Red/green Colorblindness Simulation: Creating a Colorblind Friendly Environment

Jenny Kang

A senior capstone project report submitted to the faculty of

Brigham Young University

In partial fulfillment of the requirements for the degree of

Bachelor of Science

David Allred, Advisor

Department of Physics and Astronomy

Brigham Young University

April 14, 2021

Abstract

Red/green Colorblindness Simulation: Creating a Colorblind Friendly Environment

Jenny Kang

Department of Physics and Astronomy, BYU

Bachelor of Science

A revolutionary solution for colorblindness was devised by Dr. Don McPherson who invented color corrective glasses for the colorblind to see colors in 2012. We are hoping to create greater empathy for colorblind individuals and to assist in creating a more colorblind-accessible environment. Hence, converse to the colorblind corrective glasses, in this project we assumed a person with normal color vision would perceive the world as a red-green colorblind person if only 540 nm to 570 nm of wavelengths transmitted through by the dichroic filters. We report on the creation of googles that causes people wearing them to lose red-green discrimination. Our goal is to allow non-colorblind individuals to experience red/green colorblindness. Seventeen volunteers completed Ishihara colorblind tests, sixteen of them with healthy color vision were diagnosed as severely red/green colorblind while taking the test with colorblind goggles on.

Acknowledgements

I would like to express my deepest appreciation to my advisor, Dr. David Allred, who has the attitude and the substance of a genius, Professor Sandberg, who patiently guided me, and my partner, Zach Westhoff, who helped turn a project idea into reality. In addition to the challenge of finding the appropriate filters in a short amount of time, this dissertation would not have been possible without their guidance and persistent help.

I would also like to thank my partner, Zach Westhoff, and teaching assistant, Nick Porter. Both continually and convincingly conveyed a spirit of adventure towards this research and an excitement for finding solutions from obstacles. In addition, I would like to express appreciation to all the volunteers who helped testing out the colorblind goggles.

Lastly, I would like to thank Dr. Steve Turley and Dr. Karine Chesnel, who instructed me on writing skills and gave me feedback so I can make this thesis more polished and complete. I want to also thank Brigham Young University for providing me the opportunity to gain a wonderful education and the resources which not only to helped me obtain a higher level of education, but also helped me build my character.

Table of Contents

Chapter 1: Introduction

1.1 Motivation and Background

Approximately 300 million people worldwide suffer from color blindness—roughly 1 in 10 men in most Caucasian populations [1]. The struggles of colorblind people might not be as obvious as other forms of disability; nevertheless, colorblind individuals can face many challenges in their day-to-day life. These challenges may manifest in routine activities including preparing and cooking for meals, charging electronics which use red/green indicator lights, operating a vehicle, and selecting clothing. Moreover, some people may not notice sunburns on their children or discoloring in their stool due to diseases [2]. There are several types and various levels of severity of color blindness; because the most common form is red/green color blindness, our project will focus on this type.

Human eyes have photoreceptors on the retina: color detectors called cones. Each cone contains a type of opsin protein, and each protein responds to different wavelengths of light. S-opsin is sensitive at short wavelengths (blue, ~430 nm), M-opsin at medium wavelengths (green, ~530 nm), and L-opsin at long wavelengths (red, ~560 nm) [3]. When the energy carried by light transfers to cones, the opsin protein changes shape, sending signals to the brain. Red/green colorblindness is caused by defects in M-opsin, shifting its sensitivity closer to L-opsin. Therefore, the brain receives signals from both M-opsin and L-opsin when viewing certain colors instead of just red or green, see Fig.1.

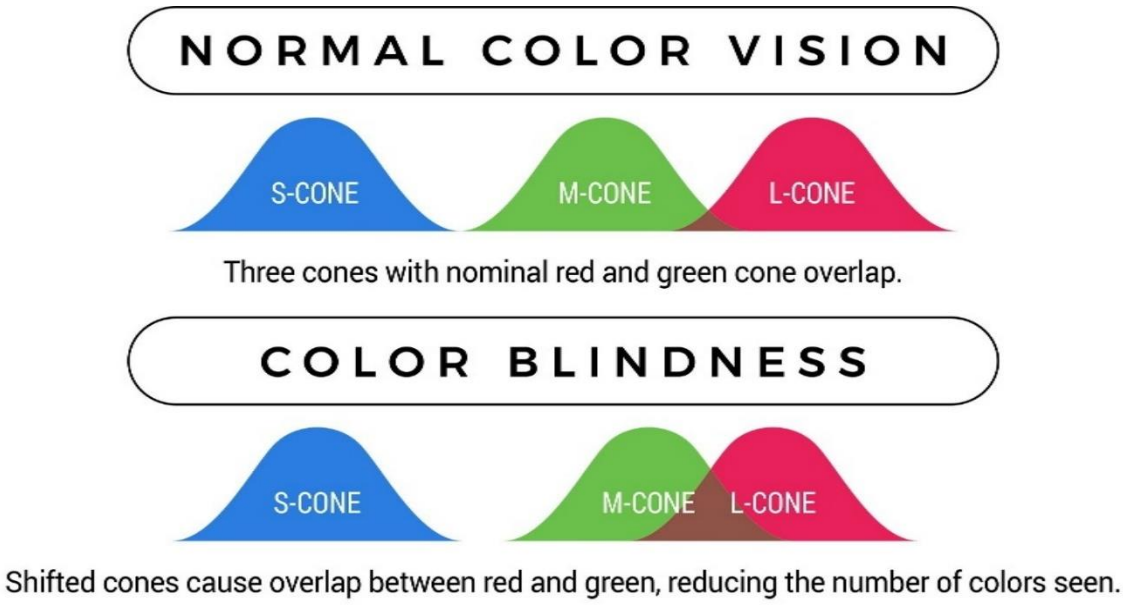

Figure 1: Color cone comparison between normal vision and red/green colorblindness
Source: Enchroma.Com, https://enchroma.com/pages/what-is-color-blindness

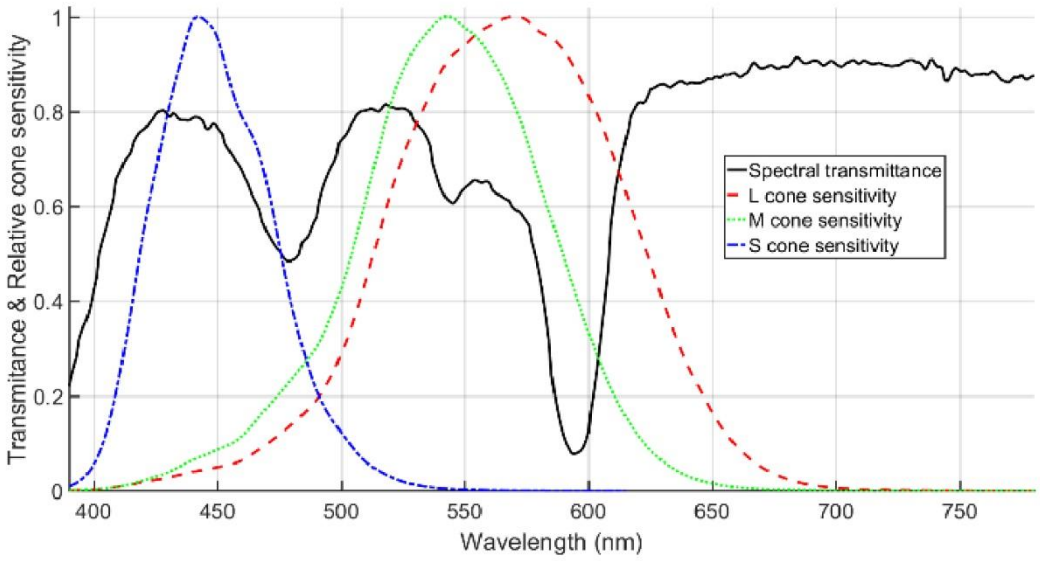

Figure 2: The black line is the transmittance spectrum of colorblind correcting lenses created by Dr. McPherson

Source: H. S. Fairman, M. H. Brill, and H. Hemmendinger, "How the CIE 1931 color-matching functions were derived from Wright-Guild data," Color Res. Appl. 22(1), 11–23 (1997).

Hence, one solution to the problem of simultaneous activation of two different color cones is to filter out the wavelengths that the cones cannot differentially respond to by adding optical filters, which control the transmittance of the light at certain wavelengths. Thus, the brain will not be confused by two signals and only produce one color at a time. See Fig. 2. These wavelengths correspond mostly to the yellow to yellow-green portion of the spectrum.

1.2 Objective of the Research

Opposite of colorblind correcting glasses, in this project we assumed that if we used filters to block out the wavelengths that M-opsin and L-opsin can distinctly respond to, leaving only the wavelengths that confused the brain when seeing red or green together, a person with healthy color vision would perceive the world as a red/green colorblind person does. **In short, we aimed to create classes that simulate red/green colorblindness in a user with normal vision.** By doing so, we hoped to create greater empathy for colorblind individuals and to assist in creating a more colorblind- environment.

Chapter 2: Methods

In this chapter we are going to unfold the mechanisms and process of creating a goggle which cause a normal vision person to become red/green colorblind.

2.1 Dichroic Filters

2.1.1 Characteristics of Bandstop and Shortpass Filters

Dichroic filters were used because the transmission as a function of wavelength range could be modulated by the thickness and the layers of coatings. In addition, tilting the filters changes the position of transmission bands. As light travels through coatings with different indices, optical interferences occur and phase reflections are produced [4]. Some resonate constructively, others cancel out destructively; only certain wavelengths can transmit through while the unwanted wavelengths are reflected. This was a desirable characteristic for our experiment, since we needed to tune the filters so that only a narrowband, centered on the the target wavelength would go through filters [5]. Hence, shortpass filters have a role to play. They allow shorter wavelengths to transmit through while reflecting all the longer wavelengths. Contrarily, bandpass filters are designed to allow longer wavelengths to transmit through and to reflect all the shorter wavelengths.

The extreme angle sensitivity of dichroic filters was a desirable characteristic to design our goggles. As dichroic filters were tilted away from the normal, the transmission spectrum is shifted to the shorter wavelengths. The shifted wavelength λ_θ could be calculated through following the formula:

$$\lambda_\theta = \lambda_0 \sqrt{1 - \left(\frac{\sin\theta}{neff}\right)^2} \qquad (1)$$

Where λ_0 is the unshifted wavelength at normal incidence, θ is the angle of incidence and n_{eff} is the effective refractive index inside the filter.

We used a spectrometer to characterize filters we obtained for our project. A spectrometer is a device to decompose wavelengths of light. In our case, the intensity of the spectra is converted into counts via an analog-to-digital converter (ADC). We integrated 100 data points over 50 ms on average. To get accurate measurements, we first collected the data of ambient light and subtracted the ADC counts of ambient light for the following four measurements, bandstop with angle of

incidence (AOI) of 0, 15, 30, and 45 degrees. AOI alludes to the tilt of the optical filter with respect to the incident light. Zero AOI is defined from the normal incidence. In Fig. 3 below, the collected data of light transmission with various AOIs using the spectrometer are shown. In the next section we will discuss which angle would capture the best target wavelengths to simulate red-green colorblindness.

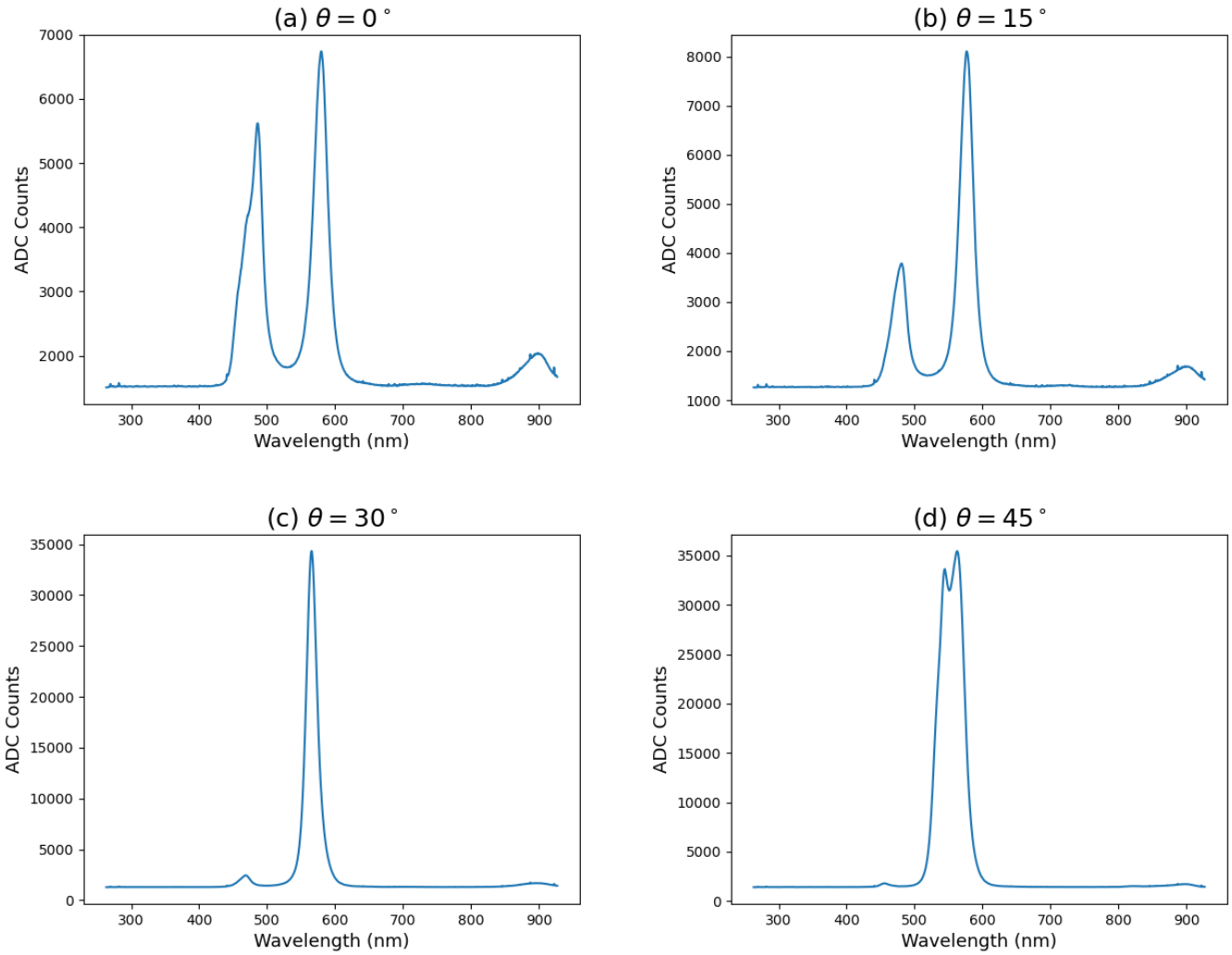

Figure 3: Bandstop filter tilted at different angles (a)AOI at 0 degrees (b) AOI at 15 degrees (c) AOI at 30 degrees (d) AOI at 45 degrees.

2.1.2 Target Wavelength Range

To cause color blindness for a person with normal vision, we aimed to block out the shorter wavelengths M-opsin responded to and the longer wavelengths L-opsin responds to while still retaining the range 535 nm - 575 nm. M-opsin and L-opsin were most likely to overlap and be activated simultaneously within this range. A combination of two optical filters attained the target range:

- Filter 1: A dichroic shortpass filter (570FDS) reduced the transmittance of the wavelengths that were longer than 570 nm.
- Filter 2: A dichroic bandstop filter (537 FDN) at AIO of 45 degrees cut off the transmittance wavelengths that were between 440 nm to 540 nm, which reduce most of the transmittance of green and still allow blue to go through.

Combining these two filters, we hoped to see the color spectrum shown in Fig. 4, red and green should turn into a combination of red and green, yellowish brown.

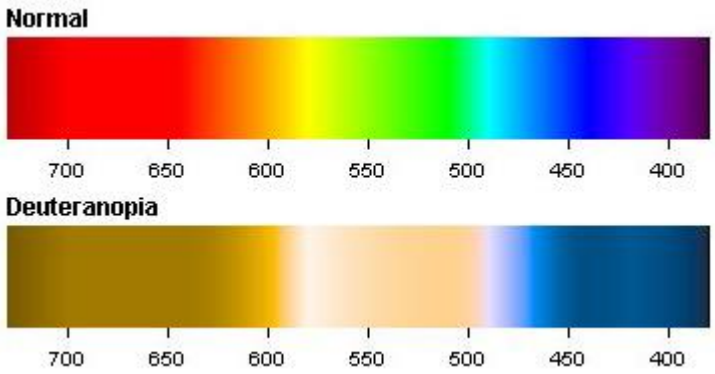

Figure 4: Normal and Deuteranopia (also called green-blind) colorblind spectrum
Source: Color-Blindness.Com, https://www.color-blindness.com/deuteranopia-red-green-color-blindness/

After finding the desirable target wavelength, we used spectrometry to measure transmittance. We stacked two filters together: the bandpass filter and the shortpass filter. The bandstop filter was tilted at 45 degrees and the shortpass filter was at normal incidence. The transmittance of the combined filters is shown in Fig. 5.

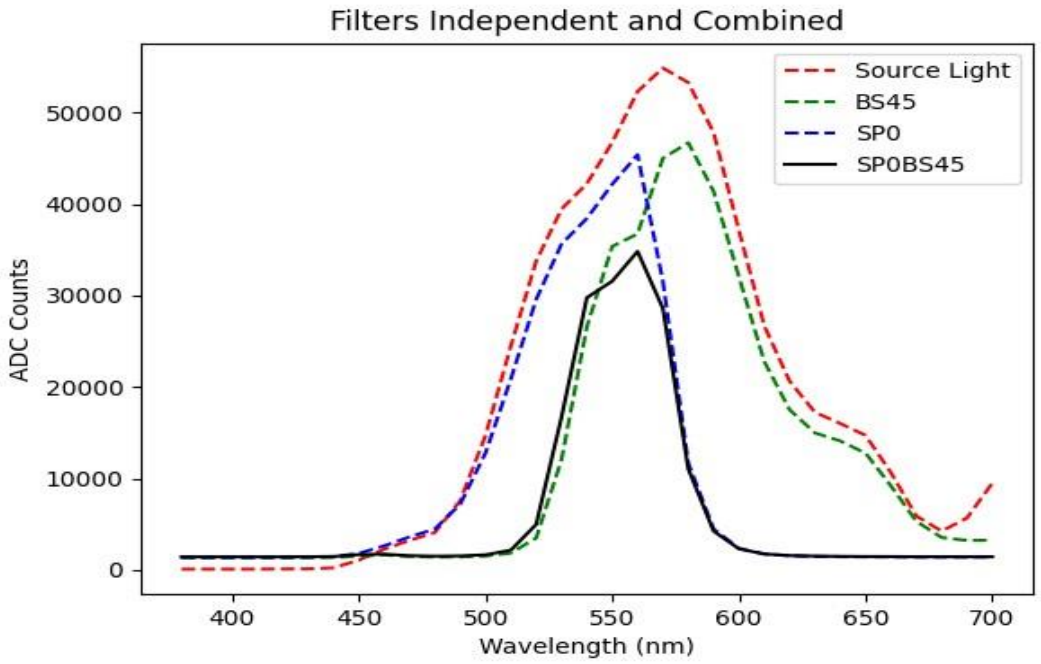

Figure 5: Target wavelength through combine filters

The black line shows the range of wavelengths which will transmit through the filters is centered on 550 nm. Theoretically, the range is the band of wavelengths for which the brain cannot differentiate red and green colors well; consequently, people with normal color vision will become red/green colorblind. As we saw the experimental data line up with our expectation, we saw the probability of achieving the research's aim increase.

2.2 Computational Simulation

Make computer simulation of red/green colorblindness with collected data in python.

2.2.1 Computational Techniques and Algorithms

To obtain the principal wavelength by converting an RGB image (red, green, and blue) to HSV (hue, saturation, and value), we followed the formula outlined in the previous entry. After obtaining that wavelength, we calculated the ratio of L- and M-cone sensitivities multiplied by the transmission of combined filters at that wavelength. We reasoned that it might be a decent approximation of the impact of our filters to ensure that the ratio of R and G (red and green) values were equal to

that ratio. That is, if ~550 nm wavelength is supposed to evenly trigger red and green based on the adjusted sensitivities, we would expect to see an even amount of R and G. Essentially, we used R, G, and B as proxies of the triggering of L, M, and S cones, respectively.

Here is the process that results in R/G = L/M without dramatically reducing brightness of pixel:

1. Create HSV array from RGB array for each pixel
2. Obtain principal wavelength from H of HSV array
3. Calculate L and M cone sensitivity at that wavelength
4. Calculate sqrt(L/M)
5. Average R and G
6. Set R equal to (Average) * sqrt(L/M)
7. Set G equal to (Average) / sqrt(L/M)

Performing this operation on a rainbow, produced better results than we had initially, though they were still not perfect. Note, for example, the obvious discontinuity in Fig. 6 below.

Figure 6: Simulation of color spectrum of anticipated red/green colorblindness

2.2.2 Challenges in Computational Simulation

In the first place, it is a difficult problem, as real-life colors are not monochromatic, and there is a good amount of biology tied up in how we perceive colors. Additionally, it is quite difficult to determine how to modify a pixel as if seen through a filter, which has broadband effects that change the character of image perceived, when the only information that can possibly be gained about the pixel is its dominant wavelength. For example, if we looked at a pure red object through our filter, we would not expect it to be pitch black. Thus, the simple model above would not work. We determined that the biggest flaw in the way we had been approaching this program was that we had completely ignored the cones of the eye, L for red, M for green, and S for blue. It was beyond our scope to make an accurate complex computer simulation of human eyes; therefore, a clinical experiment was carried out.

2.3 In Clinical Experiment

2.3.1 Testing Method and Procedure

We invited 17 people from a senior lab class at BYU to participate in the Ishihara red-green colorblindness test with and without the colorblind goggles we had created. This test contains colored digits which test takers view in a petri dish of colored splotches. Normal-visioned people see 74 in Fig 6 (a), while mildly red-green colorblind people often see 21, and severely colorblind people cannot make anything out.

This is the procedure of the clinical test is: first, we had the participant wear the goggles for a couple minutes, so their brain could adjust to them. Then, we led the volunteer into a dark room which we administered the test. We darkened the room to mitigate the effects of ambient light, which would otherwise cause reflections on the filters and make it harder to see the test through the goggles. We asked the participants to hold the goggles straight and view the test, shown on a computer screen, straight-on. Keeping the viewing angle is necessary because the transmission of the filters changes as a function of the angle of incident light. We stepped through the twelve images of the selected test and then recorded the responses of the goggle wearing participants. After taking the test with the goggles, the participant would take the same test again without the goggles in order to control for the possibility that some participants would be unable to resolve the numbers even without the goggles.

Example images from the test are shown in Fig. 6 below. Fig. 6 (a) and (b) are taken with the same phone camera; the image on the left is direct, and the image on the right is through our goggles. Finished colorblind goggles are shown in Fig. 7. Since we stacked two lenses together and the bandpass filter is tilted 45 degrees, the goggles are shaped like binoculars.

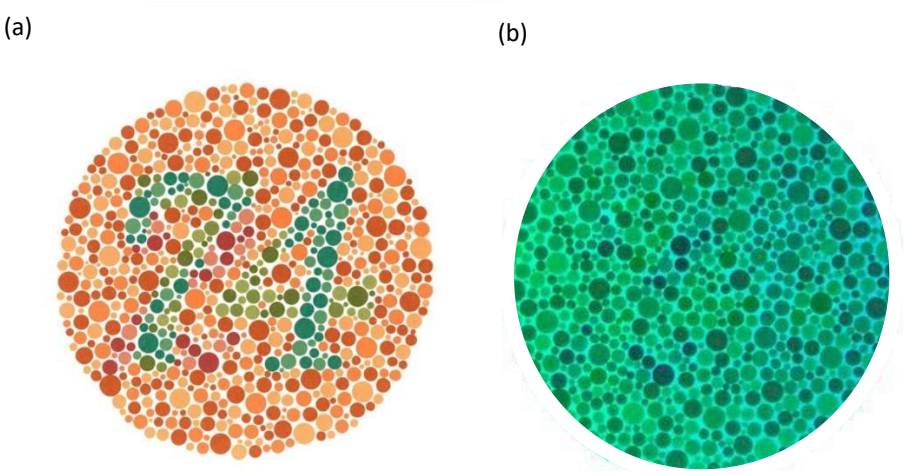

Figure 6: Pictures of Ishihara test (a) without goggles on (b) with goggles on

Figure 7: Picture of the final product: colorblindness goggles

Chapter 3: Results and Conclusions

3.1 Experimental Results

3.1.1 Data Analysis

From the Ishihara test result of 17 volunteers, we found that almost 100% of the responses indicated the test takers had sever colorblindness while wearing the colorblindness goggles, see Fig. 8. We had successfully created goggles that tricked the

brain and induced colorblindness in people who were not affected by any color vision impairments. Even with a small data sample, the test results presented consistent and strong evidence in support of our conclusion.

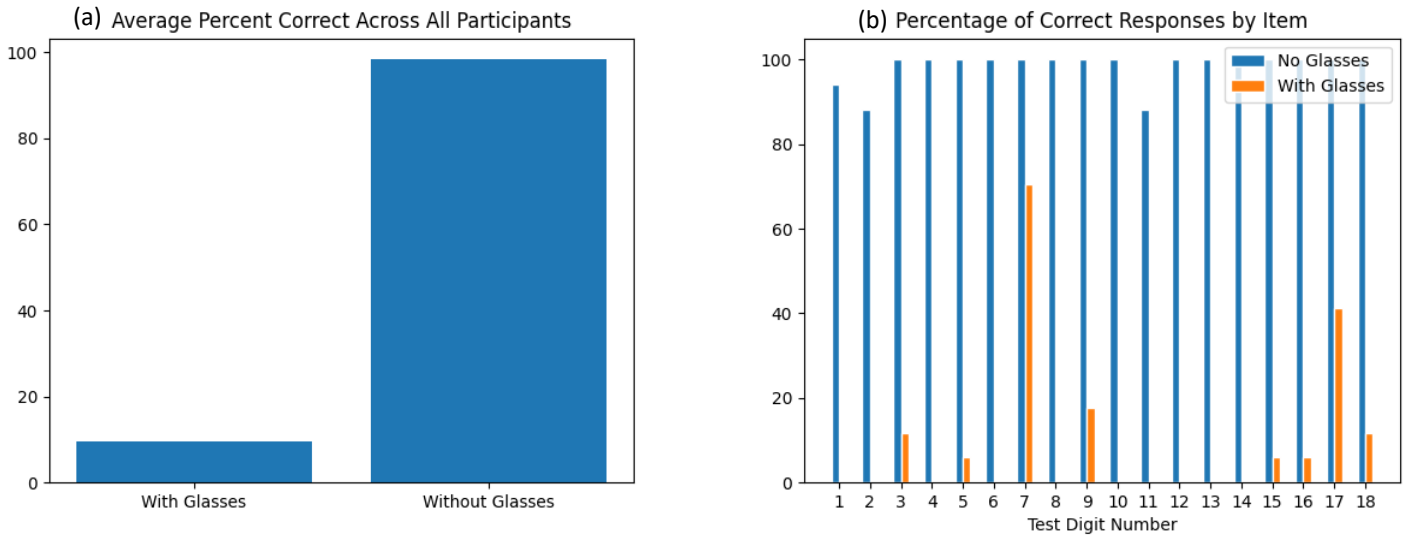

Figure 8. Ishihara test results (a) Average percent correct across all participants
(b) Percentage of correct responses by each test item

3.1.2 Highlighted Result: Cory

One of the volunteers, Cory, was unaware of his colorblindness until he took the Ishihara test without the glasses. We found out that he responded to a certain question incorrectly in the same way (that is, reporting 21 instead of 74) as many participants did while wearing the glasses. People with normal vision would see 74 in Figure 6 (a) in chapter 2, while mildly red-green deficiencies often see 21 [6]. Cory's test result and his responses reinforced our experiment's conclusion and proved that the colorblindness goggles we had created provide the desired result.

3.2 Conclusions

We had proposed that by only letting through the undistinguishable wavelength, the overlapping area of M-opsin and L-opsin activating wavelengths, people with normal color vision would experience see a red-green colorblind world. In this paper we showed that by stacking two dichroic filters, the shortpass filter (570 FDN) and the bandpass filter (537 FDN) at 45 degrees AOI, wavelengths that are between 540 nm to 570 nm would transmit through. The M-opsin and L- opsin would be activated simultaneously to then simulate red/green colorblindness. With these glasses, people with normal vision can gain a greater

Empathy for those who affected by colorblindness. This also allows engineers to create a safer and more mindful environment for those with color deficiencies.

Bibliography

[1] Enchroma.Com, https://enchroma.com/blogs/beyond-color/interesting-facts-about-color-blindness

[2] Living with Colour Vision Deficiency, https:/ /www.colourblindawareness.org/colour-blindness/living-with-colour-vision-deficiency/

[3] Purves D, Augustine GJ, Fitzpatrick D, et al., editors. Neuroscience. 2nd edition. Sunderland (MA): Sinauer Associates; 2001. Cones and Color Vision, https://www.ncbi.nlm.nih.gov/books/NBK11059/l

[4] What is a Dichroic Filter, https://abrisatechnologies.com/2014/10/what-is-a-dichroic-filter/

[5] Optical Filters | Edmund Optics, https://www.edmundoptics.com/knowledge-center/application-notes/optics/optical-filters/

[6] 1255 Ishihara 24 Plate Instruction, http://www.dfisica.ubi.pt/~hgil/p.v.2/Ishihara/Ishihara.24.Plate.Instructions.I.pdf

Appendix

Computer simulation code

```
# This is a sample Python script.

# Press ^R to execute it or replace it with your code.
# Press Double ⇧ to search everywhere for classes, files, tool windows, actions, and settings.

import numpy as np
import matplotlib.pyplot as plt
# # https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html
from scipy.interpolate import interp1d

class Optical:
    wavelengths = []
    transmittances = []
    pairing = []

# Create list of wavelengths, every ten nm, across visible spectrum
# Taking visible spectrum to be 380 to 700 nm
wavelengthArray = np.arange(380, 710, 10).tolist()

# For each Optical object
# Create a list of transmission percentages
# This is done by visual inspection, yuck
# Make sure they line up with the correct index of wavelength

shortPass = Optical()
shortPassTransmission = [0.40, 0.75, 0.91, 0.94, 0.93, 0.93, 0.95, 0.93, 0.94, 0.95, 0.93, 0.91, 0.93, 0.94, 0.93, 0.95,
0.96, 0.96, 0.90, 0.70, 0.15, 0.08, 0.03, 0.02, 0.01, 0, 0, 0, 0, 0, 0, 0, 0]

# Helps make it easier to not lose your place when looking at graphs
for i in range(0, len(wavelengthArray) - len(shortPassTransmission)):
    wavelengthArray.pop()
merged = tuple(zip(wavelengthArray, shortPassTransmission))
# print(merged)

shortPass.wavelengths = wavelengthArray
shortPass.transmittances = shortPassTransmission
shortPass.pairing = tuple(zip(wavelengthArray, shortPassTransmission))

# BANDSTOP
bandStop = Optical()

#wavelengthArray = np.arange(380, 710, 10).tolist()

bandStopTransmission = [0.75, 0.82, 0.89, 0.90, 0.90, 0.89, 0.91, 0.91, 0.91, 0.93, 0.85, 0.20, 0.05, 0.03, 0.01, 0.01,
0.01, 0.02, 0.03, 0.10, 0.40, 0.92, 0.94, 0.94, 0.92, 0.91, 0.91, 0.93, 0.93, 0.92, 0.91, 0.91, 0.91]

#Helps make it easier to not lose your place when looking at graphs
for i in range(0, len(wavelengthArray) - len(bandStopTransmission)):
    wavelengthArray.pop()
merged = tuple(zip(wavelengthArray, bandStopTransmission))
# print(merged)
```

```
bandStop.wavelengths = wavelengthArray
bandStop.transmittances = bandStopTransmission
bandStop.pairing = tuple(zip(wavelengthArray, bandStopTransmission))


# COMBINED
combinedFilters = Optical()
combinedFilters.wavelengths = wavelengthArray
combinedFilters.transmittances = []
for i in range(0, len(wavelengthArray)):
    combinedFilters.transmittances.append(abs(shortPass.transmittances[i] * bandStop.transmittances[i]))
merged = tuple(zip(combinedFilters.wavelengths, combinedFilters.transmittances))
# print(merged)

# BANDSTOP 45, angle of incidence
bs45 = Optical()
bs45Transmission = [0.65, 0.75, 0.71, 0.67, 0.62, 0.50, 0.30, 0.18, 0.10, 0.03, 0.02, 0.01, 0.01, 0.01, 0.02, 0.07, 0.60,
0.70, 0.58, 0.70, 0.85, 0.82, 0.83, 0.84, 0.83, 0.82, 0.81, 0.72, 0.61, 0.43, 0.25, 0.15, 0.09]

for i in range(0, len(wavelengthArray) - len(bs45Transmission)):
    wavelengthArray.pop()
merged = tuple(zip(wavelengthArray, bs45Transmission))
# print(merged)

bs45.wavelengths = wavelengthArray
bs45.transmittances = bs45Transmission
bs45.pairing = tuple(zip(wavelengthArray, bs45Transmission))

sp0bs45 = Optical()
sp0bs45.wavelengths = wavelengthArray
sp0bs45.transmittances = []

for i in range(0, len(wavelengthArray)):
    sp0bs45.transmittances.append(abs(shortPass.transmittances[i] * bs45.transmittances[i]))
merged = tuple(zip(sp0bs45.wavelengths, sp0bs45.transmittances))
print(merged)


spf = interp1d(shortPass.wavelengths, shortPass.transmittances, kind='cubic')
plt.plot(shortPass.wavelengths, shortPass.transmittances, 'o', shortPass.wavelengths, spf(shortPass.wavelengths), '--')
plt.title("Short Pass at 0 AOI")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Transmission")
plt.show()

bsf = interp1d(bandStop.wavelengths, bandStop.transmittances, kind='cubic')
plt.plot(bandStop.wavelengths, bandStop.transmittances, 'o', bandStop.wavelengths, bsf(bandStop.wavelengths), '--')
plt.title("Band Stop at 0 AOI")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Transmission")
plt.show()

cf = interp1d(combinedFilters.wavelengths, combinedFilters.transmittances, kind='cubic')
plt.plot(combinedFilters.wavelengths, combinedFilters.transmittances, 'o', combinedFilters.wavelengths,
cf(combinedFilters.wavelengths), '--')
```

```
plt.title("Combined Filters, 0 AOI")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Transmission")
plt.show()

bs45f = interp1d(bs45.wavelengths, bs45.transmittances, kind='cubic')
plt.plot(bs45.wavelengths, bs45.transmittances, 'o', bs45.wavelengths, bs45f(bs45.wavelengths), '--')
plt.title("BandStop at 45 Degrees AOI")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Transmission")
plt.show()

sp0bs45f = interp1d(sp0bs45.wavelengths, sp0bs45.transmittances, kind='cubic')
plt.plot(sp0bs45.wavelengths, sp0bs45.transmittances, 'o', sp0bs45.wavelengths, sp0bs45f(sp0bs45.wavelengths), '--
')
plt.title("Combined Filters, SP 0, BS 45 AOI")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Transmission")
plt.savefig("ShortpassBandstop45")
plt.show()

# Now I will make Optical() objects for each of the cones in the eye
# In this case "transmittance" is actually sensitivity
# Values taken from the graph available at https://en.wikipedia.org/wiki/Spectral_sensitivity
# Values are normalized
scone = Optical();
scone.transmittances = []
sconeSensitivities = [0.01, 0.03, 0.06, 0.18, 0.55, 0.85, 1, 0.9, 0.75, 0.45, 0.3, 0.18, 0.12, 0.08, 0.06, 0.03, 0.02, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# To help with inputting values
for i in range(0, len(wavelengthArray) - len(sconeSensitivities)):
    wavelengthArray.pop()
merged = tuple(zip(wavelengthArray, sconeSensitivities))
print(merged)

scone.wavelengths = wavelengthArray;
scone.transmittances = sconeSensitivities

sconef = interp1d(scone.wavelengths, scone.transmittances, kind='cubic')
plt.plot(scone.wavelengths, scone.transmittances, 'o', scone.wavelengths, sconef(scone.wavelengths), '--')
plt.title("S-Cone Sensitivity")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Sensitivity")
plt.savefig("S-Cone")
plt.show()


mcone = Optical();
mcone.transmittances = []
mconeSensitivities = [0, 0, 0.01, 0.02, 0.04, 0.07, 0.09, 0.1, 0.12, 0.16, 0.25, 0.35, 0.55, 0.75, 0.97, 1, 0.98, 0.9, 0.85,
0.75, 0.60, 0.45, 0.35, 0.25, 0.17, 0.09, 0.07, 0.03, 0.02, 0.01, 0, 0, 0]

# To help with inputting values
for i in range(0, len(wavelengthArray) - len(mconeSensitivities)):
    wavelengthArray.pop()
```

```
merged = tuple(zip(wavelengthArray, mconeSensitivities))
print(merged)

mcone.wavelengths = wavelengthArray;
mcone.transmittances = mconeSensitivities

mconef = interp1d(mcone.wavelengths, mcone.transmittances, kind='cubic')
plt.plot(mcone.wavelengths, mcone.transmittances, 'o', mcone.wavelengths, mconef(mcone.wavelengths), '--')
plt.title("M-Cone Sensitivity")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Sensitivity")
plt.savefig("M-Cone")
plt.show()


lcone = Optical();
lcone.transmittances = []
lconeSensitivities = [0, 0, 0, 0, 0, 0.01, 0.02, 0.04, 0.06, 0.1, 0.14, 0.2, 0.3, 0.4, 0.55, 0.7, 0.83, 0.9, 0.99, 0.98, 0.94,
0.89, 0.81, 0.7, 0.6, 0.45, 0.28, 0.17, 0.1, 0.05, 0.02, 0.01, 0]

# To help with inputting values
for i in range(0, len(wavelengthArray) - len(lconeSensitivities)):
    wavelengthArray.pop()
merged = tuple(zip(wavelengthArray, lconeSensitivities))
print(merged)

lcone.wavelengths = wavelengthArray;
lcone.transmittances = lconeSensitivities

lconef = interp1d(lcone.wavelengths, lcone.transmittances, kind='cubic')
plt.plot(lcone.wavelengths, lcone.transmittances, 'o', lcone.wavelengths, lconef(lcone.wavelengths), '--')
plt.title("L-Cone Sensitivity")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Sensitivity")
plt.savefig("L-Cone")
plt.show()


plt.plot(wavelengthArray, sp0bs45f(wavelengthArray), 'k--', wavelengthArray, sconef(wavelengthArray), 'b',
wavelengthArray, mconef(wavelengthArray), 'g', wavelengthArray, lconef(wavelengthArray), 'r')
plt.title("Sensitivies by Cone")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Normalized Sensitivity")
plt.legend(['Transmission (BS45)', 'S-Cone', 'M-Cone', 'L-Cone'], loc='best')
plt.savefig("Sens_Transm")
plt.show()

plt.plot(wavelengthArray, sconef(wavelengthArray), 'b', wavelengthArray, mconef(wavelengthArray), 'g',
wavelengthArray, lconef(wavelengthArray), 'r')
plt.title("Sensitivities of the Optical Cones of the Human Eye")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Normalized Sensitivity")
plt.legend(['S-Cone', 'M-Cone', 'L-Cone'], loc='best')
plt.savefig("Sensitivities")
plt.show()
```

```
plt.plot(wavelengthArray, spf(wavelengthArray), 'b:', wavelengthArray, bs45f(wavelengthArray), 'y--',
wavelengthArray, sp0bs45f(wavelengthArray), 'g-.')
plt.title("Transmission of Chosen Filters")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Sensitivity")
plt.legend(['Shortpass', 'Bandstop at 45 Deg.', 'Combined'], loc='best')
plt.savefig("Transmission_of_filters")
plt.show()

plt.plot(wavelengthArray, sconef(wavelengthArray)*sp0bs45f(wavelengthArray), 'b', wavelengthArray,
mconef(wavelengthArray)*sp0bs45f(wavelengthArray), 'g', wavelengthArray,
lconef(wavelengthArray)*sp0bs45f(wavelengthArray), 'r')
plt.title("Sensitivities x Transmission")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Sensitivity")
plt.legend(['S-Cone', 'M-Cone', 'L-Cone'], loc='best')
plt.savefig("Sens_x_Transm")
plt.show()

import pandas as pd

cols = ['Pixel', 'Wavelength', 'Sum', 'Average']

# Ambient light data, subtract off from the average of other experimental
ambientlight = pd.read_csv('ambient-avg100_50ms.csv')
ambientlight.columns = cols

ambientlightf = interp1d(ambientlight['Wavelength'], ambientlight['Average'], kind='cubic')
plt.plot(ambientlight['Wavelength'], ambientlightf(ambientlight['Wavelength']))
plt.title("Ambient Light, No Filters")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
plt.savefig("AmbientLight")
plt.show()

# White light, experimental
whitelight = pd.read_csv('whitelight-avg100_50ms.csv')
whitelight.columns = cols
# Subtract off ambient light
whitelight['Average'] -= ambientlight['Average']
whitelightf = interp1d(whitelight['Wavelength'], whitelight['Average'], kind='cubic')
plt.plot(whitelight['Wavelength'], whitelightf(whitelight['Wavelength']))
plt.title("Light from Source, No Filters")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
plt.savefig("SourceLight")
plt.show()
#%%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
bandstopfar_0_45 = pd.read_csv('bandstopfar_0_45.csv')
cols = ['Pixel', 'Wavelength', 'Sum', 'Average']
bandstopfar_0_45.columns = cols
# Subtracting ambient light here makes graph blank; goes negative somewhere?
bandstopfar_0_45f = interp1d(bandstopfar_0_45['Wavelength'], bandstopfar_0_45['Average'], kind='cubic')
```

```
plt.plot(bandstopfar_0_45['Wavelength'], bandstopfar_0_45f(bandstopfar_0_45['Wavelength']))
plt.title("(d) $\u03B8 = 45^\\circ$", fontsize = 18)
plt.xlabel("Wavelength (nm)", fontsize = 13)
plt.ylabel("ADC Counts", fontsize = 13)
plt.savefig("BandStopFar_0_45_Exp")
plt.show()
#%%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
bandstopfar_0_0 = pd.read_csv('bandstopfar_0_0.csv')
cols = ['Pixel', 'Wavelength', 'Sum', 'Average']
bandstopfar_0_0.columns = cols
# Subtracting ambient light here makes graph blank; goes negative somewhere?
bandstopfar_0_0f = interp1d(bandstopfar_0_0['Wavelength'], bandstopfar_0_0['Average'], kind='cubic')
plt.plot(bandstopfar_0_0['Wavelength'], bandstopfar_0_0f(bandstopfar_0_0['Wavelength']))
plt.title("(a) $\u03B8 = 0^\\circ $", fontsize = 18)
plt.xlabel("Wavelength (nm)", fontsize = 13)
plt.ylabel("ADC Counts", fontsize = 13)
plt.savefig("BandStopFar_0_0_Exp")
plt.show()
#%%
plt.plot(wavelengthArray, sp0bs45f(wavelengthArray)*65535, 'k--', wavelengthArray,
whitelightf(wavelengthArray), 'r-')
plt.title("Source Light and Calculated Transmission")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts & Transmission")
plt.legend(['Calculated Transmission (SP0, BS45)', 'Experimental Light'], loc='best')
plt.savefig("ExperimentalLight_CalculatedTransmissionBS45")
plt.show()

plt.plot(wavelengthArray, sp0bs45f(wavelengthArray) * whitelightf(wavelengthArray))
plt.title("Source Light x Calculated Transmission")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Predicted ADC Counts")
plt.savefig("ExperimentalLight_X_CalculatedTransmissionBS45")
plt.show()

plt.plot(wavelengthArray, sp0bs45f(wavelengthArray) * whitelightf(wavelengthArray), 'b--', wavelengthArray,
bandstopfar_0_45f(wavelengthArray), 'k-')
plt.title('Predicted and Experimental Spectrums, SP0 BS45')
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
plt.legend(['Calculated', 'Experimental'], loc='best')
plt.savefig("Experimental0_45_and_CalculatedTransmission")
plt.show()
#%%
# Bandstop Filter at 0 Degrees
bandstop0 = pd.read_csv('bandstop_0.csv')
bandstop0.columns = cols
bandstop0f = interp1d(bandstop0['Wavelength'], bandstop0['Average'], kind='cubic')
plt.plot(bandstop0['Wavelength'], bandstop0f(bandstop0['Wavelength']))
plt.title("Bandstop Filter at 0 Degrees")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
```

```
plt.savefig("Bandstop_0")
plt.show()
#%%
# Bandstop Filter at 45 Degrees
bandstop45 = pd.read_csv('bandstop_45.csv')
bandstop45.columns = cols
bandstop45f = interp1d(bandstop45['Wavelength'], bandstop45['Average'], kind='cubic')
plt.plot(bandstop45['Wavelength'], bandstop45f(bandstop45['Wavelength']))
plt.title("Bandstop Filter at 45 Degrees")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
plt.savefig("Bandstop_45")
plt.show()
#%%
bandstopclose_0_0 = pd.read_csv('bandstopclose_0_0.csv')
bandstopclose_0_0.columns = cols
bandstopclose_0_0f = interp1d(bandstopclose_0_0['Wavelength'], bandstopclose_0_0['Average'], kind='cubic')
plt.plot(bandstopclose_0_0['Wavelength'], bandstopclose_0_0f(bandstopclose_0_0['Wavelength']))
plt.title("Both, Bandstop Filter Close, SP0 BS0")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
plt.savefig("BSClose_SP0_BS0")
plt.show()
#%%
bandstopclose_45_0 = pd.read_csv('bandstopclose_45_0.csv')
bandstopclose_45_0.columns = cols
bandstopclose_45_0f = interp1d(bandstopclose_45_0['Wavelength'], bandstopclose_45_0['Average'], kind='cubic')
plt.plot(bandstopclose_45_0['Wavelength'], bandstopclose_45_0f(bandstopclose_45_0['Wavelength']))
plt.title("Both, Bandstop Filter Close, SP0 BS45")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
plt.savefig("BSClose_SP0_BS45")
plt.show()
#%%
bandstopfar_0_30 = pd.read_csv('bandstopfar_0_30.csv')
bandstopfar_0_30.columns = cols
# Subtracting ambient light here makes graph blank; goes negative somewhere?
bandstopfar_0_30f = interp1d(bandstopfar_0_30['Wavelength'], bandstopfar_0_30['Average'], kind='cubic')
plt.plot(bandstopfar_0_30['Wavelength'], bandstopfar_0_30f(bandstopfar_0_30['Wavelength']))
plt.title("(c) $\u03B8 = 30^\\circ$", fontsize = 18)
plt.xlabel("Wavelength (nm)", fontsize = 13)
plt.ylabel("ADC Counts", fontsize = 13)
plt.savefig("BandStopFar_0_30_Exp")
plt.show()
#%%
bandstopfar_0_15 = pd.read_csv('bandstopfar_0_15.csv')
bandstopfar_0_15.columns = cols
# Subtracting ambient light here makes graph blank; goes negative somewhere?
bandstopfar_0_15f = interp1d(bandstopfar_0_15['Wavelength'], bandstopfar_0_15['Average'], kind='cubic')
plt.plot(bandstopfar_0_15['Wavelength'], bandstopfar_0_15f(bandstopfar_0_15['Wavelength']))
plt.title("(b) $\u03B8 = 15^\\circ$", fontsize = 18)
plt.xlabel("Wavelength (nm)", fontsize = 13)
plt.ylabel("ADC Counts", fontsize = 13)
plt.savefig("BandStopFar_0_15_Exp")
plt.show()
#%%
```

```
shortpassonly = pd.read_csv('shortpassonly.csv')
shortpassonly.columns = cols
# Subtracting ambient light here makes graph blank; goes negative somewhere?
shortpassonlyf = interp1d(shortpassonly['Wavelength'], shortpassonly['Average'], kind='cubic')
plt.plot(shortpassonly['Wavelength'], shortpassonlyf(bandstopfar_0_15['Wavelength']))
plt.title("Shortpass Filter")
plt.xlabel("Wavelength (nm)")
plt.ylabel("ADC Counts")
plt.savefig("Shortpass Only")
plt.show()

#%%
wavelengthArray = np.arange(380, 710, 10).tolist()

# White light, experimental
whitelight = pd.read_csv('whitelight-avg100_50ms.csv')
whitelight.columns = cols
# Subtract off ambient light
whitelight['Average'] -= ambientlight['Average']
whitelightf = interp1d(whitelight['Wavelength'], whitelight['Average'], kind='cubic')

plt.plot(wavelengthArray, whitelightf(wavelengthArray), 'r--', wavelengthArray, bandstop45f(wavelengthArray), 'g-
-', wavelengthArray, shortpassonlyf(wavelengthArray), 'b--', wavelengthArray,
bandstopfar_0_45f(wavelengthArray), 'k')
plt.title('Filters Independent and Combined')
plt.xlabel("Wavelength (nm)")
plt.ylabel('ADC Counts')
plt.legend(['Source Light', 'BS45', 'SP0', 'SP0BS45'], loc='best')
plt.savefig("FiltersSep&Combined")
plt.show()
#%%
# Plot actual transmission of combined filters at 45
# This can be done by dividing the amount of the light with filters by the light with no filters
plt.plot(wavelengthArray, bandstopfar_0_45f(wavelengthArray)/whitelightf(wavelengthArray))
plt.title("Experimentally Determined Transmission by Wavelength")
plt.xlabel("Wavelength (nm)")
plt.ylabel('Transmission')
plt.savefig("ExperimentalTransmissionWRONG")
plt.show()

#%%

from skimage import data
from skimage import io
from skimage.color import rgb2hsv, hsv2rgb

# # changed
 rgb_img = data.coffee()
 rgb_img = io.imread("rainbow_squares.jpg")
#
# hsv_img = rgb2hsv(rgb_img)
# hue_img = hsv_img[:, :, 0]
# value_img = hsv_img[:, :, 2]
#
# fig, (ax0, ax1, ax2, ax3) = plt.subplots(ncols=4, figsize=(8, 2))
#
```

```
# ax0.imshow(rgb_img)
# ax0.set_title("RGB image")
# ax0.axis('off')
# ax1.imshow(hue_img, cmap='hsv')
# ax1.set_title("Hue channel")
# ax1.axis('off')
# ax2.imshow(value_img)
# ax2.set_title("Value channel")
# ax2.axis('off')
#
# new_rgb_img = hsv2rgb(hsv_img)
# ax3.imshow(new_rgb_img)
# ax3.set_title("HSV to RGB")
# ax3.axis('off')
#
#
#
# fig.tight_layout()
# fig.savefig("Original")
# fig.show()



# rainbow = io.imread("rainbow_squares.jpg")
# hsv_rainbow = rgb2hsv(rainbow)
# hue_rainbow = hsv_rainbow[:, :, 0]
# value_rainbow = hsv_rainbow[:, :, 2]
#
# fig, (ax0, ax1, ax2, ax3) = plt.subplots(ncols=4, figsize=(8, 2))
#
# ax0.imshow(rainbow)
# ax0.set_title("RGB Rainbow")
# ax0.axis('off')
# ax1.imshow(hue_rainbow, cmap='hsv')
# ax1.set_title("Hue channel")
# ax1.axis('off')
# ax2.imshow(value_rainbow)
# ax2.set_title("Value channel")
# ax2.axis('off')
#
# # Get wavelength from hue?
# # https://stackoverflow.com/questions/11850105/hue-to-wavelength-mapping
# wavelengths_from_image = ((650 - 250) / 270) * hue_img
# # print(wavelengths_from_image)
#
# # JUST TESTING IMPACT
# # V is the 2 column
# # Setting all values to 0 makes whole image black
# # Gets darker as you multiply by numbers less than 1
# # hsv_rainbow[:, :, 2] = 1 * hsv_rainbow[:, :, 2]
#
# # JUST TESTING IMPACT
# # S is the 1 column
# # Setting s to 0 and changing nothing else makes the image grayscale
# # hsv_rainbow[:, :, 1] = 0.1 * hsv_rainbow[:, :, 1]
```

```
#
# # JUST TESTING IMPACT
# # H is the 1 column
# # Setting h to 0 and changing nothing else makes the image all red
# # Multiplying by 0.5 changes colors, as you'd expect
# # hsv_rainbow[:, :, 0] = 0.5 * hsv_rainbow[:, :, 0]
#
# # So!
# # Go through the entire array, get wavelength of pixel (first two indices)
# # by using the hue value (a 0 in the third index)
# # Knowing the wavelength, do different things to different SV values at
# # that pixel according to the transmission spectrum
# # Namely,
#
#
# # get transmission by hue!
# # should save some work inside loops
#
maxhue = 270 # was 270, changed to 360
wavelengthrange = 250 # was 250, changed to 320
maxwavelength = 650 # was 650, changed to 700
# hueArray = list(map(lambda x: (x / ((maxwavelength - wavelengthrange) / maxhue)), wavelengthArray))
hueArray = list(map(lambda x: (x - maxwavelength) * (-1)*(maxhue / wavelengthrange), wavelengthArray))
# sp0bs45f() is function for calculated transmission by wavelength
hueTransmission = sp0bs45f(wavelengthArray) * hueArray / maxhue
hueTransmissionf = interp1d(hueArray, hueTransmission, kind='cubic')
#
# rows = hsv_rainbow.shape[0]
# columns = hsv_rainbow.shape[1]
# # print(rows)
# # print(columns)
# print("Entering modification loop!")
# for i in range(0, rows):
#     print(str(i))
#     for j in range(0, columns):
#         # TODO make the alterations more sophisticated, at the moment just attenuates
#         # Modify saturation
#         # print("Hue: " + str(270 * hsv_rainbow[i, j, 0]))
#         # print("Transmission: " + str(hueTransmissionf(270 * hsv_rainbow[i, j, 0])))
#
#         hsv_rainbow[i, j, 1] = hsv_rainbow[i, j, 1] * (hueTransmissionf(maxhue * (1 - hsv_rainbow[i, j, 0])))
#         # Modify value
#         hsv_rainbow[i, j, 2] = hsv_rainbow[i, j, 2] * (hueTransmissionf(maxhue * (hsv_rainbow[i, j, 0])))
#
#         # hsv_rainbow[i, j, 0] = 0 # Make everything red to test
#         # Wavelength of that pixel?:
#         # Thought process:
#         # The less transmission in a given hue, the more grayscale it should be
#         # Reducing V makes more dark
#         # Reducing S makes more grayscale
#
# #new 2 was with 1 being inverted, 2 commented out
# #new 3 is with 1 being inverted, 2 also inverted
# #new 4 is with 1 being inverted, 2 not inverted
#
#
```

```
#
# new_rgb_rainbow = hsv2rgb(hsv_rainbow)
# ax3.imshow(new_rgb_rainbow)
# ax3.set_title("HSV to RGB")
# ax3.axis('off')
#
# fig.tight_layout()
# fig.savefig("rainbow_modifyingsaturation_new4")
# fig.show()
#
#
#
#
plt.plot(hueArray, hueTransmissionf(hueArray), '-')
plt.title("Transmission by Hue")
plt.xlabel("Hue")
plt.ylabel("Transmission")
plt.savefig("Transmission by Hue")



# ishihara45 = io.imread("ishihara45.jpg")
# hsv_ishihara45 = rgb2hsv(ishihara45)
# hue_ishihara45 = hsv_ishihara45[:, :, 0]
# value_ishihara45 = hsv_ishihara45[:, :, 2]
#
# fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(8, 2))
#
# ax0.imshow(ishihara45)
# ax0.set_title("RGB ishihara45")
# ax0.axis('off')
#
#
# rows = hsv_ishihara45.shape[0]
# columns = hsv_ishihara45.shape[1]
# print("Entering modification loop!")
# for i in range(0, rows):
#     print(str(i))
#     for j in range(0, columns):
#         hsv_ishihara45[i, j, 1] = hsv_ishihara45[i, j, 1] * (hueTransmissionf(maxhue * (1 - hsv_ishihara45[i, j, 0])))
#         # Modify value
#         # hsv_rainbow[i, j, 2] = hsv_rainbow[i, j, 2] * (1 - (hueTransmissionf(maxhue * (1 - hsv_rainbow[i, j, 0]))))
#
#
#
# new_rgb_ishihara45 = hsv2rgb(hsv_ishihara45)
# ax1.imshow(new_rgb_ishihara45)
# ax1.set_title("HSV to RGB")
# ax1.axis('off')
#
# fig.tight_layout()
# fig.savefig("ishihara45modified_2")
# fig.show()

# This is sensitivity of each cone at various wavelengths multiplied by our filter's transmission
s_new_sensitivity = sconef(wavelengthArray)*sp0bs45f(wavelengthArray)
```

```
m_new_sensitivity = mconef(wavelengthArray)*sp0bs45f(wavelengthArray)
l_new_sensitivity = lconef(wavelengthArray)*sp0bs45f(wavelengthArray)

# rainbow = io.imread("spectrum.jpg")
# rgb_rainbow = rainbow
# hsv_rainbow = rgb2hsv(rainbow)
# hue_rainbow = hsv_rainbow[:, :, 0]
# value_rainbow = hsv_rainbow[:, :, 2]
#
# fig, (ax0, ax1) = plt.subplots(nrows=2, figsize=(8, 2))
#
# ax0.imshow(rgb_rainbow)
# ax0.set_title("RGB Rainbow Before")
# ax0.axis('off')
#
# rows = hsv_rainbow.shape[0]
# columns = hsv_rainbow.shape[1]
# print(rows)
# print(columns)
# print("Entering modification loop!")
# # Maybe I could do this faster with a map?
# for i in range(0, rows):
#     print(str(i))
#     for j in range(0, columns):
#         # Get dominant wavelength from the hue
#         pixel_dominant_wavelength = maxwavelength - ((wavelengthrange / maxhue) * maxhue * hsv_rainbow[i, j,
0])
#         # Get sensitivities of cones at that wavelength
#         s_sensitivity = sconef(pixel_dominant_wavelength) * sp0bs45f(pixel_dominant_wavelength)
#         m_sensitivity = mconef(pixel_dominant_wavelength) * sp0bs45f(pixel_dominant_wavelength)
#         l_sensitivity = lconef(pixel_dominant_wavelength) * sp0bs45f(pixel_dominant_wavelength)
#         # print("New Pixel, Hue: " + str(hsv_rainbow[i, j, 0]) + ", Wavelength: " + str(pixel_dominant_wavelength))
#         # print(str(rgb_rainbow[i, j, 0]) + " * " + str(l_sensitivity))
#         # print(str(rgb_rainbow[i, j, 1]) + " * " + str(m_sensitivity))
#         # print(str(rgb_rainbow[i, j, 2]) + " * " + str(s_sensitivity))
#
#         # I want the ratio R / G = L_Sens / M_Sens for given pixel
#         # I also want the ratio G / B = M_Sens / S_Sens for given pixel
#         # First I will try this using G as reference and modifying the others
#             # That didn't work because divide by zero
#             # Try average
#
#         # R = G * L_Sens / M_Sens
#         # B = G * S_Sens / M_Sens
#
#         r_g_ave = (rgb_rainbow[i, j, 0] + rgb_rainbow[i, j, 1]) / 2
#         r_g_b_ave = (1/3)*(rgb_rainbow[i, j, 0] + rgb_rainbow[i, j, 1] + rgb_rainbow[i, j, 2])
#         # print("HERE")
#         # print((l_sensitivity / m_sensitivity))
#         l_over_m_sens_sqrt = np.sqrt(abs(l_sensitivity / m_sensitivity))
#         # Modify blue after obtaining green
#         rgb_rainbow[i, j, 0] = r_g_ave * l_over_m_sens_sqrt
#         rgb_rainbow[i, j, 1] = r_g_ave / l_over_m_sens_sqrt
#         rgb_rainbow[i, j, 2] = rgb_rainbow[i, j, 1] * (s_sensitivity / m_sensitivity)
#         # print(rgb_rainbow[i, j, 0] / rgb_rainbow[i, j, 1])
#         # this_g = rgb_rainbow[i, j, 1]
```

```
#         # rgb_rainbow[i, j, 0] = this_g * (l_sensitivity / m_sensitivity)
#         # rgb_rainbow[i, j, 2] = this_g * (s_sensitivity / m_sensitivity)
#
#
#         # Modify rgb_rainbow
#         # 0th index is red, 1st is green, 2nd is blue
#         # rgb_rainbow[i, j, 0] = rgb_rainbow[i, j, 0] * l_sensitivity + 75
#         # rgb_rainbow[i, j, 1] = rgb_rainbow[i, j, 1] * m_sensitivity + 75
#         # rgb_rainbow[i, j, 2] = rgb_rainbow[i, j, 2] * s_sensitivity + 75
# ax1.imshow(rgb_rainbow)
# ax1.set_title("AfterModifying")
# ax1.axis('off')
# fig.tight_layout()
# fig.savefig("Spectrum_Approach3_RealVals_averagingmodifyingrgb")
# fig.show()


ishihara45 = io.imread("ishihara45.jpg")
hsv_ishihara45 = rgb2hsv(ishihara45)

fig, (ax0, ax1) = plt.subplots(nrows=2, figsize=(8, 2))

ax0.imshow(ishihara45)
ax0.set_title("RGB ishihara45")
ax0.axis('off')

rows = hsv_ishihara45.shape[0]
columns = hsv_ishihara45.shape[1]
print("Entering modification loop!")
for i in range(0, rows):
    print(str(i))
    for j in range(0, columns):
        pixel_dominant_wavelength = maxwavelength - ((wavelengthrange / maxhue) * maxhue * hsv_ishihara45[i, j,
0])
        r_g_ave = (ishihara45[i, j, 0] + ishihara45[i, j, 1]) / 2
        s_sensitivity = sconef(pixel_dominant_wavelength) * sp0bs45f(pixel_dominant_wavelength)
        m_sensitivity = mconef(pixel_dominant_wavelength) * sp0bs45f(pixel_dominant_wavelength)
        l_sensitivity = lconef(pixel_dominant_wavelength) * sp0bs45f(pixel_dominant_wavelength)
        r_g_ave = (ishihara45[i, j, 0] + ishihara45[i, j, 1]) / 2
        l_over_m_sens_sqrt = np.sqrt(abs(l_sensitivity / m_sensitivity))
        ishihara45[i, j, 0] = r_g_ave * l_over_m_sens_sqrt
        ishihara45[i, j, 1] = r_g_ave / l_over_m_sens_sqrt

ax1.imshow(ishihara45)
ax1.set_title("After Changing R and G")
ax1.axis('off')

fig.tight_layout()
fig.savefig("Ishihara45_App3_RG_ForumVals")
fig.show()
```