1. Color Reproduction

Part A:
The following is a plot of the XYZ functions.

The following is a plot of the spectrum of the illuminant $I$ and a plot of the spectra of the three display primaries $P_i$ ($i = 1, 2, 3$).

The XYZ values for the scene under the illuminant can be expressed using matrix notation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \bar{x}[1]/[1] & \cdots & \bar{x}[N_\lambda]/[N_\lambda] \\ \bar{y}[1]/[1] & \cdots & \bar{y}[N_\lambda]/[N_\lambda] \\ \bar{z}[1]/[1] & \cdots & \bar{z}[N_\lambda]/[N_\lambda] \end{bmatrix} \begin{bmatrix} \rho[1] \\ \vdots \\ \rho[N_\lambda] \end{bmatrix}$$

Similarly, we can express the XYZ values for the display with primaries $P_i[n]$, $i = 1, 2, 3$:
where \( w_1, w_2, w_3 \) are the weights for the primaries and \( A \) is the following 3x3 matrix:

\[
A = \begin{bmatrix}
\bar{x}[1] & \cdots & \bar{x}[N_{\lambda}]
\end{bmatrix} \begin{bmatrix}
\end{bmatrix}
\]

To match the two sets of XYZ values, we must have

\[
\begin{bmatrix}
w_1 \\
w_2 \\
w_3
\end{bmatrix} = A^{-1} \begin{bmatrix}
\bar{X} \\
\bar{Y} \\
\bar{Z}
\end{bmatrix} = A^{-1} \begin{bmatrix}
\bar{x}[1][1] & \cdots & \bar{x}[N_{\lambda}][1] \\
\bar{y}[1][1] & \cdots & \bar{y}[N_{\lambda}][1] \\
\bar{z}[1][1] & \cdots & \bar{z}[N_{\lambda}][1]
\end{bmatrix} \begin{bmatrix}
\rho[1] \\
\vdots \\
\rho[N_{\lambda}]
\end{bmatrix}
\]

Thus, the mapping from the spectral reflectance response to the weights for the three display primaries is the 3xN\(_{\lambda}\) matrix denoted as \( T \) in the above expression.

Part B:
Using the last expression in Part A, we can find the linear RGB weights/values at each pixel in the face image. To account for the display’s \( \gamma \)-nonlinearity, we apply an inverse \( \gamma \) function to each pixel value before displaying the image. Here, \( \gamma = 2.2 \).
Part C:
At pixel position \([x,y] = [47,149]\), we have the same XYZ values, but the product of the spectral reflectance response and the illuminant’s spectrum response is significantly different from the spectrum response emitted by the display. Thus, the two spectra are metamers.

MATLAB Code:

```matlab
% EE368/CS232
% Homework 2
% Problem: Color Reproduction
% Script by David Chen, Huizhong Chen
clc; clear all;

%% Load XYZ spectra
load('data/XYZ');
figure(1); clf;
set(gcf, 'Position', [50 50 400 300]);
h = plot(wavelength, XYZSpectra(:,1), 'r-', ...
    wavelength, XYZSpectra(:,2), 'g-', ...
    wavelength, XYZSpectra(:,3), 'b-');
set(h, 'LineWidth', 2);
xlabel('Wavelength \(\lambda\) (nm)'); ylabel('Energy');
legend('x(\lambda)', 'y(\lambda)', 'z(\lambda)');
numLambda = length(wavelength);

%% Load face spectra
load('data/face');
[rows, cols, bands] = size(face1Spectrum);
numPix = rows * cols;
face1Spectrum = reshape(face1Spectrum, [numPix bands]);

%% Load light spectra
load('data/light');
figure(2); clf;
set(gcf, 'Position', [100 100 400 300]);
plot(wavelength, light1Spectrum, 'LineWidth', 2); title('Illuminant I');
xlabel('Wavelength \(\lambda\) (nm)'); ylabel('Energy (Watts/sr/nm/m^2)');
axis([min(wavelength) max(wavelength) 0 1.2*max(light1Spectrum)]);

%% Find XYZ values
light1SpectrumRep = repmat(light1Spectrum, [numPix 1]);
face1LightSpectrum = face1Spectrum .* light1SpectrumRep;
faceXYZ = face1LightSpectrum * XYZSpectra;

%% Load display spectra
load('data/display');
figure(3); clf;
```
set(gcf, 'Position', [150 150 400 300]);
h = plot(wavelength, display1Spectra(:,1), 'r-', ...
    wavelength, display1Spectra(:,2), 'g-', ...
    wavelength, display1Spectra(:,3), 'b-');
set(h, 'LineWidth', 2);
xlabel('Wavelength \lambda (nm)'); ylabel('Energy (Watts/sr/nm/m^2)');
legend('P_1(\lambda)', 'P_2(\lambda)', 'P_3(\lambda)');
axis([min(wavelength) max(wavelength) 0 1.2*max(display1Spectra(:))]);

%% Solve for display primary weights
rgb2xyz = zeros(3,3);
rgb2xyz = XYZSpectra.' * display1Spectra;
xyz2rgb = inv(rgb2xyz);
faceRGB = faceXYZ * xyz2rgb.';

%% Show face image
faceImage = reshape(faceRGB, [rows cols 3]);
gamma = 2.2;
faceImage = faceImage / max(faceImage(:));
faceImage = faceImage .^ (1/gamma);
figure(4); clf;
set(gcf, 'Position', [200 200 400 300]);
imshow(faceImage,
imwrite(faceImage, 'face.jpg');

%% Plot some sample spectra
figure(5); clf;
set(gcf, 'Position', [250 250 600 300]);
x = 47; y = 149; nPix = (y-1)*cols + x;
lightProd = light1Spectrum .* reshape(face1Spectrum(nPix,:),1,numLambda);
subplot(1,2,1);
plot(wavelength, lightProd, 'LineWidth', 2);
xlabel('Wavelength \lambda (nm)'); ylabel('Energy (Watts/sr/nm/m^2)');
title('Scene x Illuminant');
axis([min(wavelength) max(wavelength) 0 1.2*max(lightProd)]);
lightProd = display1Spectra * faceRGB(nPix,:).';
subplot(1,2,2);
plot(wavelength, lightProd, 'LineWidth', 2);
xlabel('Wavelength \lambda (nm)'); ylabel('Energy (Watts/sr/nm/m^2)');
title('Display Emission');
axis([min(wavelength) max(wavelength) 0 1.2*max(lightProd)]);
2. Color Gamut and Saturation

Part A:
Using the following linear transformation, we can map the edges of the unit cube in CIE RGB space into a corresponding parallelepiped in CIE XYZ space.

\[
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} =
\begin{bmatrix}
0.490 & 0.310 & 0.200 \\
0.177 & 0.813 & 0.011 \\
0.000 & 0.010 & 0.990
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

The edges of the parallelepiped are drawn in black in the following two figures, which show two different views of the XYZ space. Regions of the color gamut that lie outside the parallelepiped will be clipped/saturated when we restrict RGB values to [0,1]. As we move further away from the origin in XYZ space, the overall luminance increases but the range of colors that we can render without clipping/saturation in the RGB space decreases.

Part B:
The following figures show xy-chromaticity diagrams for \(X + Y + Z = S\), where \(S = 0.5, 1.0, 2.0\). Alongside each chromaticity diagram, we also show binary maps indicating which regions have been clipped/saturated in the R, G, and B values. As S increases, we move further away from the origin in XYZ space and more of the color gamut lies outside the parallelepiped. Consequently, as S increases, more regions in the xy space have RGB values clipped/saturated to 0 or 1. The straight edges of the clipped/saturated regions correspond to intersections of the color gamut solid with the boundaries of the parallelepiped.

Regardless of how small a positive value you substitute for S in \(X + Y + Z = S\), there is a region where R < 0. This corresponds to the negative intensity regime for the \(r(\lambda)\) curve in the CIE RGB color matching functions.


MATLAB Code:

```matlab
% EE368/CS232
% Homework 2
% Problem: Color Gamut and Saturation
% Script by David Chen, Huizhong Chen

clc; clear all; close all;

%% Part A

% Load chromaticity figure
open('Chroma_Solid.fig');
h = gcf; set(gcf, 'Color', 'white'); hold on;

% Define XYZ to RGB mapping
rgb2xyz = [ 0.490 0.310 0.200
0.177 0.813 0.011
0.000 0.010 0.990 ];
xyz2rgb = inv(rgb2xyz);

% Transform boundaries of RGB cube
numSamples = 1000;
RGB = zeros(3,numSamples);
for G = [0 1]
    for B = [0 1]
        RGB(1,:) = linspace(0,1,numSamples);
        RGB(2,:) = G;
        RGB(3,:) = B;
        XYZ = rgb2xyz * RGB;
        scatter3(XYZ(1,:), XYZ(2,:), XYZ(3,:), 5, [0 0 0]);
    end % B
end % G

for R = [0 1]
    for B = [0 1]
        RGB(1,:) = R;
        RGB(2,:) = linspace(0,1,numSamples);
        RGB(3,:) = B;
        XYZ = rgb2xyz * RGB;
        scatter3(XYZ(1,:), XYZ(2,:), XYZ(3,:), 5, [0 0 0]);
    end % B
end % R

for R = [0 1]
    for G = [0 1]
        RGB(1,:) = R;
        RGB(2,:) = G;
        RGB(3,:) = linspace(0,1,numSamples);
        XYZ = rgb2xyz * RGB;
        scatter3(XYZ(1,:), XYZ(2,:), XYZ(3,:), 5, [0 0 0]);
    end % G
end % R

%% Part B

% Load chromaticity polygon
load('Chroma_Map.mat');

% Generate chromaticity diagram and find saturated regions
gamma = 2.2;
scale = size(IN,1);
k = 2;
xVec = linspace(0,1,scale); yVec = xVec;
[yPoly, xPoly] = find(IN == 1);
colorGamut = zeros(scale, scale, 3);
satGamut = zeros(scale, scale, 3);
for nPoint = 1:numel(yPoly)
    disp(sprintf('Processing %d/%d points’, nPoint, numel(yPoly)));
```
xD = xPoly(nPoint);
yD = yPoly(nPoint);
x = xPoly(nPoint)/scale;
y = yPoly(nPoint)/scale;
z = 1 - x - y;

% Find XYZ and RGB, with constraint X+Y+Z=k
XYZ = [x y z] * k;
RGB = reshape(XYZ * xyz2rgb.', [1 1 3]);
for c = 1:3
    if (RGB(c) > 1) || (RGB(c) < 0);
        satGamut(yD,xD,c) = 1;
    end
end
RGB = max(0, min(1, RGB));
RGB_gamma = RGB.^(1/gamma);
% Update buffer
colorGamut(yD,xD,:) = RGB_gamma;
end

% Show results
figure(2); clf; set(gcf, 'Color', 'w');
subplot(1,2,1);
imshow(colorGamut, 'XData', xVec, 'YData', xVec);
set(gca, 'YDir', 'Normal', 'FontSize', 12);
axis on; axis square; axis([0 1 0 1]); hold on;
xlabel('x'); ylabel('y'); title(sprintf('Chromaticity Diagram: X+Y+Z=%.2f', k));
subplot(1,2,2);
imshow(max(satGamut,

[xD yD],3), 'XData', xVec, 'YData', xVec);
set(gca, 'YDir', 'Normal', 'FontSize', 12);
axis on; axis square; axis([0 1 0 1]); hold on;
xlabel('x'); ylabel('y'); title('Saturated Regions');
3. Binarization of Scanned Book Pages

Part A:
Using global thresholding, the regions of the book pages near the spine are binarized to black. As a result, some parts of the text near the spine are no longer visible after binarization. A single global threshold is ineffective against the illumination variations caused by the curved pages.
Part B:
Using locally adaptive thresholding, we can compensate for the uneven illumination. Now, the text near the spine and everywhere else on the page is correctly binarized to black, while the empty spaces on the page are binarized to white.

---

**SEATTLE AND THE ORIENT.**

Storage is practically every state in the Union, and the product is smoked, salted, and canned fish in demand in all of the civilized countries of the world. The total pack of all canned fish reached a total of 384,962 cases, and exceeded by 59,000 cases that of the previous year. The eighteen canneries on Puget Sound packed 274,000 cases, of which amount the five canneries on the Washington side of the river put up 165,000 cases, as follows: 

- Canned Sardines: 120,000
- Canned Tuna: 50,000
- Canned Canned Fish: 20,000
- Canned Herring: 30,000

On the Columbia River the eighteen canneries put up a total of 70,000 cases, of which the forty-nine canneries on the Fraser River put up 38,000 cases, and the northern pack is as follows: 

- Canned Sardines: 20,000
- Canned Tuna: 10,000
- Canned Canned Fish: 5,000
- Canned Herring: 10,000

This shows the Puget Sound canneries packed nearly double the amount of any other locality.

---

**SEATTLE AND THE ORIENT.**

The four places in the city. Eleven steamships make a landing at this place, and have intercourse with all parts of the United States. About their wharf and in their warehouse they give steady employment to thirty people, and during the year they will probably increase this number, at least to the extent of putting representatives on the road to travel in their interest.

ROHLPS & SCHODER.

Seattle has one of the largest banks and office fixtures manufacturing concerns in banks, offices, steamboats and stores and covers a pretty wide range. The firm is one of the best known on Puget Sound, and the fact that their establishment is crowded with orders and their mill is about one of the busiest places in Seattle is evidence of the standing they possess. An illustration is shown here which will give some idea of the size of the buildings they occupy, but to thoroughly appreciate the great amount of business manufactured one is compelled to make a visit through their place.

---

**ONE OF THE BIG MACHINE WORKS.**

One of the largest and best known machine works in the Northwest is the firm of Rohle & Schoder. They are located at 345 to 347 First Avenue South in a building 100 feet by 200 feet in area. They have been established here since 1885 and are really the manufacturers of the Hall & Paulson Furniture Company. At the present time they give employment to forty people and the extent of their trade is very considerable. Their product, as above stated, consists of bank and office fixtures, which makes practically all the manufacturers of coastwise steamboats fixtures, and all kinds of stationary furnishings, such as are used in banks, offices, steamboats and stores and covers a pretty wide range. The firm is one of the best known on Puget Sound, and the fact that their establishment is crowded with orders and their mill is about one of the busiest places in Seattle is evidence of the standing they possess. An illustration is shown here which will give some idea of the size of the buildings they occupy, but to thoroughly appreciate the great amount of business manufactured one is compelled to make a visit through their place.
To perform the locally adaptive thresholding, we use a horizontally sliding window of (21 columns) x (height of the image). Within each window, we compute the local variance. If the local variance exceeds a variance threshold $T_v$ (e.g., $T_v = 0.002$ for gray values in the range $[0,1]$), then we compute a local gray value threshold $T_g$ by Otsu’s method and binarize the column of pixels at the center of the window to white/black. Otherwise, if the local variance is below $T_v$, we have encountered empty space on the page and we binarize the center column to white.

MATLAB Code:

```matlab
% EE368/CS232
% Homework 2
% Problem: Binarization of Book Pages
% Script by David Chen, Huizhong Chen
clc; clear all;

imageFiles = {'Book_Page_1.jpg', 'Book_Page_2.jpg'};
for nImage = 1:length(imageFiles)
    % Load image
    img = im2double(imread(imageFiles{nImage}));
    figure(1); clf;
    imshow(img);
    [height, width] = size(img);
    [pathStr, name, ext] = fileparts(imageFiles{nImage});

    % Global thresholding
    globalThresh = graythresh(img);
    imgBinGlobal = im2bw(img, globalThresh);
    figure(2); clf;
    imshow(imgBinGlobal);
    figure(3); clf;
    set(gca, 'Color', 'w');
    imhist(img); hold on;
    histCounts = imhist(img);
    h = plot(globalThresh*ones(1,100), linspace(0,max(histCounts)), 'r-');
    set(h, 'LineWidth', 2);
    set(gca, 'FontSize', 26);
    h = text(globalThresh+0.01, max(histCounts)/4, sprintf('T = %.2f', globalThresh));
    set(h, 'FontSize', 26);
    ylabel('Frequency');
    imwrite(imgBinGlobal, ['Global_', name, '.jpg']);

    % Locally adaptive thresholding
    imgBinLocal = imgBinGlobal;
    winHalfWidth = 10;
    localVarThresh = 0.002;
    for col = 1:width
        inCols = max(1,(col-winHalfWidth)) : min(width,(col+winHalfWidth));
        inRows = 1:height;
        inTile = img(inRows, inCols);
        localThresh = graythresh(inTile);
        localMean = mean2(inTile);
        localVar = std(inTile(:))^2;
        if localVar > localVarThresh
            imgBinLocal(:,col) = im2bw(img(:,col), localThresh);
        else
            imgBinLocal(:,col) = 1;
        end
    end
    figure(4); clf;
    imshow(imgBinLocal);
    imwrite(imgBinLocal, ['Local_', name, '.jpg']);
end
if nImage == 1
    pause
end
```
end % nImage
4. Traffic Cone Detection

Part A:
Below, we show some training samples of non-cone pixels in blue and cone pixels in red. There is some clear separation between these two classes in RGB space, so a multidimensional MAP detector in RGB space can be effective at distinguishing between the two classes.

Part B:
Using a multidimensional MAP detector in RGB space with 16x16x16 uniformly spaced bins, approximately 4 percent of the bins belong to the cone class.

Part C:
Below, we show the results of applying the MAP detector on the two testing images. The MAP detector performs well at selecting the cone pixels while rejecting non-cone pixels. Small positive regions (false positives) and small negative regions (false negatives) can be removed by small region removal as a post-processing step.
MATLAB Code:

```matlab
% EE368/CS232
% Homework 2
% Problem: Cone Detection
% Script by David Chen, Huizhong Chen

%% Load training images
numTraining = 5;
for nImage = 1:numTraining
    imgTrain{nImage} = double(imread(['hw2_cone_training_'
    num2str(nImage) '.jpg']));
    imgTrainMask{nImage} = imread(['hw2_cone_training_map_'
    num2str(nImage) '.png']);
    imgTrainMask{nImage} = imgTrainMask{nImage} > 0;
end

%% Train MAP detector
step = 16;
centroids1D = step/2 : step : 256;
numCentroids1D = numel(centroids1D);
countsCone3D = zeros(numCentroids1D, numCentroids1D, numCentroids1D);
countsNonCone3D = zeros(numCentroids1D, numCentroids1D, numCentroids1D);
for nImage = 1:numTraining
    disp(['Sampling training image ' num2str(nImage)]); pause(0.5);
    [height, width, channels] = size(imgTrain{nImage});
    for y = 1:height
        for x = 1:width
            pix = imgTrain{nImage}(y, x, :);
            for c = 1:3
                [minDist, idxRGB(c)] = min(abs(pix(c) - centroids1D));
                end % c
                if imgTrainMask{nImage}(y,x) == 1
                    N = countsCone3D(idxRGB(1), idxRGB(2), idxRGB(3));
                    countsCone3D(idxRGB(1), idxRGB(2), idxRGB(3)) = N + 1;
                else
                    N = countsNonCone3D(idxRGB(1), idxRGB(2), idxRGB(3));
                    countsNonCone3D(idxRGB(1), idxRGB(2), idxRGB(3)) = N + 1;
                end % else
            end % x
        end % y
    end % nImage
    centroidClass = countsCone3D > countsNonCone3D;
```

14
save('MAP_training.mat', 'centroidClass', 'countsCone3D', 'countsNonCone3D');

%% Plot training samples
% Collect samples
numConeSamples = 0;
numNonConeSamples = 0;
for nImage = 1:numTraining
    numConeSamples = numConeSamples + numel(find(imgTrainMask{nImage} == 1));
    numNonConeSamples = numNonConeSamples + numel(find(imgTrainMask{nImage} == 0));
end % nImage
RGBCone = zeros(numConeSamples,3);
RGBNonCone = zeros(numNonConeSamples,3);
counterCone = 1;
counterNonCone = 1;
skipY = 10;
skipX = 10;
for nImage = 1:5
    disp(sprintf('Processing training image %d', nImage));
    [height, width, channels] = size(imgTrain{nImage});
    for y = 1:skipY:height
        for x = 1:skipX:width
            pix = imgTrain{nImage}(y,x,:);
            if (imgTrainMask{nImage}(y,x) == 1)
                RGBCone(counterCone,:) = imgTrain{nImage}(y,x,:);
                counterCone = counterCone + 1;
            else
                RGBNonCone(counterNonCone,:) = imgTrain{nImage}(y,x,:);
                counterNonCone = counterNonCone + 1;
            end
        end
    end
end % end x
end % end y
end % nImage
RGBCone = RGBCone(1:counterCone-1,:);
RGBNonCone = RGBNonCone(1:counterNonCone-1,:);

% Plot samples
figure(1); clf; set(gcf, 'Color', 'w');
scatter3(RGBCone(:,1), RGBCone(:,2), RGBCone(:,3), 10, [1 0 0]); hold on;
scatter3(RGBNonCone(:,1), RGBNonCone(:,2), RGBNonCone(:,3), 10, [0 0 1]); hold on;
set(gca, 'XTick', 0:32:256, 'YTick', 0:32:256, 'ZTick', 0:32:256);
set(gca, 'FontSize', 16);
xlabel('R'); ylabel('G'); zlabel('B'); axis square; axis([0 256 0 256 0 256]);

%% Load test images
numTest = 2;
for nImage = 1:numTest
    imgTest{nImage} = double(imread(['hw2_cone_testing_' num2str(nImage) '.jpg']));
end % nImage

% Apply MAP detector
load('MAP_training.mat');
for nImage = 1:numTest
disp(['Applying MAP detector to test image ' num2str(nImage)]); pause(0.5);
    [height, width, channels] = size(imgTest{nImage});
    imgClassify{nImage} = zeros(height, width);
    for y = 1:height
        for x = 1:width
            pix = imgTest{nImage}(y,x,:);
            for c = 1:3
                [minDist, idxRGB(c)] = min(abs(pix(c) - centroids1D));
            end % c
            pixClass = centroidClass(idxBG(1), idxBG(2), idxBG(3));
            imgClassify{nImage}(y,x) = pixClass;
        end % x
    end % y
end % nImage
save('MAP_test.mat', 'imgClassify');

%% Clean up MAP detector result with region labeling
load('MAP_test.mat');
for nImage = 1:numTest
    disp(['Post processing test image ' num2str(nImage)]); pause(0.5);
    imgClassifyPost{nImage} = imgClassify{nImage};
    % Remove small positive regions
    [imgLabels, numLabels] = bwlabel(imgClassifyPost{nImage}, 8);
    for label = 1:numLabels
        idx = find(imgLabels == label);
        if numel(idx) < 200
            imgClassifyPost{nImage}(idx) = 0;
        end
    end
    % Remove small negative regions
    [imgLabels, numLabels] = bwlabel(~imgClassifyPost{nImage}, 8);
    for label = 1:numLabels
        idx = find(imgLabels == label);
        if numel(idx) < 100
            imgClassifyPost{nImage}(idx) = 1;
        end
    end
end % nImage
save('MAP_post.mat', 'imgClassifyPost');

%% Show results
load('MAP_test.mat'); load('MAP_post.mat');
figure(1); clf;
for nImage = 1:numTest
    subplot(numTest,3,3*nImage-2);
    imshow(uint8(imgTest{nImage}));
    set(gca, 'FontSize', 16');
    title('Original Image');
    subplot(numTest,3,3*nImage-1);
    imshow(imgClassify{nImage});
    set(gca, 'FontSize', 16');
    title('MAP Detector');
    subplot(numTest,3,3*nImage);
    imshow(imgClassifyPost{nImage});
    set(gca, 'FontSize', 16');
    title('MAP Detector + Small Region Removal');
end % nImage
**Bonus: Color Balancing**

Part A: 
In the following figures, we show the original images and the color-balanced images using the gray-world, scale-by-max, and shades-of-gray algorithms. All three algorithms perform comparably for Macbeth. Gray-world generates an unnatural color appearance for Leaf, because the gray-world assumption is not fulfilled for the Leaf image.
<table>
<thead>
<tr>
<th></th>
<th>Macbeth</th>
<th>Leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gray-world</td>
<td>$k_r = 1.15, \ k_g = 1.00, \ k_b = 1.43$</td>
<td>$k_r = 2.19, \ k_g = 1.00, \ k_b = 3.55$</td>
</tr>
<tr>
<td>Scale-by-max</td>
<td>$k_r = 1.12, \ k_g = 1.00, \ k_b = 1.13$</td>
<td>$k_r = 1.16, \ k_g = 1.00, \ k_b = 1.09$</td>
</tr>
<tr>
<td>Shades-of-gray ($p = 6$)</td>
<td>$k_r = 1.08, \ k_g = 1.00, \ k_b = 1.33$</td>
<td>$k_r = 1.54, \ k_g = 1.00, \ k_b = 1.57$</td>
</tr>
</tbody>
</table>

Part B:
In the following figures, we show the original images and the color-balanced images after using the shades-of-gray algorithm for different values of $p$. Also shown are the histograms of the R, G, and B channels. For $p = 1$, the means of the RGB channels must match after color balancing. In the case of Leaf, the B histogram must be stretched severely to meet this criterion. In contrast, for larger values of $p$, the $p$-th norm of the RGB channels must match after color balancing. For Leaf, meeting this criterion requires a much less severe distortion to the B histogram and better preserves the natural color appearance in the Leaf image.
MATLAB Code:

```matlab
clc; clear all; close all;

%% Part 1: Develop 3 automatic color balancing algorithms:
%% 1. Gray-world
%% 2. Scale-by-max
%% 3. Shades-of-gray
%% Test each of these algorithms on two images
%% 1. Macbeth picture
%% 2. Leaf picture
img_names = {'data/macbeth.jpg', 'data/leaf.jpg'};
gamma = 2.2;
for j=1:length(img_names)
    imgs{j} = (double(imread(img_names{j}))/255).^(gamma);
end

%% Display Results
for j=1:length(imgs)
    for i=1:length(algorithms)
        figure(j); set(gcf, 'Color', 'w');
        subplot(2,2,i); imshow((imgs_balanced{j,i}).^(1/gamma)); title(algorithms{i});
    end
end
```

```matlab
%% Part 2: Explore causes of gray-world algorithm failure
%% calculate the histogram after the completion of each algorithm
% algorithms = {'Original', 'Gray-world', 'Shades-of-gray'};
% k = cell(size(imgs_balanced),length(algorithms));
in Matlab code:
```
```matlab
subplot(2,length(algorithms),i);
imshow((imgs_balanced_shades{j,i}).^\{(1/gamma)}); title(titles{i});
subplot(2,length(algorithms),length(algorithms)+i);
hold on;
for c=1:3,
    bar(bins,hist_balanced_shades{j,i,c},...
        'EdgeColor',colors{c}); axis([-1.1 1.1 0 10^4]);
end
end
end
```