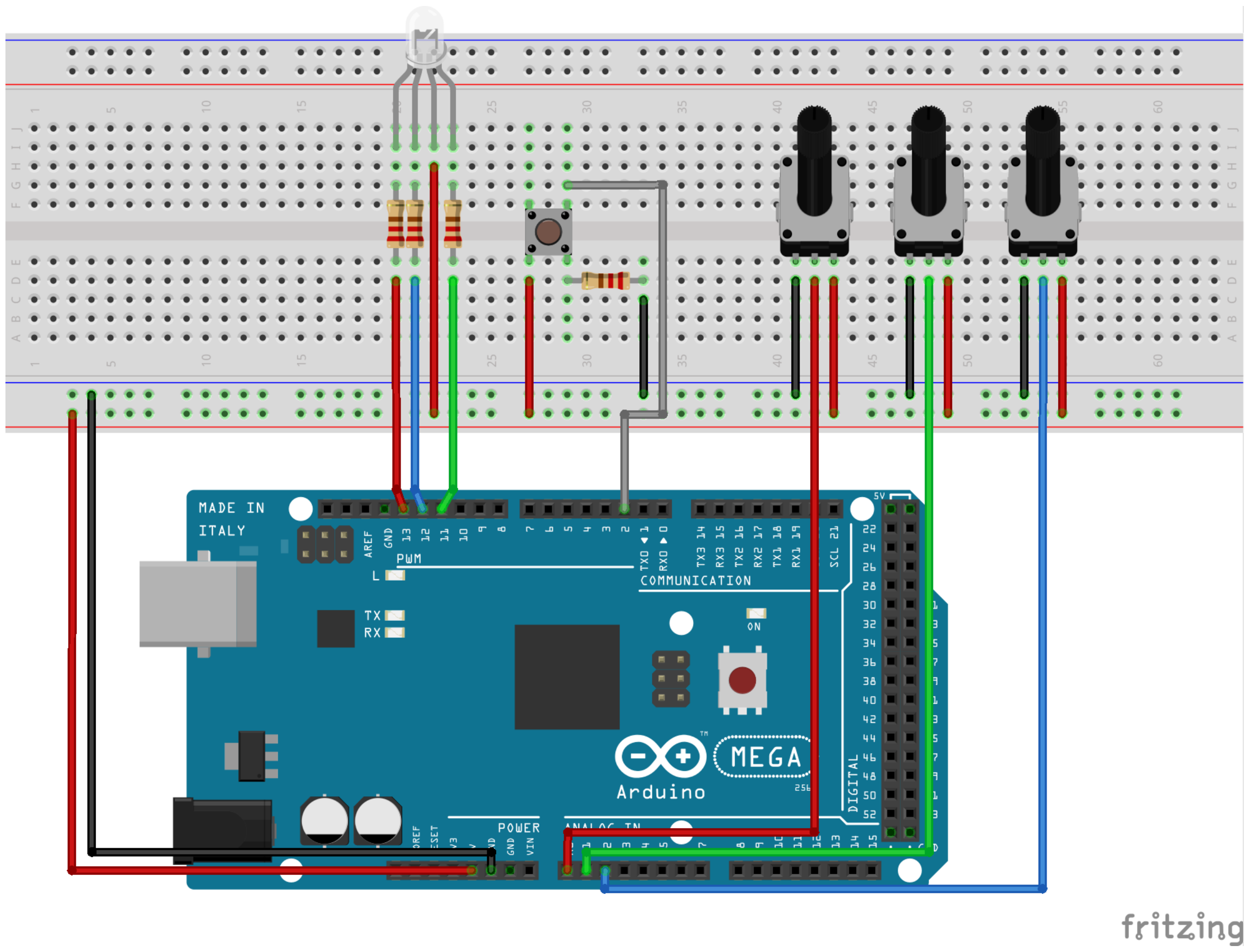
conceptually, the goal of my project to allow a user to discover that there are inherent differences between the two color spaces.

to illustrate that rgb is an additive color space, an rgb led connected to three knobs (potentiometers) was used. these knobs could be turned, changing each color value so that a user could discover for him/herself all the different color combinations possible in this additive color space. the led itself was housed in a 3x3 inch (the size of a post it note) 3d-printed black cube covered in a translucent gray plastic. this represents the fundamental unit of digital displays - a pixel.

the cmyk color space was represented using an epson stylus pro 4800 printer. ink was chosen over laser toner because it was closer to the original medium by which this color space existed. in representing each color space in its own unique medium rather than all on print or on a screen, the user is able to discover many more intricate details of each space.

i chose to link the two color spaces using the intuitive mechanism of a button. once a person had picked a color they liked on the "pixel," they could press a button and the exact same color would be printed out on a 3x3 inch square using the epson. of course, the color would not be an exact match. this discrepancy would allow the user to discover that in fact, the two color spaces do not provide a one-to-one correspondence.

the resulting print out would be pinned up on a wall, titled "in progess" which illustrates the almost limitless combinations of colors available. it also provides a static display of colors only available to the print medium whereas the pixel provides a dynamic, always changing, display of colors inherent to the digital medium.


MADE IN
ITALY
PWM
COMMUNICATION
POWER
ANALOG IN
DIGITAL
MEGA
Arduino
fritzing

implementation

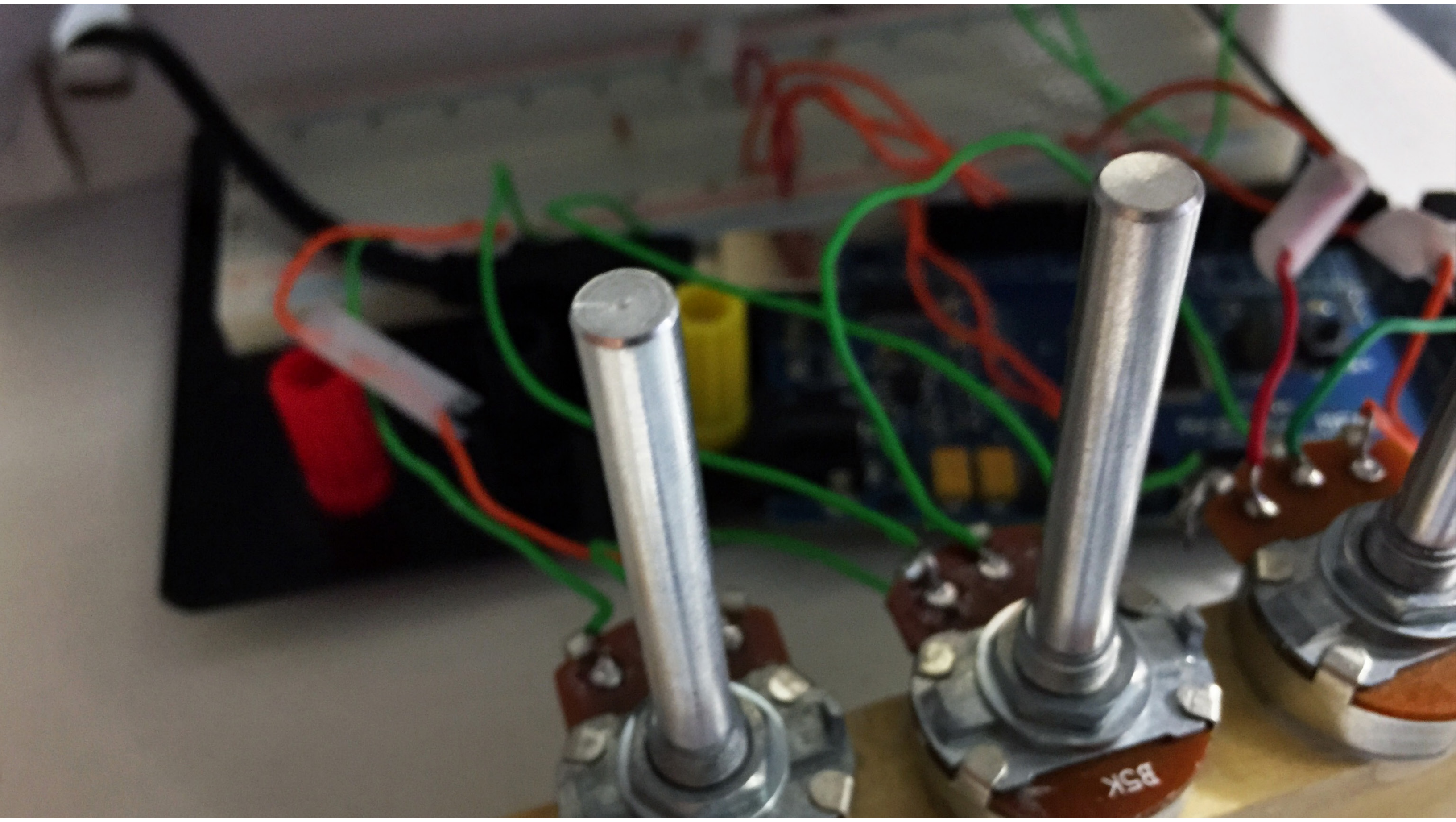
the electronic component to this project was housed on an arduino mega 1280 combined with a breadboard. a cathode rgb led was used as the light source and was connected to three separate pulse width modulation (pwm) pins on the arduino. a button was also attached to the breadboard, providing a positive digital signal when pressed. special code had to be used to ensure that the button would not give false positives (debouncing).

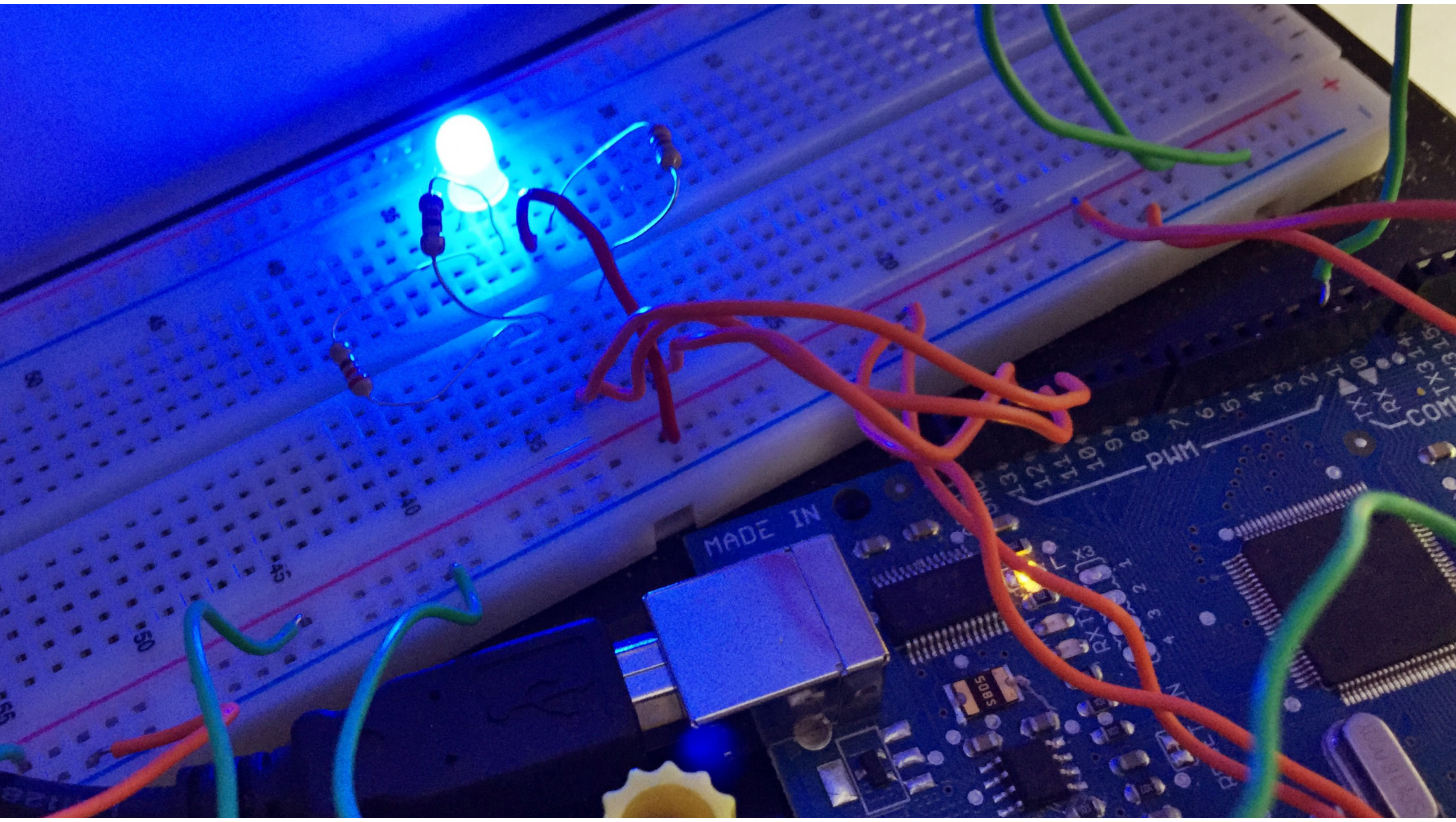
also attached to the arduino were three 5k-ohm linear taper potentiometers. these provided analog input on a scale of 0-255 for each of the rgb values. the arduino would read in these inputs and update the colors of the led accordingly.

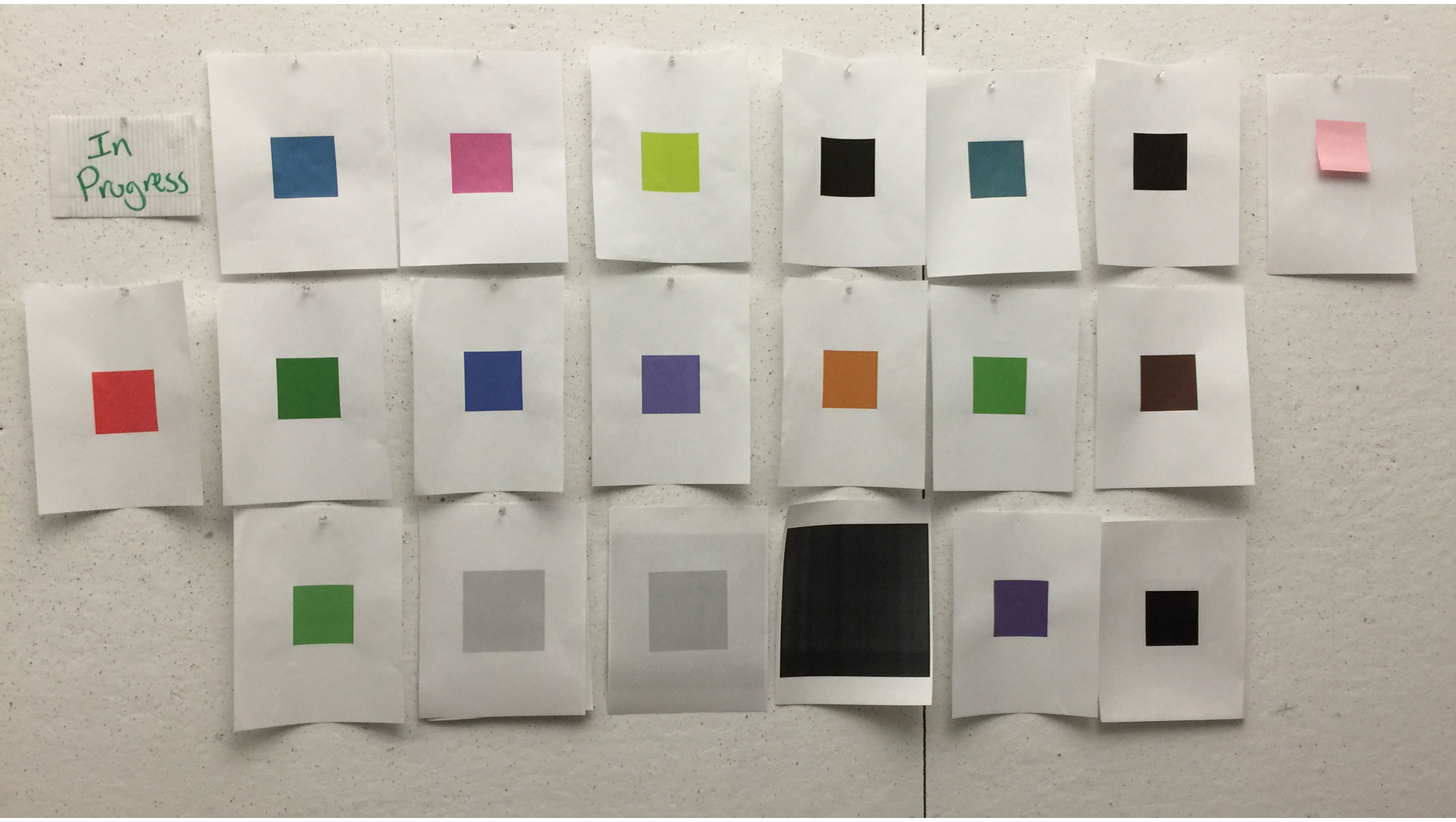
when the button is pressed, the arduino outputs a comma separated value (csv) of the rgb values to the serial at 9600 baud. a python script reads in these serial values and produces a bitmap image at 72 dpi that is 3x3 inches in size of that specific rgb color.

the script then sends the image to the print queue of the epson printer which in turn prints it out, converting the rgb color to cmyk in the process. the final product is two 3x3 inch squares of colors that theoretically should be the same, but in fact are not.

the arduino and all the electronic components are housed in a white box, so that the only things that the user can see are the “pixel,” knobs, and button. this idea of encapsulation is one that carries over from software engineering, where the actual implementation is hidden from the user.


B5K


MADE IN
PWM
COMM
TX
RX
L


In Progress

http://vimeo.com/ericyli/rgbcmyk

```
/* RGB to CMYK Eric Li */ int rLed = 13; // Red
LED Pin int gLed = 11; // Green LED Pin int
bLed = 12; // Blue LED Pin int rVal, gVal, bVal;
// Values of RGB int rPot = 0; // Analog 0 - Red
Pot int gPot = 3; // Analog 3 - Green Pot int
bPot = 7; // Analog 7 - Blue Pot int buttonPin
= 2; int buttonState; // the current reading from
the input pin int lastButtonState = LOW; // the
previous reading from the input pin long lastDe-
bounceTime = 0; // the last time the output pin
was toggled long debounceDelay = 50; // the
debounce time; increase if the output flickers
// the setup routine runs once when you press
reset: void setup() { // declare pin 9 to be an
output: pinMode(rLed, OUTPUT); pinMode(-
gLed, OUTPUT); pinMode(bLed, OUTPUT); pin-
Mode(buttonPin, INPUT); pinMode(rPot, INPUT);
pinMode(gPot, INPUT); pinMode(code, INPUT);
rVal = 0; gVal = 0; bVal = 0; Serial.begin(9600);
// Serial.write("Setup Done\n"); } // the loop rou-
tine runs over and over again forever: void loop()
```

rgbcmyk.ino

```
/*
RGB to CMYK
Eric Li
 */

int rLed = 13; // Red LED Pin
int gLed = 11; // Green LED Pin
int bLed = 12; // Blue LED Pin
int rVal, gVal, bVal; // Values of RGB

int rPot = 0; // Analog 0 - Red Pot
int gPot = 3; // Analog 3 - Green Pot
int bPot = 7; // Analog 7 - Blue Pot

int buttonPin = 2;

int buttonState;             // the current reading from the input pin
int lastButtonState = LOW;   // the previous reading from the input pin

long lastDebounceTime = 0;  // the last time the output pin was toggled
long debounceDelay = 50;    // the debounce time; increase if the output flickers

// the setup routine runs once when you press reset:
void setup()  {
  // declare pin 9 to be an output:
  pinMode(rLed, OUTPUT);
  pinMode(gLed, OUTPUT);
  pinMode(bLed, OUTPUT);
  pinMode(buttonPin, INPUT);

  pinMode(rPot, INPUT);
  pinMode(gPot, INPUT);
  pinMode(bPot, INPUT);

  rVal = 0;
  gVal = 0;
  bVal = 0;

  Serial.begin(9600);
// Serial.write("Setup Done\n");
}

// the loop routine runs over and over again forever:
void loop()  {

  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin);
  rVal = analogRead(rPot)/4;
  gVal = analogRead(gPot)/4;
  bVal = analogRead(bPot)/4;

  // set the brightness of pin 9:
  analogWrite(rLed, 255-rVal);
  analogWrite(gLed, 255-gVal);
  analogWrite(bLed, 255-bVal);


  // If the switch changed, due to noise or pressing:
    if (reading != lastButtonState) {
      // reset the debouncing timer
      lastDebounceTime = millis();
    }

    if ((millis() - lastDebounceTime) > debounceDelay) {

      if (reading != buttonState) {
        buttonState = reading;

        // only toggle the LED if the new button state is HIGH
        if (buttonState == HIGH) {
          Serial.print(rVal);
          Serial.print(",");
          Serial.print(gVal);
          Serial.print(",");
          Serial.print(bVal);
          Serial.print("\n");
        }
      }
    }

    // save the reading.  Next time through the loop,
    // it'll be the lastButtonState:
    lastButtonState = reading;

}
```

setup.py

```
#!/usr/bin/env python

from setuptools import setup

setup(name='ledhost',
      version='0.1',
      description='Looks for rgb values over serial, prints out a cmyk image based off that',
      author='Eric Li',
      author_email='eyli@princeton.edu',
      install_requires=[
          'pyserial',
          'pillow',
      ])
```

ledhost.py

```
#!/usr/bin/env python

import argparse
import subprocess
import serial
from PIL import Image, ImageDraw

# changeme for different dpi
DPI = 72
PRINTER_NAME = ‘Epson_4800_303__IP_’


def parse_args():
    p = argparse.ArgumentParser(description=’Listen for RGB vals over serial, and print a
cmyk picture’)
    p.add_argument(‘port’, type=str)
    p.add_argument(‘--baudrate’, type=int, default=9600)
    return p.parse_args()


def draw_image(rgb):
    return Image.new(‘RGB’, (DPI*3,DPI*3), tuple(rgb))


def print_img(img):
    img.save(‘tmp.png’)
    subprocess.check_call([‘lpr’, ‘-P’, PRINTER_NAME, ‘tmp.png’])


# uncomment to debug without serial port
#class FakeSerial(object):
#    def __init__(self, port, baudrate):
#        pass
#    def __enter__(self):
#        return self
#    def __exit__(self, exc_type, exc_value, traceback):
#        pass
#    def readline(self):
#        import time
#        import random
#        time.sleep(2)
#        return ‘,’.join([str(random.randint(0,255)) for _ in range(3)]) + ‘\n’
#serial.Serial = FakeSerial


if __name__ == ‘__main__’:
    args = parse_args()
    with serial.Serial(args.port, args.baudrate) as ser:
        print(‘opened port {} with {} baud’.format(args.port, args.baudrate))
        while True:
            readline = ser.readline()
            try:
                rgb = [int(x) for x in readline.strip().split(‘,’)]
            except Exception as e:
                print(‘Failed to parse line from arduino: {}’.format(e))
            else:
                print(‘read rgb value: {}’.format(rgb))
                print_img(draw_image(rgb))
                print(‘printed image’)

    print(‘finished’)
```

randomgrid.pde

```
/* Random grid of colors that fill an 8.5 x 11 page */
/* Eric Li */
double L = 8.5; /* Length */
double W = 11; /* Width */
int DPI = 72; /* DPI */

int n = 40; /* Dimensions of square */

int r,g,b; /* R G B values */

boolean animate = false; /* true to animate */

/* Sets up the canvas */
void setup() {
  size((int)(L*DPI),(int)(W*DPI));
  background(255);

   /* X, Y coord system */
  for (int x = 15; x < width-n; x+=n+n/2.)
  {
    for (int y = 15; y < height-n; y+=n+n/2.)
    {
      stroke(255);
      r = (int)random(255);
      g = (int)random(255);
      b = (int)random(255);
      fill(r,g,b);
      rect(x,y,n,n);
    }
  }

  save("output.png");
}

void draw()
{
  if (animate)
  {
    frameRate(25);
     /* X, Y coord system */
    for (int x = 15; x < width-n; x+=n+n/2.)
    {
      for (int y = 15; y < height-n; y+=n+n/2.)
      {
        stroke(255);
        r = (int)random(255);
        g = (int)random(255);
        b = (int)random(255);
        fill(r,g,b);
        rect(x,y,n,n);
      }
    }
  }
}
```