COLOR SCIENCE

Charmaine S. Tolledo

OBJECTIVES

01

Understand Chromaticity Basics

Gain a foundational understanding of CIE-xy chromaticity coordinates and their application in reproducing the CIExy Chromaticity Diagram.

02

Analyze Color Differences

Comprehend the transformation from CIE xy to u'v' chromaticity coordinates and its practical implications, including the assessment of color differences.

03

Recreate Color with Light

Explore the process of rendering colors and arranging them in a chart, with a focus on evaluating color similarity with real-world images.

ACTIVITY 1 OF 3: Trichromaticity of Color Mixture

CIE Standard Observer Data

```
% Load CIE Standard Observer 1964 data from CSV file
        cie_data = 'CIE_xyz_1964_10deg.csv';
        cie_data = readmatrix(cie_data);
```

Figure 1
Matlab code snippet of loading the CIE Standard Observer Data

The CIE Standard Observer data is a fundamental basis of color science, as it represents the typical color sensitivity of the human eye throughout the visible spectrum. This data pertains to the color matching functions (x, y, and z), which quantify the response of individual cone photoreceptors in the eye to varying wavelengths of light.

The first task for this activity is to download the table named 'CIE_xyz_1964_10deg.csv'. This file contains the CIE Standard Observer 1964 data and is readily available online.

CIE xy Chromaticity Coordinate Calculation

```
% Define the range of wavelengths (Monochromatic lights from 380nm to 780nm in steps of 10nm)
   wavelengths = 380:10:780;

% Initialize arrays to store xy chromaticity coordinates and tristimulus values
   y_values = zeros(size(wavelengths));
   z_values = zeros(size(wavelengths));
   x_stored = zeros(size(wavelengths));
   y_stored = zeros(size(wavelengths));
   z_stored =zeros(size(wavelengths));

 % Loop through each wavelength
   for i = 1:length(wavelengths)
       wavelength = wavelengths(i);

       % Find the index corresponding to the given wavelength
       idx = find(cie_data(:, 1) == wavelength);

       % Extract color matching functions at the specified wavelength
       x_bar = cie_data(idx, 2);
       y_bar = cie_data(idx, 3);
       z_bar = cie_data(idx, 4);

       % Calculate tristimulus values
       X = x_bar;
       Y = y_bar;
       Z = z_bar;

       % Normalize tristimulus values
       X_normalized = X / (X + Y + Z);
       Y_normalized = Y / (X + Y + Z);

       % Calculate CIE-xy chromaticity coordinates
       x_values(i) = X_normalized;
       y_values(i) = Y_normalized;

       %in order to store the values of X Y Z
       x_stored(i) = X;
       y_stored(i) = Y;
       z_stored(i) = Z;
   end
```

Figure 2
Matlab code snippet of calculating CIE xy Chromaticity Coordinate

Now, to calculate the CIE xy chromaticity coordinates for various monochromatic lights, the first step we took was to define a range of wavelengths spanning from 380nm to 780nm, which encompasses the visible spectrum.

As each wavelength was iterated through, color matching functions were used to calculate the tristimulus values (X, Y, Z). These values were then normalized to obtain X_normalized and Y_normalized. Subsequently, the CIE-xy chromaticity coordinates (x_values and y_values) were determined. The original tristimulus values were also retained for future analysis.

CIE-xy Chromaticity Diagram

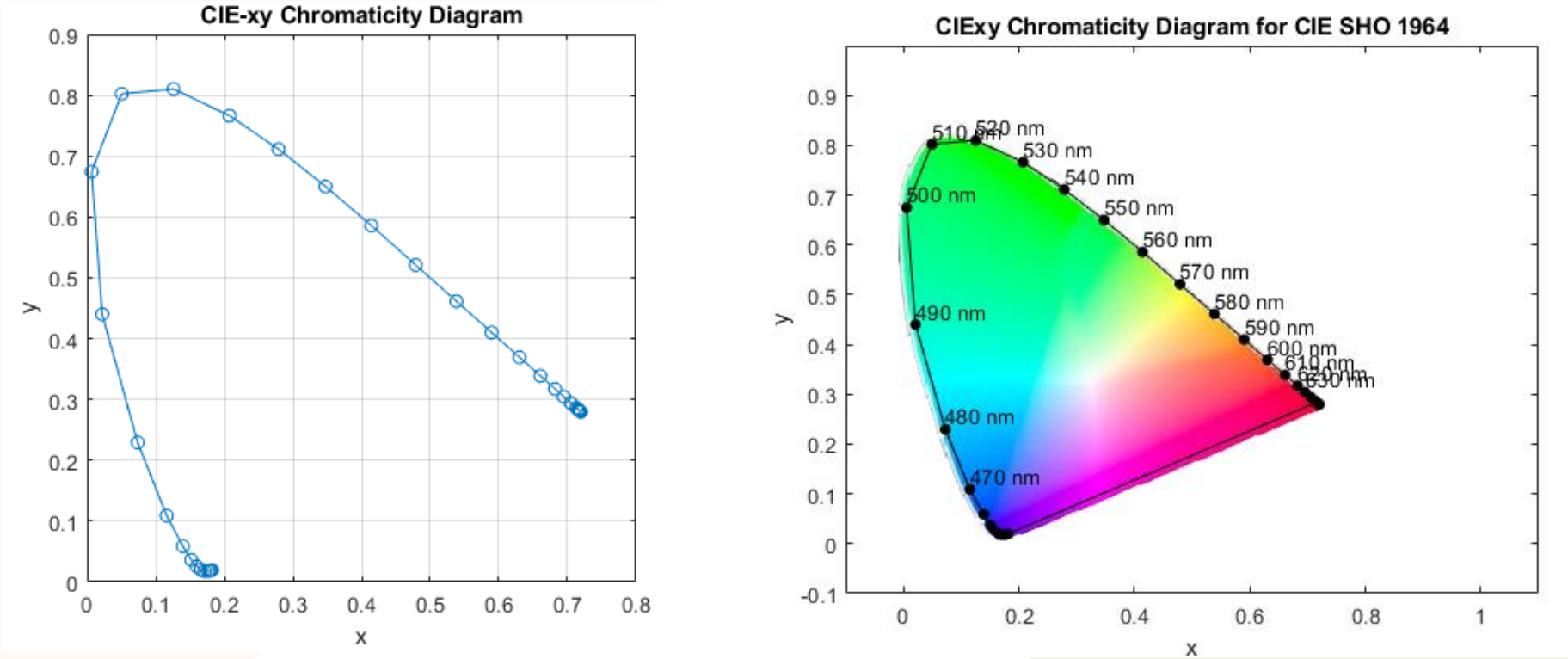

Figure 3

(a) The boundaries of the CIE xy chromaticity coordinates plot and (b) The CIE xy chromaticity coordinates plot with a color tongue embedded

After calculating the CIE xy chromaticity coordinates, we plotted the results and embedded it in the color tongue image, thus giving us the two plots above. These visually map the entire visible spectrum. It doesn't only provides a visual representation of how colors are distributed, but also provides valuable information on the range of colors that can be created by mixing different shades of light.

Planckian Locus

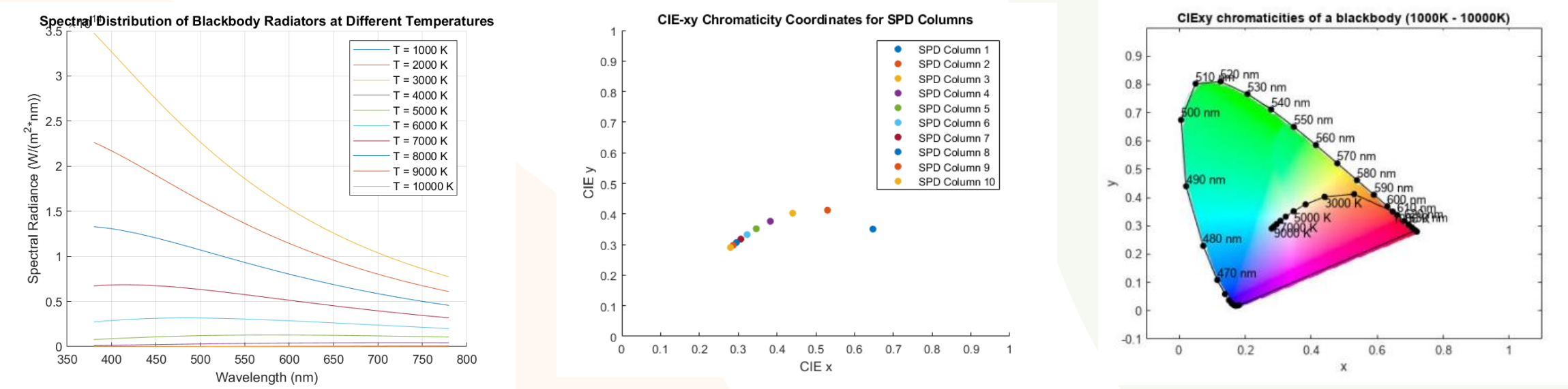

Figure 4

(a) Spectral distribution of Blackbody Radiators at different temperatures; (b) CIE-xy chromaticity coordinates for SPD columns; and (c) CIE xy chromaticities of a blackbody

For the second half of the activity, we were instructed to calculate the CIE-xy chromaticities of a blackbody emitting from 1000K to 10,000K in steps of 1000K and plot it in CIE xy chromaticity diagram, thus we got 3 plots. The first plot shows how black bodies' underlying energy is distributed across the visible range. As temperature increases, the peak of the distribution moves towards shorter wavelengths, changing the color from red to blue hues. This was done to get the spectral data.

The second figure plots the calculated CIE xy coordinates for each blackbody temperature on the CIE xy Chromaticity Diagram. These points collectively form the renowned "Planckian Locus," a distinct curved path that represents the range of color that can be produced by blackbody radiators.

And lastly, the final figure in which we successfully embedded the Planckian Locus into the color tongue image.

ACTIVITY 2 OF 3: Color Order System and Color Difference Specification

Spectral power distribution of Illuminant D65

```
% Load the spectral power distribution of Illuminant D65
illuminant = readmatrix('CIE_std_illum_D65.csv');
cie_d65 = illuminant(81:10:481,2);
```

Figure 5
MATLAB code snippet of loading the spectral power distribution of Illuminant D65

The spectral power distribution (SPD) of Illuminant D65, which represents the average midday sunlight, is essential for this task. By serving as the standard light source, it helps in the computation and analysis of the u'v' chromaticity coordinates for different light sources.

u'v' chromaticity coordinates calculation

```
% Calculate the tristimulus values for Illuminant D65
d65_X = sum(cie_d65.*xbar);
d65_Y = sum(cie_d65.*ybar);
d65_Z = sum(cie_d65.*zbar);

% Compute u'v' chromaticity coordinates for Illuminant D65
d65_u_prime = (4*d65_X)/(d65_X + 15*d65_Y + 3*d65_Z);
d65_v_prime = (9*d65_Y)/(d65_X + 15*d65_Y + 3*d65_Z);

% Initialization of matrices for monochromatic lights and Planckian sources
u_prime = zeros(41,1);
v_prime = zeros(41,1);
u_prime_bb = zeros(10,1);
v_prime_bb = zeros(10,1);

% Calculate u'v' chromaticity coordinates for monochromatic lights and Planckian sources
for i = 1:numel(u_prime(:,1))
 u_prime(i,1) = (4*x(i,1))/(-2*x(i,1) + 12*y(i,1) + 3);
 v_prime(i,1) = (9*y(i,1))/(-2*x(i,1) + 12*y(i,1) + 3);
end

for i = 1:10
    u_prime_bb(i, 1) = (4 * xy_coordinates(i, 1)) / (-2 * xy_coordinates(i, 1) + 12 * xy_coordinates(i, 2));

    v_prime_bb(i, 1) = (9 * xy_coordinates(i, 2)) / (-2 * xy_coordinates(i, 1) + 12 * xy_coordinates(i, 2));
end
```

Figure 6
MATLAB code snippet of calculating u'v' chromaticity coordinates

- The code snippet above was used to calculate and store the u'v' chromaticity coordinates of various light sources. First, the tristimulus values (X, Y, Z) for Illuminant D65 were computed, followed by the derivation of its corresponding u'v' coordinates. Subsequently, empty matrices were prepared to accommodate the u'v' values for both monochromatic lights and Planckian sources. Finally, an iterative process was employed to calculate and store the respective u'v' coordinates for each type of light source.

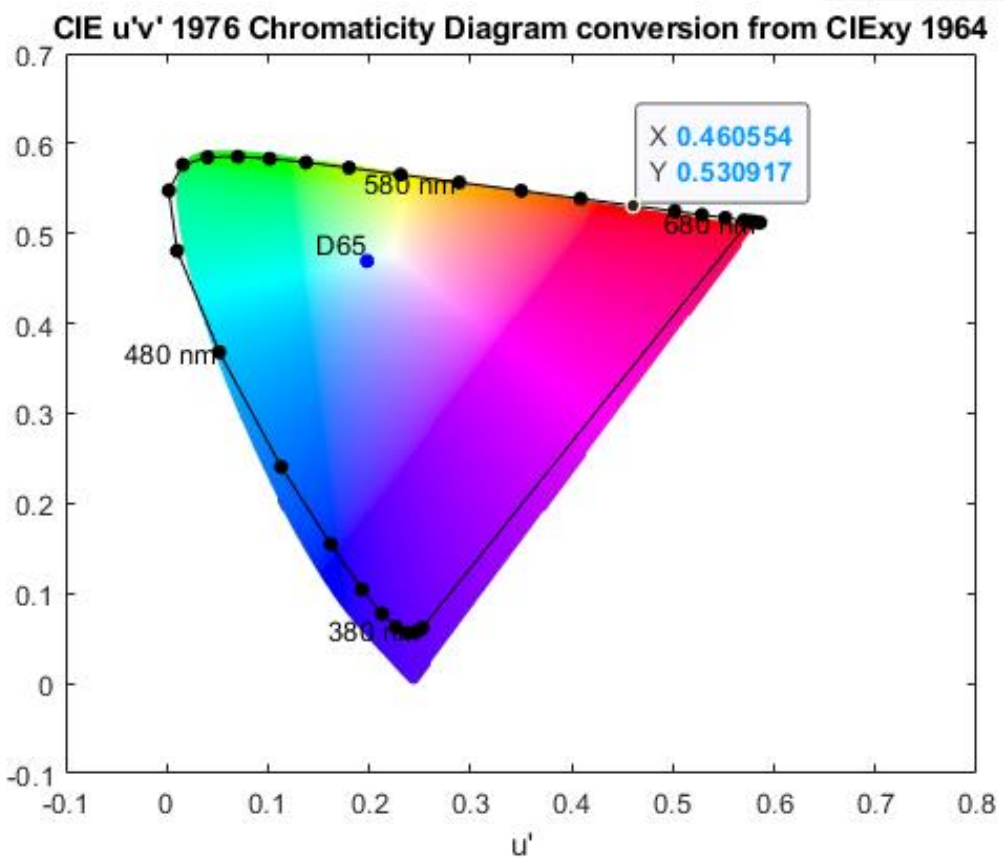

Figure 7
CIE u'v' chromaticity diagram conversion from CIE xy 1964 plot

This is the result as we plot the u'v' coordinates of various light sources within the CIE 1976 chromaticity diagram. Each point corresponds to the u'v' coordinates of monochromatic lights, Planckian sources, and Illuminant D65. The standardized u'v' space enables us to make more accurate comparisons between various sources and analyze the individual characteristics of each light source and their relationships within the color space.

Reflectance measurements of 1200 Munsell color chips

```
% Load reflectance measurements from the file
load('munsell400_700_5.mat');

% Specify three new random chip indices
chipRandomizer = [676; 1119; 456];
```

Figure 8
MATLAB code snippet of loading the reflectance measurements

The file munsell400_700_5.mat contains reflectance measurements of 1200 Munsell color chips, taken at 5-nanometer intervals across the visible spectrum (400nm to 700nm). Color chips play an important part in color science and are extensively used for comprehending color perception, calibration, and color matching.

For analysis, three color chips were randomly selected from the dataset (Chip numbers: 676; 1119; 456).

Munsell color ships on D65 and White Light Illuminant

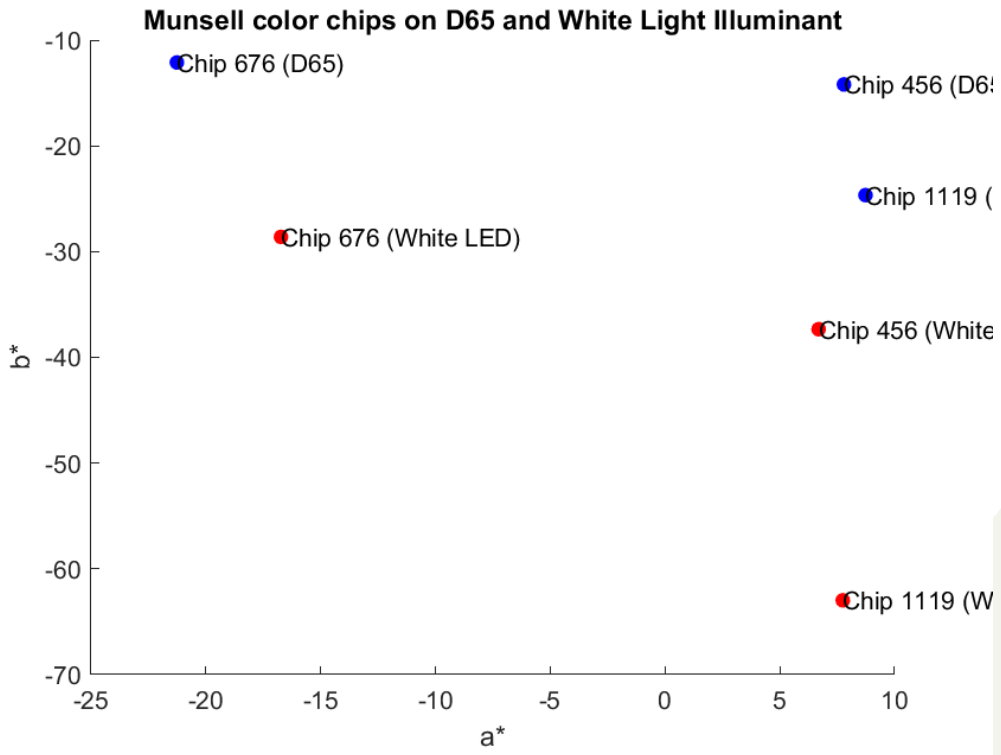

Figure 9
Munsell color chips on D65 and white light illuminant plot

Here, color shifts experienced by three randomly selected Munsell chips when illuminated by a D65 light source and a white LED were visualized. The XYZ values for each chip under both lights were calculated, followed by conversion to the CIE Lab space. And above is the generated plot comparing the ab values for each chip under both illuminants. It provides a visual representation of the color shifts.

Color differences

```
difference = 3×1
    4.5467
    1.0672
    1.1173
```

Figure 10
(a) Value and (b) Visual color difference between the observed color under D65 and white LED

Here, we calculated the color difference between the observed color under D65 and white LED. The Euclidean distance in the Lab color space is computed, considering the differences in lightness (L*), chroma (a*), and hue (b*). The resulting differences are then displayed, as seen in the figure above.

And as we can observe, The two rectangles, one representing the Munsell color of Chip 676 and the other representing the perceived color under the white LED, are clearly different in color, indicating a noticeable color shift between the two lighting conditions. Under the white LED, the Munsell color of Chip 676 appears more saturated and reddish in comparison to D65. This is possibly the result of the white LED's spectral power distribution, which causes it to emanate a greater quantity of light in the red wavelength region compared to D65.

ACTIVITY 3 OF 3:
Color Image Capture and Color Rendering

Files used

```
% Load data
Kodak_DCS_420 = load("KODAK DCS 420.txt");
canon_400D = load("CANON 400D.txt");
macbeth = readmatrix('MacbethColorChecker.xls');
```

Figure 11
MATLAB code snippet of loading the Macbeth color checker and the two camera's spectral sensitivity

In this activity, we were tasked to simulate the appearance of the Macbeth Color Checker under different camera sensors. With that, it is important to load the file containing the Macbeth Color Chart itself. After so, two camera's spectral sensitivity were downloaded from https://nae-lab.org/~rei/research/cs/zhao/database.html, in this case, we chose the Canon 400D and Kodak DCS 420 cameras.

Plots

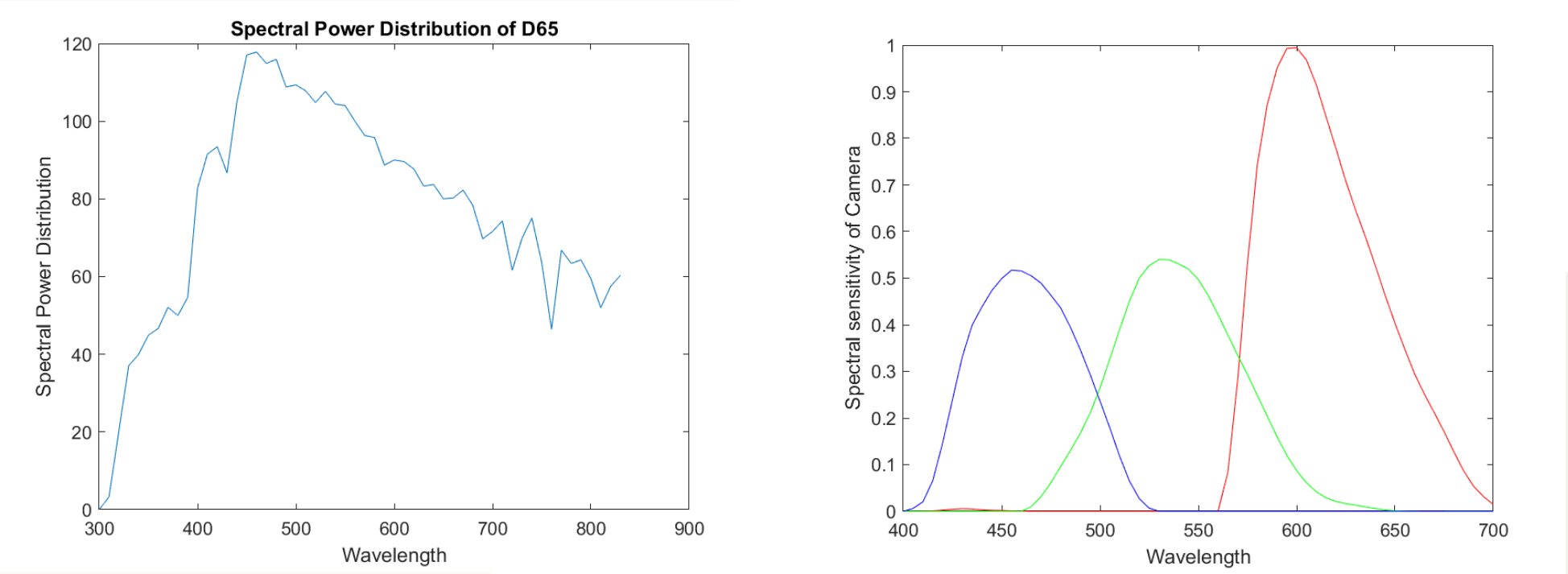

Figure 12
(a) Spectral Power Distribution of D65 and (b) Spectral camera sensitivity vs. wavelength plot

New x-values with finer steps of 5 nanometers were defined in the range of 400 to 700 nanometers. Subsequently, camera sensitivity data was interpolated to the newly defined wavelengths using the interp1 function. Relevant sections of the D65 illuminant and Macbeth Color Checker data were extracted to correspond with the chosen wavelength range. We have here the plots of Spectral power distribution of D65, which visualizes how the intensity of light varies across different wavelengths in the visible spectrum, and the Camera Sensitivity, which shows how sensitive the camera is to light at different wavelengths.

Color Rendering

```
% Matrix Initialization
rgbMatrixMacbeth_Canon = zeros(24,3);
for i = 1:24
 for j = 1:3
 rgbMatrixMacbeth_Canon(i,j) = sum(lightSourceEmittance.*surfaceReflectance(:,i).*sensitivityKodak_DCS_420(:,j))/sum(lightSourceEmittance.*sensitivityKodak_DCS_420(:,j));
 end
end

figure('Position', [100, 100, 1024, 512],'Color', [0.06 0.06 0.06]);
for i = 1:24
 subplot(4,6,i);
 bar(1, 1, 'FaceColor', rgbMatrixMacbeth_Canon(i,:), 'EdgeColor', 'none');

 title(sprintf('Patch %d', i),'Color','white');
 xlim([0.75 1.25]);
 ylim([0 0.2]);
 axis off;
end
saveas(gcf, 'Macbeth_recovery_canon400D.png');
```

Figure 13
MATLAB code snippet of color rendering using a Canon 400D camera

This code simulated the color rendering of the Macbeth ColorChecker under specific conditions using a Canon 400D camera. A matrix was created to record the RGB values for each of the 24 patches. The predicted RGB values for each patch were determined by calculating a weighted sum of the light source emittance, surface reflectance, and camera sensitivity. The product was subsequently combined across the specified range of wavelengths and normalized, giving the camera's response for each individual channel. The RGB values obtained for each patch were shown as colored bars organized in a style that imitated the original Macbeth chart.

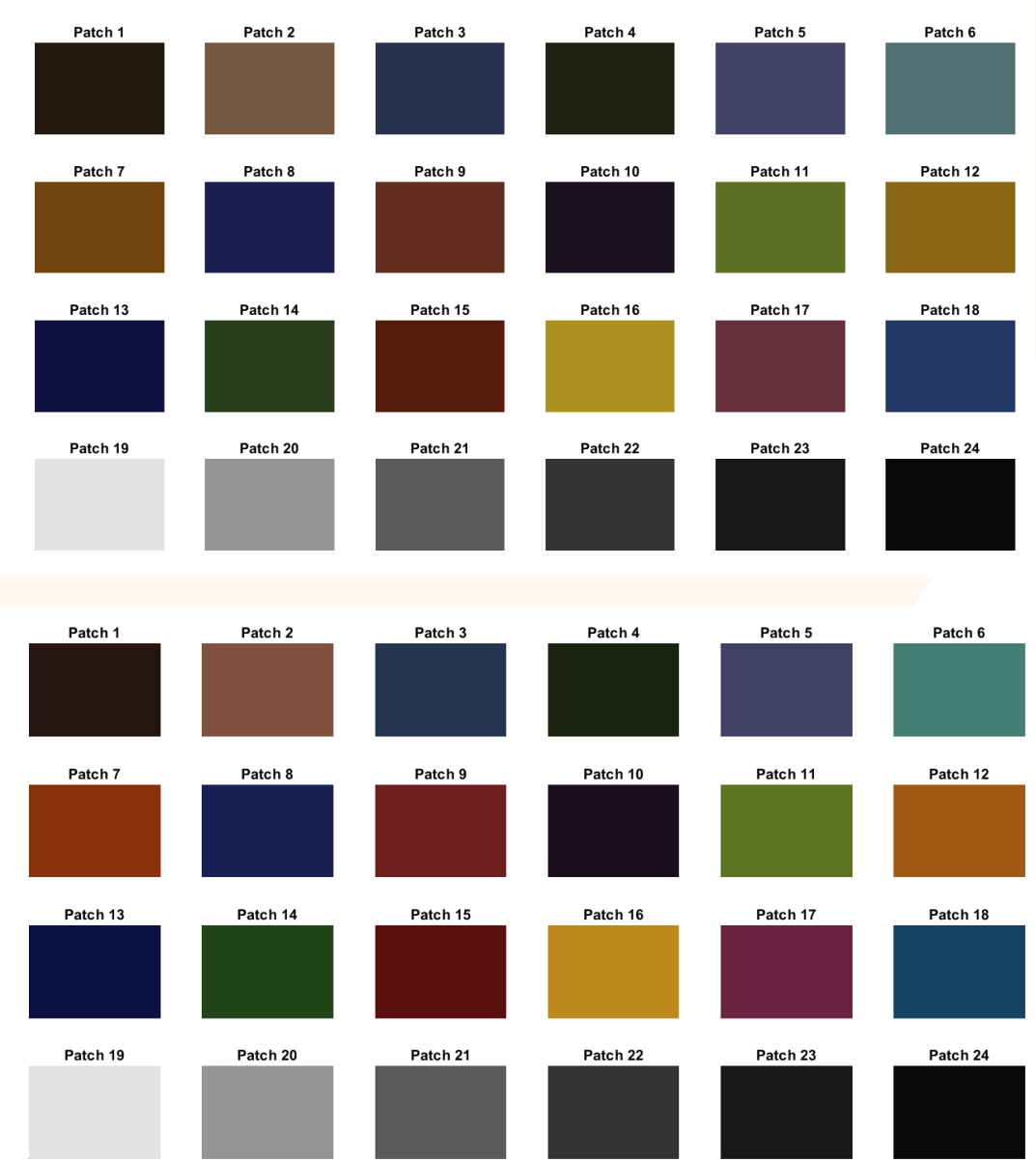

Figure 14
Color rendered of (a) canon_400D and (b) Kodak_DCS_420

We have successfully rendered the colors in two different camera's spectral sensitivity. And if we observe closely, we can see that The Kodak DCS 420 has a consistently brighter and more saturated appearance in comparison to the Canon 400D. These results suggest that the Kodak sensor is more sensitive to light across the visible spectrum, resulting in enhanced brightness.

GROUP ACTIVITY:
Color Capture in Cameras

Methodology

Figure 15
The captured image of different colorful objects in (a) normal color mode and under (b) red, (c) blue, and (d) green lights

This activity is a color capture in cameras. With this, the materials used were digital camera set to BW, tripod or camera holder, red, green and blue lights, and a bunch of colorful objects. The colorful objects were arranged on a flat surface. We ensured that there were representations of all major hues and the inclusion of a white object in the scene (in this case, the tissue). The camera was set to black and white mode and mounted on a tripod for stability.
The objects were illuminated with three separate colored lights – red, blue, and green – while an image of the tableau was captured under each individual light source. This resulted in three distinct images capturing the scene under each colored illumination. Careful positioning ensured no camera movement between captures to avoid misalignment. Additionally, an image of the tableau was captured with the camera set to normal color mode.

Through image processing software (GIMP)

Figure 16
Resulting color image using GIMP

The three captured black and white images were digitally overlaid to create a final color image. This was achieved through two methods: (a) Through image processing software such as GIMP or Photoshop and (b) Through programming.

Using GIMP, the three black and white images were loaded and the Color – Decompose – Compose function was utilized to overlay these images. The resulting composite image was then enhanced by applying Color – Auto – White Balance to improve color balance and tonal adjustments.

the resulting color image is displayed on the left.

Through programming

```
% Step i: Read the images as grayscale images
redImage = imread('red.JPG');
greenImage = imread('green.JPG');
blueImage = imread('blue.JPG');

% Ensure that the images are 2D
redImage = squeeze(redImage(:,:,1));
greenImage = squeeze(greenImage(:,:,1));
blueImage = squeeze(blueImage(:,:,1));

% Step ii: Normalize the grayscale images to the range [0, 1]
redImage = double(redImage) / 255;
greenImage = double(greenImage) / 255;
blueImage = double(blueImage) / 255;

% Step iii: Create a truecolor image matrix
M = size(redImage, 1);
N = size(redImage, 2);

% Initialize the truecolor image matrix
I = zeros(M, N, 3);

% Assign the grayscale images to the corresponding color channels
I(:,:,1) = redImage;
I(:,:,2) = greenImage;
I(:,:,3) = blueImage;

% Step iv: Enhance the image using white balancing
xI = I ./ max(I(:));

% Display the final image
figure; imshow(I);
title('Overlay of Grayscale Images');
```

Figure 17
(a) MATLAB code snippet and (b) Resulting color image through MATLAB

Now, through MATLAB, the images were read and normalized to the range [0, 1], to create a true color image matrix with dimensions MxN. Afterwards, each grayscale image was allocated to its own color channel in the matrix to reconstruct the original scene's color spectrum using the recorded light intensities. White balancing techniques were applied to improve the color representation by equalizing the different color channels which resulted in a more authentic and neutral appearance.

COMPARISON

Figure 18
(a) The color capture image using normal mode camera; and the digitally overlad color images using (b) GIMP and (c) MATLAB

Although the GIMP and MATLAB images capture the overall color palette of the original scene, they exhibit slight differences in brightness, saturation, and noise. The GIMP image is slightly brighter and more saturated than the original image. The colors are also slightly more vibrant. The detail retention is still good, but there is a slight loss of contrast in some areas. Meanwhile, the MATLAB image is slightly darker and less saturated than the original image. There is also some minor noise visible in the image. The detail retention is good, but not as good as the GIMP image. The obtained results can be attributed to the non-uniform illumination conditions during the individual red, green, and blue light captures. The varying light sources' places likely led to inconsistencies in the captured intensities which impacted the final color representation.

REFLECTION

This activity was the toughest one yet. It was challenging enough on its own, but then November rolled around and I landed myself in the hospital. That threw me off completely, and I wasn't able to fully grasp the concepts we were learning. This, unfortunately, led to my late submission. But even with the late start, I gave it my all and made sure every detail was covered, every question answered, and every aspect of the report was complete. I would like to thank Mar Princer for helping me understand the concept and helping me with the code.

Despite the late submission, I'm proud of the final report. Considering the quality of the work, the effort I put in, and the resilience I showed throughout the process, I believe a perfect score is still fair. So, I would give myself a **100/100.**

THANK YOU!