CS 477 HW #7

Questions completed: All undergrad level questions

Programming/Deliverables: MATLAB

Estimated time for assignment: 14 hours

A. Understanding the color constancy problem.

1. Estimate of the illuminant color for macbeth_syl image.

   First, we click points on the white patch of the image to get the coordinates of a rectangle that is inside the white patch. MATLAB code for this part is shown below.

```
macbeth_syl = imread('color_constancy_images/macbeth_syl-50MR16Q.tif');
figure;
datacursormode on;
imagesc(macbeth_syl);
```

   Then, we collect the RGB of the pixels in our rectangle. MATLAB code for this part is shown below.

```
white_patch_macbeth_syl = [];
for i = 325:380
    for j = 95:148
        rgb = squeeze(macbeth_syl(i,j,:))';
        white_patch_macbeth_syl = [white_patch_macbeth_syl; rgb];
    end
end
```

   Next, we take the average RGB of these collected pixels and then compute a single scale factor so that the max color channel value is 250. Finally, we multiply the computed average RGB by our scale factor to get our estimate of the illuminant color. MATLAB code for this part is shown below.

```
RGB_SW = mean(white_patch_macbeth_syl)
s = 250/max(RGB_SW)
RGB_SW = round(RGB_SW*s)
```

The resulting color of our illuminant is [238, 220, 250] in RGB.

2. Estimate of the illuminant color for macbeth_solux image.

   We repeat the same steps as described in part A1. MATLAB code for this shown below.

```
macbeth_solux = imread('color_constancy_images/macbeth_solux-4100.tif');
figure;
datacursormode on;
imagesc(macbeth_solux);

white_patch_macbeth_solux = [];
for i = 325:380
    for j = 95:148
        rgb = squeeze(macbeth_solux(i,j,:))';
        white_patch_macbeth_solux = [white_patch_macbeth_solux; rgb];
    end
end

RGB_UW = mean(white_patch_macbeth_solux)
```

```
s = 250/max(RGB_UW)

RGB_UW = round(RGB_UW*s)
```

The resulting illuminant is [132, 159, 250] in RGB.

3. Angular error between two light colors.

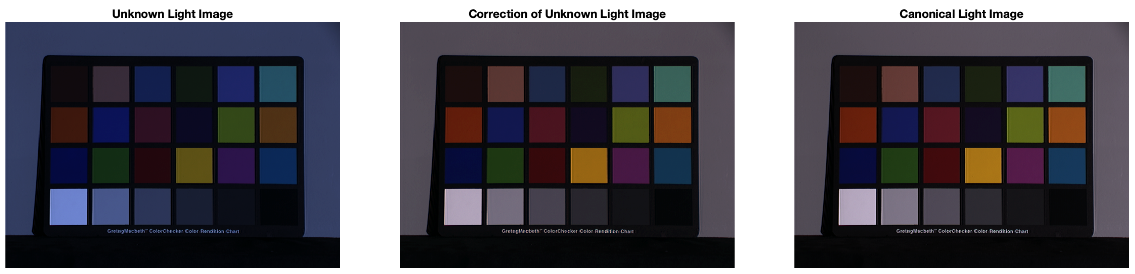
   The angular error was computed in MATLAB using the code shown below.

   ```
   angular_error = acosd(dot(RGB_UW, RGB_SW)/(norm(RGB_SW)*norm(RGB_UW)))
   ```

   The code above computes the angular error as described in the assignment. The angular error turned out to be 13.84°.

4. Remapping image under canonical light.

   Below in Figure 1, I provide the image triplet of the original image, the improved image, and the canonical image.



(a) Original (blueish) image. (b) Improved (corrected) image. (c) Canonical image.

Figure 1: Image triplet generated in MATLAB showing the original image (a) compared to its corrected version (b) and the canonical image (c). The corrected image (b) was produced using the diagonal model computed from the illuminant RGB found in problems 1) and 2) (as described in the slides)to map the original (blueish) image (a) to the one under the canonical light (c). The brightness of the images were scaled by a constant factor for each image (based on the max RGB for that image) so as to reduce confusion about differences between the images being due to brightness instead of chromaticity (color without brightness).

Looking at the corrected image (b) versus the canonical image (c) above in Figure 1, it appears that using the diagonal model provides a pretty good correction of the original image (a). However, closely comparing the brighter tiles in (b) and (c) such as the orange and yellow tiles, one can see that (c) is a bit brighter overall (at least with the manual scaling of the brightness) than (b).

5. RMS error of the chromacity coordinates.

   To compute RMS error, filter out dark pixels defined by $R+G+B<10$ as requested in the assignment. The MATLAB code for this is shown below.

```
A_solux = zeros(290525,2);
A_syl = zeros(290525,2);
B_corrected = zeros(291280,2);
B_syl = zeros(291280,2);
x = 1;
y = 1;
for i = 1:size(macbeth_syl,1)
    for j = 1:size(macbeth_syl,2)
        RGB_syl = sum(macbeth_syl_scaled(i,j,:));
        RGB_solux = sum(macbeth_solux_scaled(i,j,:));
        RGB_corrected = sum(macbeth_corrected_scaled(i,j,:));
        R_syl = macbeth_syl_scaled(i,j,1);
        G_syl = macbeth_syl_scaled(i,j,2);
        R_solux = macbeth_solux_scaled(i,j,1);
        G_solux = macbeth_solux_scaled(i,j,2);
        R_corrected = macbeth_corrected_scaled(i,j,1);
        G_corrected = macbeth_corrected_scaled(i,j,2);
        if RGB_solux >= 10 && RGB_syl >= 10
            r_solux = double(R_solux)/double(RGB_solux);
            g_solux = double(G_solux)/double(RGB_solux);
            r_syl = double(R_syl)/double(RGB_syl);
            g_syl = double(G_syl)/double(RGB_syl);
            A_solux(x,:) = [r_solux, g_solux];
            A_syl(x,:) = [r_syl, g_syl];
            x = x + 1;
        end
        if RGB_corrected >= 10 && RGB_syl >= 10
            r_corrected = double(R_corrected)/double(RGB_corrected);
            g_corrected = double(G_corrected)/double(RGB_corrected);
            r_syl = double(R_syl)/double(RGB_syl);
            g_syl = double(G_syl)/double(RGB_syl);
            B_corrected(y,:) = [r_corrected, g_corrected];
            B_syl(y,:) = [r_syl, g_syl];
            y = y + 1;
        end
    end
end
```

The code above seems very verbose, but all it is doing is filtering out the dark pixels and computing the r and g values as specified in the assignment.

Now that we have r and g, we compute the RMS error in terms of the chromaticity coordinates (r, g) as defined in the assignment. The MATLAB code for this is shown below.

```
d_A = sqrt(sum((A_solux-A_syl).^2, 2));
RMSE_A = sqrt(mean(d_A.^2))


d_B = sqrt(sum((B_corrected-B_syl).^2, 2));
RMSE_B = sqrt(mean(d_B.^2))
```

In the code shown above, the error between two pixels in chromaticity coordinates is computed as the Euclidean distance between two (r, g) vectors. The RMS error is then the square root of the average of the squared error values.

For part A (original (blueish) versus canonical), the RMSE was found to be 0.1149. For part B (corrected versus canonical), the RMSE was found to be 0.0475. These results make sense because you would expect the *corrected* image to have a lower RMSE in relation to the canonical image than the RMSE of the *original* image in relation to the canonical image.

6. Estimating light color for remaining images using MaxRGB algorithm.

   Estimates were computed using the MaxRGB algorithm described in the slides, and the angular error was computed the same way as in problem 3. Note that the estimated light colors are scaled so that the R, G, or B value is 250 as was done when finding the light color in problems 1 and 2.

   For apples2, the light color estimate is $(182, 228, 250)$, and the angular error in relation to the macbeth solux light is $9.80°$.

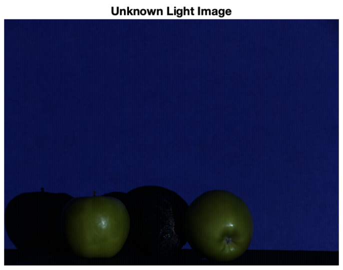
   For ball, the light color estimate is $(153, 177, 250)$, and the angular error in relation to the macbeth solux light is $3.57°$. For ball solux specifically, there seemed to be a camera artifact in the form of a vertical, white line located at the bottom middle of the image. I chose to this filter out this white line as it could interfere with the

MaxRGB algorithm. I simply made it so that the MATLAB "max()" function could not search below a certain point on the image as to avoid the whitest parts of the white line artifact. This is not a perfect method as we cannot scan the rest of the ball which has white parts around the level of the white line artifact. However, this method was simple and effective as it reduced the angular error from 3.66 (originally when the whole image was scanned for the max) down to the current reported value of 3.57. Note that this should not be done for the gray-world method which relies on averaging all the pixel values.

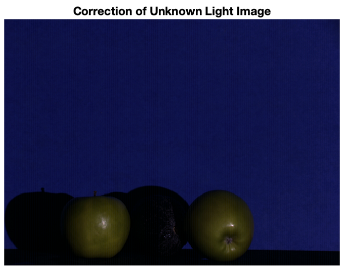
For blocks1, the light color estimate is $(250, 242, 209)$, and the angular error in relation to the macbeth solux light is $19.95°$.

The angular errors in relation to the macbeth solux light are consistent with the MaxRGB algorithm assumption that each channel has an albedo somewhere in the image that is the same as white. In apples2, you have shiny parts of the apple reflecting back light which can serve as white albedo hence the second lowest angular error. In ball, you have a part of the ball surface that is white hence producing the lowest angular error. Finally, in blocks, there is no white or shiny surface hence producing the highest angular error out of the three.

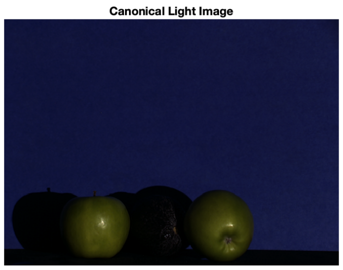
7. Displaying results as one figure and reporting RMS error.



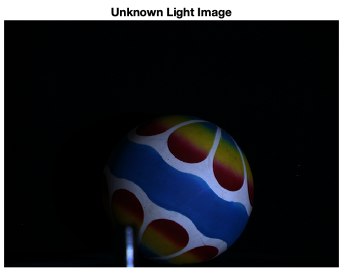
(a) Original apple image.



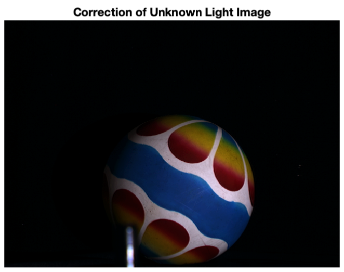
(b) Corrected apple image.



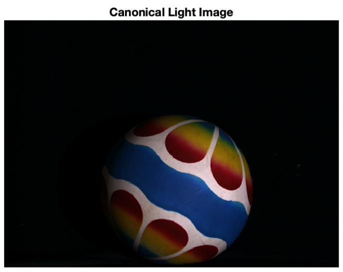
(c) Canonical apple image.



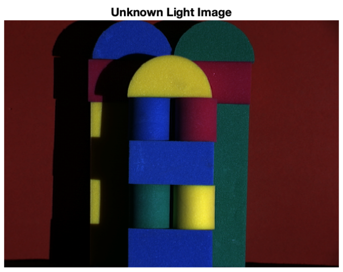
(d) Original ball image.



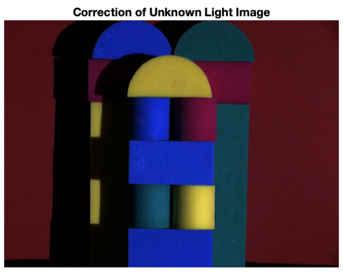
(e) Corrected ball image.



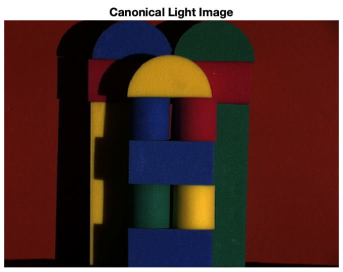
(f) Canonical ball image.



(g) Original blocks image.



(h) Corrected blocks image.



(i) Canonical blocks image.

Figure 2: Image triplets of remaining solux images generated in MATLAB. For each scene (row in the table), I followed the same process as in problem (4) for the macbeth image. Also as in problem (4), I have scaled the image brightness so that the max R,G, or B value is 250.

The RMS errors for the corrected images in relation to their respective canonical image counterparts is reported as an output from MATLAB below.

```
RMSE_apples =
    0.0608

RMSE_ball =

    0.0494

RMSE_blocks =

    0.0984
```

The ranking of the RMS errors (from lowest to highest) does indeed reflect the ranking of the angular error for the scenes. This is intuitive because the angular error tells us how far off our unknown light color estimate for each scene via the MaxRGB algorithm was from the "true" color estimate of our unknown light from the macbeth solux image. A higher degree of angular error tells us our unknown light color estimate was further off from the real unknown light color so the diagonal model correction mapping should have a higher (r, g) RMS error as well.

8. Repeating error computations using gray-world method.

   The estimated illuminating color and the angular error (still in degrees) for each scene using the gray-world method is reported as a MATLAB output below.

```
RGB_apples =

   49   78  250

angular_error_apples =

   19.62


RGB_ball =

  129  167  250

angular_error_ball =

    1.39
```

```
RGB_blocks =

  250  154  180

angular_error_blocks =

   23.43
```

The RMS error using the gray-world method is reported as a MATLAB output below.

```
RMSE_apples =

    0.1382

RMSE_ball =

    0.0554

RMSE_blocks =

    0.1514
```

Based on the above reporting, it appears that the MaxRGB algorithm is working better on this dataset based on higher RMS errors across the board and higher angular errors for the apples and blocks scene but a lower angular error for the ball scene. How is it that for the ball scene, the gray-world angular error can be lower than the MaxRGB angular error, but, despite that, the gray-world RMS error is higher than the MaxRGB RMS error? My understanding is that the diagonal model is directly reliant on the color obtained from the method used (gray-world or MaxRGB), so I am stumped on this. Perhaps it is a calculation error on my part, or that by some convention of the computed ratios in the diagonal model, you could experience a higher RMS error depending on the weight of the ratios of each channel despite a closer (r, g) color vector to the real color vector.