Homework 5

Tasks:

1. Explain what a steerable filter is and apply it in MATLAB to both "Migrant Mother Download Migrant Mother" and "Penn State Download Penn State" images.

Answer ==> An orientation selective filter that is formed using the combination of small, fixed set of 'basis filters' is called a steerable filter. The idea behind steerable filter is to first process the image through simple filters or basis filters and then determine the response to any orientation by weighing the initial response to basis filters and summing them together. This removes the need to create filters for each orientation and creates an efficient method for edge detection, texture analysis and for photometric stereo applications.
Reference - https://en.wikipedia.org/wiki/Steerable_filter

MATLAB code -

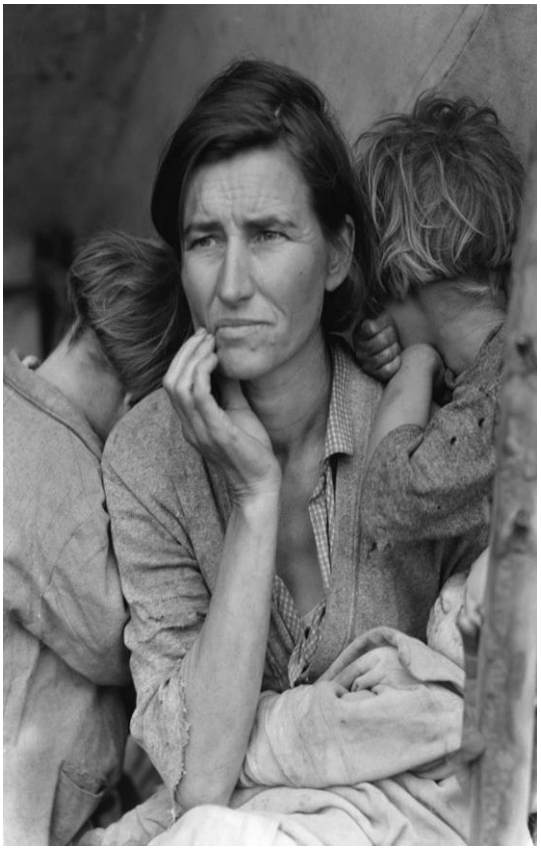
```
%%
close all; clear; clc;
% Step 1
% Reading image into 2 separate variables
I1 = imread("L01 Migrant Mother.png");
I2 = imread("L01 greatvalley.jpg");
% Step 2
% Setting Filter Parameters
sigma = 2; % Standard deviation of the Gaussian
sz = 45; % Filter size (higher numbers needed more computation time)
[x, y] = meshgrid(-floor(sz/2):floor(sz/2), -floor(sz/2):floor(sz/2));
% Step 3
% Creating Basis Filters (Derivatives of 2D Gaussian Function)
% Gx: Derivative with respect to x
Gx = -(x / (2 * pi * sigma^4)) .* exp(-(x.^2 + y.^2) / (2 * sigma^2));
% Gy: Derivative with respect to y
Gy = -(y / (2 * pi * sigma^4)) .* exp(-(x.^2 + y.^2) / (2 * sigma^2));
% Step 4
% Formatting images as grayscale double for processing
if size(I1, 3) == 3
I1_gray = rgb2gray(I1); % Convert to grayscale if it's not
else
I1_gray = I1;
end
if size(I2, 3) == 3
```

```
I2_gray = rgb2gray(I2); % Convert to grayscale if it's not
else
I2_gray = I2;
end
I1_gray = im2double(I1_gray); % Convert to double for convolution
I2_gray = im2double(I2_gray);
% Step 5
% Steering the Filter to 45 Degree Angle
theta = deg2rad(45); % Angle to steer (45 degrees)
% Interpolation functions
k_x = cos(theta);
k_y = sin(theta);
% Linear combination to create the steered filter
G_theta = k_x * Gx + k_y * Gy;
% Step 6
% Applying the steered filter using convolution to both images
filtered_I1 = conv2(I1_gray, G_theta, 'same'); % Keep output img size same
filtered_I2 = conv2(I2_gray, G_theta, 'same'); % Keep output img size same
% Step 7
% Displaying Results
figure;
subplot(2, 2, 1);
imshow(I1);
title('Original Image 1');
subplot(2, 2, 2);
imshow(filtered_I1, []); % [] to scale display range
title(['Image 1 Filtered at ' num2str(rad2deg(theta)) ' Degrees']);
subplot(2, 2, 3);
imshow(I2);
title('Original Image 2');
subplot(2, 2, 4);
imshow(filtered_I2, []); % [] to scale display range
title(['Image 2 Filtered at ' num2str(rad2deg(theta)) ' Degrees']);
% Step 8
% Save the filtered images
imwrite(im2uint8(mat2gray(filtered_I1)), 'filtered_image1.png');
imwrite(im2uint8(mat2gray(filtered_I2)), 'filtered_image2.png');
```

Output -

Image 1 – Migrant Mother

Original



Steering Filter (steered at 45 degrees)

Image 2 – Great Valley

Original

Steering Filter (steered at 45 degrees)

2. Solve problems 3.1 and 3.12 from the Szeliski textbook.

Problem 3.1:

Ex 3.1 (Color balance) Write a simple application to change the color balance of an image by multiplying each color value by a different user-specified constant. If you want to get fancy, you can make this application interactive with sliders.

1. Do you get different results if you take out the gamma transformation before/after doing the multiplication? Why or why not?
2. Take the same picture with your digital camera using different color balance settings (most cameras control the color balance from one of the Menus). Can you recover what the color balance ratios are between the different settings? You may need to put your camera on a tripod and/or align the images manually or automatically to make this work. Alternatively, use a color checker chart (Figure 10.3b), as discussed in x2.3 and x10.1.1.
3. If you have access to the RAW image for the camera, perform the demosaicing yourself x10.3.1, or downsample the image resolution to get a "true" RGB image. Does your camera perform a simple linear mapping between RAW values and the color-balanced values in a JPEG? Some high end cameras have a RAW+JPEG mode, which makes this comparison much easier.
4. Can you think of any reason why you might want to perform a color twist x3.1.2 on the images? See also Exercise 2.8 for some related ideas.

Answer ==>

MATLAB code for color balance -

```
%%
% Loading the sample peppers image
img = imread("peppers.png");
img_double = im2double(img); % Converting to double for multiplications
% Color balancing by multiplying values to individual channels
% Original image seems warm
% So, correcting for a yellowish tint by boosting blue and reducing red.
r_mult = 0.85; % Reduce the red channel
g_mult = 1.0; % Keep green the same
b_mult = 1.15; % Boost the blue channel
% Applying color balance
balanced_img = img_double; % Create a copy to modify
% Apply the multiplication to each channel
balanced_img(:, :, 1) = balanced_img(:, :, 1) * r_mult; % Red channel
balanced_img(:, :, 2) = balanced_img(:, :, 2) * g_mult; % Green channel
balanced_img(:, :, 3) = balanced_img(:, :, 3) * b_mult; % Blue channel
% Clip the values so they stay within the valid [0, 1] range
balanced_img = clip(balanced_img, 0, 1);
```

```
% Displaying the Results
figure;
subplot(1, 2, 1);
imshow(img);
title('Original Image');
subplot(1, 2, 2);
imshow(balanced_img);
title('Color Balanced Image');
% Save the color balanced image
% Ensure the output image is in the correct format for saving
balanced_img = im2uint8(balanced_img);
imwrite(img, 'peppers.png');
imwrite(balanced_img, 'color_balanced_peppers.png');
```

Original image

Color balanced image

1. Do you get different results if you take out the gamma transformation before/after doing the multiplication? Why or why not?

1. Answer ==> Gamma correction makes the images brighter or darker through a power function on the original pixel value. If color balancing is carried out on an image after gamma correction is completed on the same image, it would not provide accurate colors as the non-linear function of gamma correction has changed the intensities of colors. If the reverse is carried out – color balancing first and then gamma correction, then the colors would appear accurate as the color balancing corrects them and the gamma correction just changes the intensity (makes them darker or brighter).

```
% Perform gamma correction and then color balancing
% Apply gamma correction with a gamma value of 1.5
gamma = 2.0;
gamma_corrected_img = imadjust(img_double, [], [], gamma);
% Apply color balancing on the gamma corrected image
gamma_col_bal_img = gamma_corrected_img;
% Apply the multiplication to each channel of the gamma corrected image
gamma_col_bal_img(:, :, 1) = gamma_col_bal_img(:, :, 1) * r_mult; % Red channel
gamma_col_bal_img(:, :, 2) = gamma_col_bal_img(:, :, 2) * g_mult; % Green channel
gamma_col_bal_img(:, :, 3) = gamma_col_bal_img(:, :, 3) * b_mult; % Blue channel
% Save the gamma color balanced image
gamma_col_bal_img = im2uint8(gamma_col_bal_img);
imwrite(gamma_col_bal_img, 'gamma_color_balanced_peppers.png');
% Perform color balancing and then gamma correction
% Apply gamma correction on the color balanced image
col_bal_gamm_corr_img = imadjust(balanced_img, [], [], gamma);
% Save the color balanced gamma corrected image
col_bal_gamm_corr_img = im2uint8(col_bal_gamm_corr_img);
imwrite(col_bal_gamm_corr_img, 'gamma_corrected_balanced_peppers.png');
```

Gamma corrected color balanced image

Color balanced gamma corrected image

2. Take the same picture with your digital camera using different color balance settings (most cameras control the color balance from one of the Menus). Can you recover what the color balance ratios are between the different settings? You may need to put your camera on a tripod and/or align the images manually or automatically to make this work. Alternatively, use a color checker chart (Figure 10.3b), as discussed in x2.3 and x10.1.1.

2. Answer ==> To retrieve the color balance ratios of an image, we can calculate the color intensity across each channel by first summing all the color intensities across all pixels in the image and then dividing it by the pixel count. An example of the same is given below:

```
% Determining color balance ratios of an image
img = imread('yoga.jpg');
colorBalance = sum(img, [1, 2]) ./ numel(img(:,:,1));
% Display the color balance ratios
disp('Color Balance Ratios:');
disp(colorBalance);
```

```
Color Balance Ratios:

(:,:,1) = 145.4072
(:,:,2) = 140.0970
(:,:,3) = 140.0423
```

Image - 

3. If you have access to the RAW image for the camera, perform the demosaicing yourself x10.3.1, or downsample the image resolution to get a “true” RGB image. Does your camera perform a simple linear mapping between RAW values and the color-balanced values in a JPEG? Some high end cameras have a RAW+JPEG mode, which makes this comparison much easier.

3. Answer ==> If the operations are performed manually – demosiacing and then downsampling, then those operations are linear operations. But when the camera does it, it might include gamma correction as well for standard displays (sRGB). Gamma correction is a non-linear operation and therefore the conversion from RAW to JPEG by camera is not a linear operation, but the manual steps asked here are linear in nature.

4. Can you think of any reason why you might want to perform a color twist x3.1.2 on the images? See also Exercise 2.8 for some related ideas.

4. Answer ==> Standard color twist helps correct the camera’s native color space to a standard sRGB color space, correct any color grading, and any sensor corrections. This is done by a complete 3x3 matrix multiplication that each camera model has built within it, and companies provide new firmware updates on that as well. Regarding the question at hand, the reason why I would like to perform a color twist is to correct for the source of light used in taking the picture. The light source can change the way an image looks to us and correcting it for white balance would display the image true to reality.

Ex 3.12 (Steerable filters) Implement Freeman and Adelson's (1991) steerable filter algorithm.

The input should be a grayscale or color image, and the output should be a multi-banded image consisting of G01 and G901 . The coefficients for the filters can be found in (Freeman and Adelson1991). Test the various order filters on a number of images of your choice, and see if you can reliably find corner and intersection features. These filters will be quite useful later to detect elongated structures such as lines x4.3.

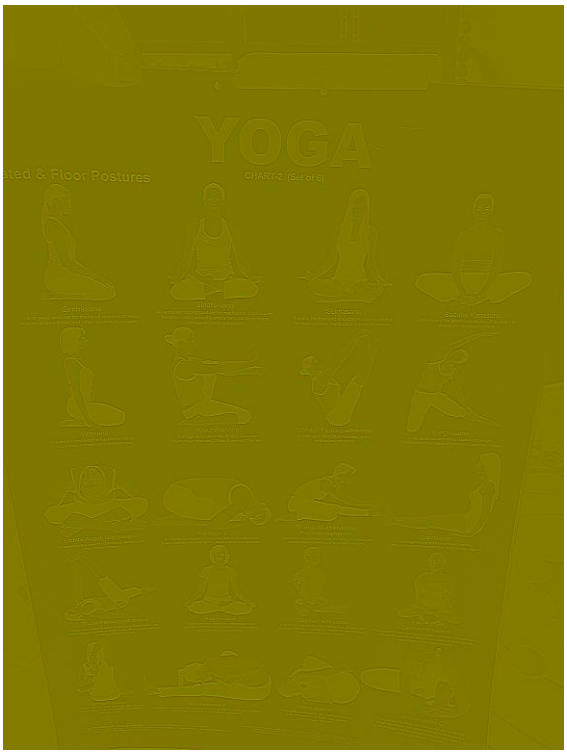
Answer ==> The problem mentions that the intention is to reliably find corner and intersection features which is something that is found through the 2^{nd} order derivative of a Gaussian function whereas the first order finds edges. So, we would create 2^{nd} order derivative basis functions first and then add steering to them. Through the code we can identify corner and intersection features in the given image.

```
% Perform Freeman and Adelson steering filter algorithm
% Loading image
img = imread("yoga.jpg");
% Convert to grayscale and double format for processing
if size(img, 3) == 3
img = rgb2gray(img);
end
img = im2double(img);
% Defining Basis Filters (Second Derivatives of Gaussian)
sigma = 2.0; % Standard sigma of Gaussian
sz = ceil(sigma * 6); % To cover 3 sigma on each side
if mod(sz, 2) == 0, sz = sz + 1; end % Ensure filter size is odd
[x, y] = meshgrid(-floor(sz/2):floor(sz/2), -floor(sz/2):floor(sz/2));
% Gaussian component
G = exp(-(x.^2 + y.^2) / (2 * sigma^2));
% Second partial derivatives
Gxx = (x.^2 / sigma^4 - 1 / sigma^2) .* G / (2*pi*sigma^2);
Gyy = (y.^2 / sigma^4 - 1 / sigma^2) .* G / (2*pi*sigma^2);
Gxy = (x .* y / sigma^4) .* G / (2*pi*sigma^2);
% Convolving Image with Basis Filters to first find their responses
Rxx = imfilter(img, Gxx, 'replicate', 'conv');
Ryy = imfilter(img, Gyy, 'replicate', 'conv');
Rxy = imfilter(img, Gxy, 'replicate', 'conv');
% Steering the Response to 0 and 90 Degrees
R0 = (cosd(0))^2*Rxx + (sind(0))^2*Ryy + 2*sind(0)*cosd(0)*Rxy;
R90 = (cosd(90))^2*Rxx + (sind(90))^2*Ryy + 2*sind(90)*cosd(90)*Rxy;
% Displaying the Results ---
```

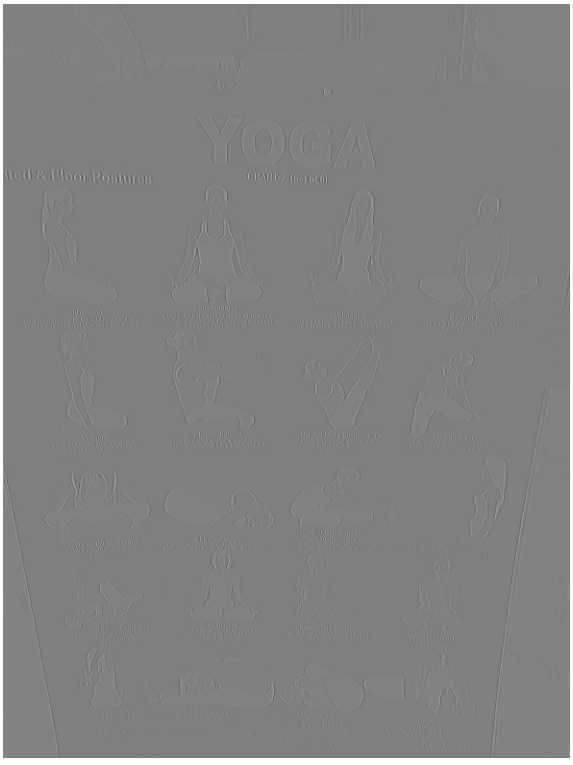
```
figure;
subplot(2, 2, 1);
imshow(img);
title('Original Image');
subplot(2, 2, 2);
imshow(R0, []); % [] to scale the display range
title('Output G0_1 (Response at 0 degrees)');
subplot(2, 2, 3);
imshow(R90, []);
title('Output G90_1 (Response at 90 degrees)');
% Create a combined multi-band image
multi_band_output = cat(3, R0, R90);
% To visualize the third "band", another channel of zeros is added
multi_band_display = cat(3, mat2gray(R0), mat2gray(R90), zeros(size(R0)));
subplot(2, 2, 4);
imshow(multi_band_display);
title('Combined Output (R=G0, G=G90)');
% Save all the generated images to files
imwrite(im2uint8(mat2gray(R0)), 'Freeman_0_degrees.png');
imwrite(im2uint8(mat2gray(R90)), 'Freeman_90_degrees.png');
imwrite(im2uint8(multi_band_display), 'Freeman_multi_band_output.png');
```

Original image

Multi band image



Response at 0 degrees



Response at 90 degrees