Ilke Kas
Ixk238
CSDS490

Homework 1

Problem 1 (G&W 2.7)

An automobile manufacturer is automating the placement of decorative inlays on the bumpers of a limited-edition line of sports cars. The decorations are color-coordinated, so the assembly robots need to know the color of each car in order to select the appropriate bumper component. Models come in only four colors: blue, green, red, and white. You are hired to propose a solution based on imaging (space does not permit the construction of four different assembly lines, and synchronizing the cars leaving the spray-paint area would slow down the whole process). How would you solve the problem of determining the color of each car, keeping in mind that cost is the most important consideration in your choice of components?

Answer: To solve this problem, I need two of these filters: red filter, green filter, or blue filter and one monochrome camera. Let's say we chose red and green filters. I would put the filters in front of the camera in a way that half of the camera lens will be covered with a red filter while the other half is covered with the green filter. Then I would focus the camera on the car in a way that image can be seen as flat. Let's think about this setting:

- As we know, filters only pass the colors that have the same color as them. For example, if we have a red filter, it only lets red light to pass through the filter while it is blocking the blue and green lights. That is why if a red object is behind a red filter, we can see the object. However, if we put a green object behind a red filter we cannot see the color of the green object. We see it black (dark).
- As we know, monochrome cameras can only produce grayscale, colorless images.

Let's think about the car possibilities under this setting:

- If the car is green, the green filter passes the green light (reflected from the car) but the red filter does not. The Monochrome camera catches the grayscale gray/or white (high intensity value) from green filter half but it will catch black from the red filter half. Then, we can detect the color of the car from intensity values as green.
- If the car is red, the green filter does pass the red light (reflected from the car) but the red filter passes. The Monochrome camera catches the grayscale gray/or white (high intensity value) from the red filter half but it will catch black from the green filter half. Then, we can detect the color of the car from intensity values as red.
- If the car is blue, the green filter and the red filter do not pass the blue light (reflected from the car). The Monochrome camera catches black from the green and red filter half. This means the car is neither green or red but it can not be white either. Because if it was white, both of the filters would pass the reflected color from the car.Then, we can detect the color of the car from low intensity values as blue.
- If the car is white, the green filter and the red filter pass the white light ingredients green and red light (reflected from the car). The Monochrome camera catches the grayscale gray/or white (high intensity value) from the green and red filter half. This means the car's color includes both green. In our situation only the white car has a possibility like that.Then, we can detect the color of the car from high intensity values as white.

Problem 2 (G&W 2.18)
As in Section 2.5, let V be the set of intensity values used to define adjacency. Compute the lengths of the shortest paths between (p) and (q) in the following image. If a particular path does not exist between these two points, explain why.

3 1 2 1 (*q*)
2 2 0 2
1 2 1 1
(*p*) 1 0 1 2

a) The shortest 4-path with V ={0 1}.
b) The shortest 8-path with V ={0 1}.
c) The shortest m-path with V ={0 1}.

Answer:
a-) The shortest 4-path from p to q does not exist if we define the V ={ 0 1}. Let's examine the possible 4-paths from p. As you can see from **Figure 1,** I draw all of the possible 4-paths from p with V ={0 1}. However, it is not possible to reach q through these paths by using values V ={ 0 1} only. If we increase the elements in V, maybe we can find a path from p to q.

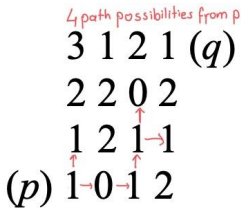


Figure 1

b-) The shortest 8-path from p to q exists if we define the V ={ 0 1}. As you can see from **Figure 2,** I draw the shortest 8-path from p to q with V ={0 1}. Its length is 4.

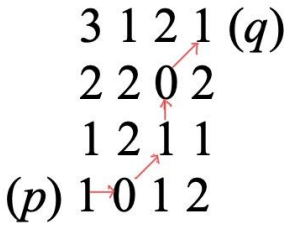


Figure 2

c-) The shortest m-path from p to q exists if we define the V ={ 0 1}. As you can see from **Figure 3,** I draw the shortest m-path from p to q with V ={0 1}.. Its length is 5.

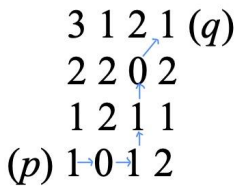


Figure 3

Problem 3 (G&W 2.36)
With reference to Table 2.3, provide single, composite transformation functions for performing the following operations:
a) Scaling and translation.
b) Scaling, translation, and rotation.
c) Vertical shear, scaling, translation, and rotation.
d) Does the order of multiplication of the individual matrices to produce a single transformation make a difference? Give an example based on a scaling/translation transformation to support your answer.

Answer:
a) If we want to scale first and translate later,we will perform the following matrix multiplication:

$$\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} cx & 0 & 0 \\ 0 & cy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

And find the single, composite transformation function as

$$\begin{pmatrix} cx & 0 & tx \\ 0 & cy & ty \\ 0 & 0 & 1 \end{pmatrix}$$

b) If we want to scale first, then translate and lastly rotate it, we will perform the following matrix multiplication:

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} cx & 0 & 0 \\ 0 & cy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

And find the single, composite transformation function as

$$\begin{pmatrix} cx\cos(\theta) & -cy\sin(\theta) & tx\cos(\theta) - ty\sin(\theta) \\ cx\sin(\theta) & cy\cos(\theta) & tx\sin(\theta) + ty\cos(\theta) \\ 0 & 0 & 1 \end{pmatrix}$$

c-) If we want to perform transformation in this order: Vertical shear, scaling, translation, and rotation, we need to perform the following matrix multiplication:

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} cx & 0 & 0 \\ 0 & cy & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & sv & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

And find the single, composite transformation function as

$$\begin{pmatrix} cx\cos(\theta) & cxsv\cos(\theta) - cy\sin(\theta) & tx\cos(\theta) - ty\sin(\theta) \\ cx\sin(\theta) & cxsv\sin(\theta) + cy\cos(\theta) & tx\sin(\theta) + ty\cos(\theta) \\ 0 & 0 & 1 \end{pmatrix}$$

d-) The order of multiplication of the individual matrices to provide a single transformation makes a difference. For example, let's say we perform scale first and translate it later. Then the matrix multiplication we will perform have this order:

$$\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} cx & 0 & 0 \\ 0 & cy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

And the resulting single, composite transformation function will be as in the part a.

$$\begin{pmatrix} cx & 0 & tx \\ 0 & cy & ty \\ 0 & 0 & 1 \end{pmatrix}$$

However, If we tried to perform translation first and scale it later, the order of multiplication will be like this:

$$\begin{pmatrix} cx & 0 & 0 \\ 0 & cy & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}$$

As a result of this multiplication we get single, composite transformation function as in the following:

$$\begin{pmatrix} cx & 0 & cx.tx \\ 0 & cy & cy.ty \\ 0 & 0 & 1 \end{pmatrix}$$

As you can see, the resulting composite transformation functions are different from each other. Therefore, the order of multiplication of the individual matrices to produce a single transformation makes a difference.

Problem 4 (G&W 3.1)
Give a single intensity transformation function for spreading the intensities of an image so the lowest intensity is 0 and the highest is L − 1.

Answer: Let's say that f(x,y) is the intensity function of the image. If we want to spread the intensities of an image so the lowest intensity is 0 and the highest is L-1, we can perform contrast stretching by using the intensity transformation function following, where $f_{new}(x, y)$ is the new intensity value.

$$f_{new}(x, y) = (f(x, y) - min\_value).\frac{(L-1)}{(max\_value - min\_value)}$$

```
clear; clc; close all;

% Read the image
img = imread('intensity_spread.png');

% Convert the image to grayscale if it's not already grayscale
if size(img, 3) == 3
    img = rgb2gray(img);
end

% Display the original image and its histogram
figure
subplot(1,3,1)
imshow(img)
title('Original Image')
subplot(1,3,2:3)
imhist(img)
title('Histogram')

% Apply intensity stretching
L = 255;
min_intensity = min(img(:));
max_intensity = max(img(:));

img_stretched =  (img - min_intensity) * ((L-1) / (max_intensity - min_intensity));


% Display the stretched image and its histogram
figure
subplot(1,3,1)
imshow(uint8(img_stretched)) % Convert back to uint8 for imshow
title('Stretched Image')
subplot(1,3,2:3)
imhist(uint8(img_stretched)) % Convert back to uint8 for imhist
title('Stretched Histogram')
```

Figure 4

Figure 4 shows the code segment that I perform the spreading the intensities of an image so the lowest intensity is 0 and the highest is L − 1. I give 255 to the L value. These are the results:

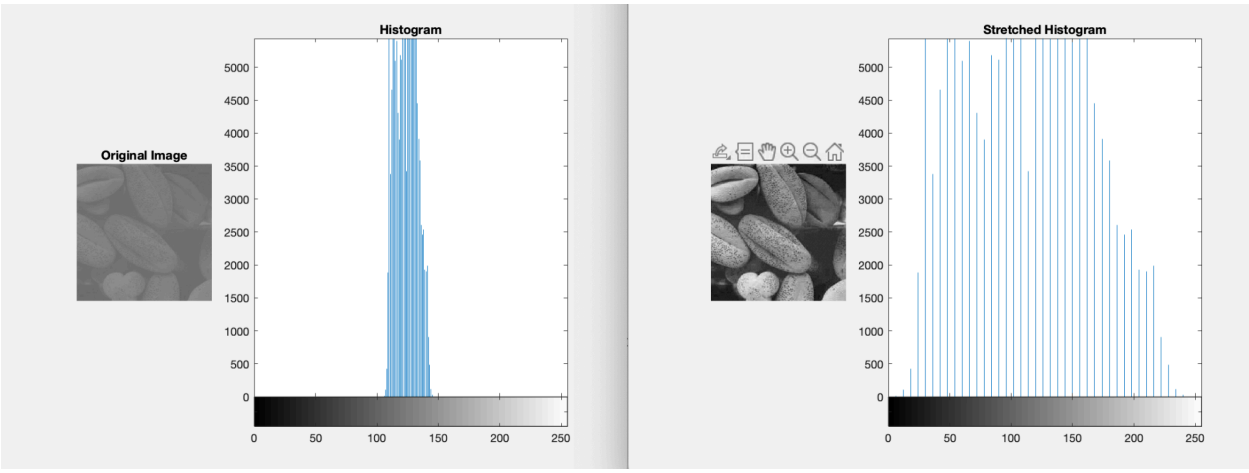


Figure 5

As you can see from **Figure 5,** the intensity values of the images are stretched to values between 0 to L-1 (254 for this example). This is called contrast stretching.

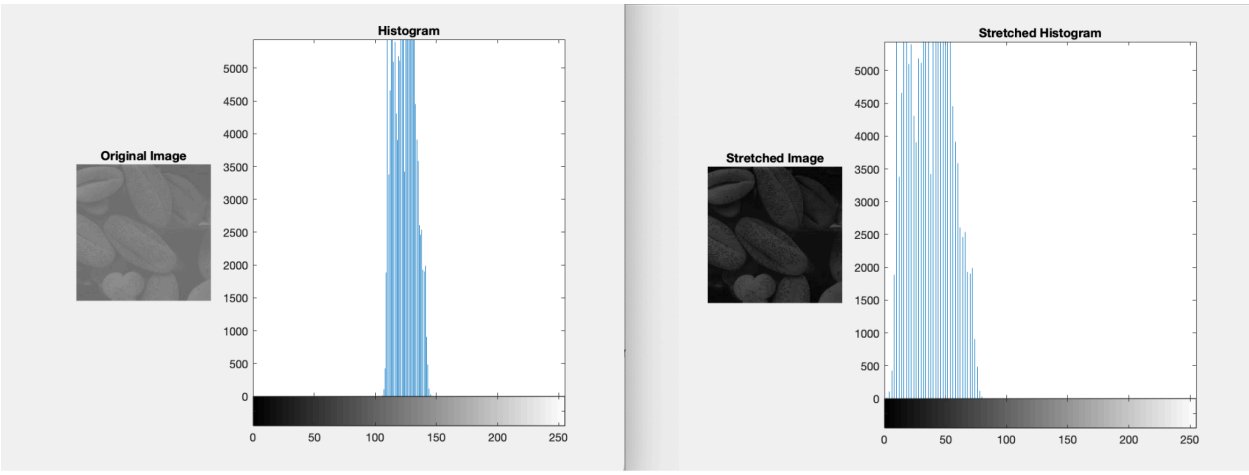


Figure 6

When I give L=90, As you can see from **Figure 6,** the intensity values of the images are stretched to values between 0 to 89.

Problem 5 (G&W 3.7)

Suppose that a digital image is subjected to histogram equalization. Show that a second pass of histogram equalization (on the histogram-equalized image) will produce exactly the same image as the first pass. You can either prove this mathematically by examining the equalization function, or construct a MATLAB script that demonstrates it experimentally by performing the process on some sample images, and visually highlighting any differences.

Answer: I chose to construct a MATLAB script that demonstrates it experimentally by performing the process on some sample images, and visually highlighting any differences. As seen from **Figure 7,** I performed the histogram equalization on the image and then performed a second histogram equalization on the equalized image. I used **histeq** function to perform that operation. In addition to that, I wrote a showImages function to both show images and the histogram of the images properly and save them to the folder called "Q5_Resulting_Images". One can find this folder in the folder I submitted.

```
clear; clc; close all;
% Read the image
img = imread('histogram_equalization.png');
%Convert the image to the grayscale
if size(img, 3) == 3
    img = rgb2gray(img);
end

% Adjust the contrast using histogram equalization.
% Use the default behavior of the histogram equalization function, histeq.
% The default target histogram is a flat histogram with 64 bins.
equalized_img = histeq(img);

% Second pass on the histogram-equalized image
second_pass_equalized_img = histeq(equalized_img);

images = {img,equalized_img,second_pass_equalized_img};
labels = {"Original Image", "Image After Histogram Equalization", "Second Pass on the Histogram Equalized Image"};

showImages(images, labels, "Q5_Resulting_Images", "Histogram Equalization", 1,3)

% Compare the second pass and first pass after histogram equalization
is_same_img = (equalized_img == second_pass_equalized_img);
if is_same_img
    disp(" second pass of histogram equalization (on the histogram-equalized image) ..." + ...
        " will produce exactly the same image as the first pass ")
else
    disp(" second pass of histogram equalization (on the histogram-equalized image) ..." + ...
        " will not produce exactly the same image as the first pass ")
end
```

```
%% This function will display the images and their histograms
function showImages(images, labels, file_name, final_image_name, subplot_x,subplot_y)
%Display the Images
    figure
    for i=1:length(images)
        currentImage = images{i};
        % Create a subplot
        subplot(subplot_x, subplot_y, i);
        % Display the image with its label
        imshow(currentImage, []);
        % Compute the noise standard deviation and mean before and after filtering.
        title(labels{i} + " mean:" + mean2(currentImage) + " std: " + std2(currentImage));
        imwrite(mat2gray(currentImage),fullfile(file_name, labels{i}+".jpg"));
    end
    % Adjust layout
    sgtitle(final_image_name + "-Images");
    set(gcf, 'Position', [100, 100, 800, 600]);
    saveas(gcf, fullfile(file_name, final_image_name+ ".jpg"));
    figure
    for i=1:length(images)
        currentImage = images{i};
        % Create a subplot
        subplot(subplot_x, subplot_y, i);
        % Create histogram for both images
        histogram(currentImage, 'Normalization','count');
        title(labels{i});
        imwrite(mat2gray(currentImage),fullfile(file_name, labels{i}+".jpg"));
    end
    % Adjust layout
    sgtitle(final_image_name+"-Histograms");
    set(gcf, 'Position', [100, 100, 800, 600]);
    saveas(gcf, fullfile(file_name, final_image_name+ ".jpg"));
end
```

Figure 7

Resulting Images:

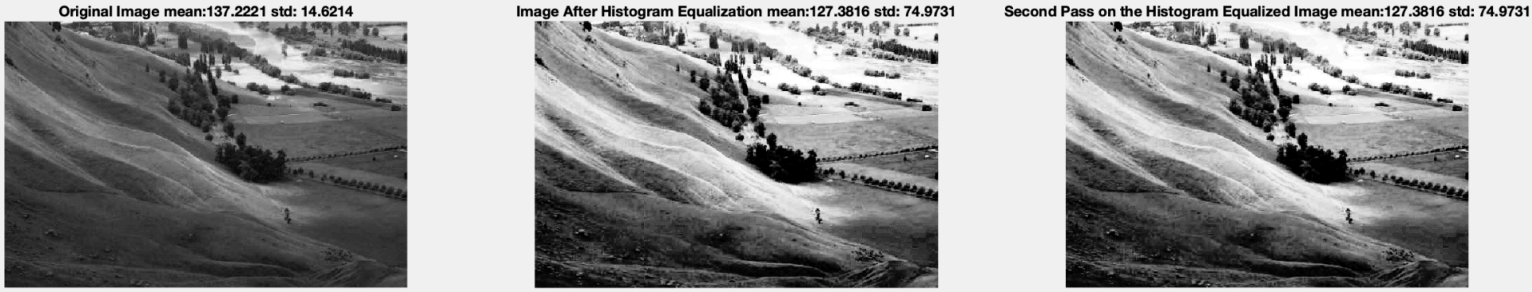


Figure 8

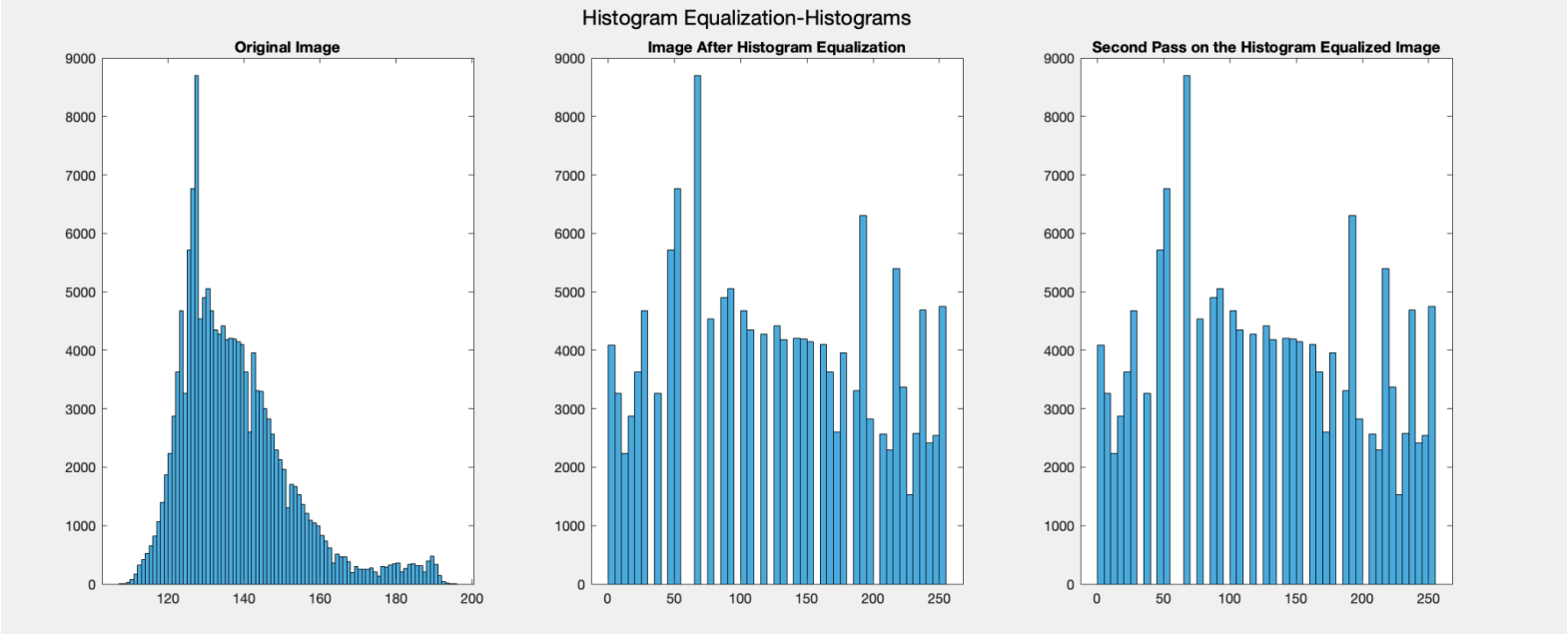


Figure 9

As one can see from **Figure 8** and **Figure 9,** the second pass of the histogram equalization is the same with the first pass of the histogram equalization. They seem exactly the same in **Figure 8** and their histograms seem to be the same in **Figure 9.** To be sure about this, I also compared these two image arrays in matlab and check whether they are the same or not in this code snippet:

```
% Compare the second pass and first pass after histogram equalization
is_same_img = (equalized_img == second_pass_equalized_img);
if is_same_img
    disp(" second pass of histogram equalization (on the histogram-equalized image) ..." + ...
        " will produce exactly the same image as the first pass ")
else
    disp(" second pass of histogram equalization (on the histogram-equalized image) ..." + ...
        " will not produce exactly the same image as the first pass ")
end
```

Figure 10

The result is:

```
 second pass of histogram equalization (on the histogram-equalized image) ... will produce exactly the same image as the first pass
>>
```

Problem 6 (G&W 3.21)

Given the following kernel and image:

$$w = \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix} \qquad f = \begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

a) Give the convolution of the two.

b) Does your result have a bias?

Answer:

```
clear; clc; close all;
% Create kernel and the image
f = ones(5,5);
w = [1 2 1
    2 4 2
    1 2 1];
%Convolve these two
conv_result = conv2(f,w)
%Check bias
sum_f = sum(f(:))
sum_conv_result = sum(w(:))

if sum_f == sum_conv_result
    disp(['the sum of the pixels in the original and filtered images were the same,' ...
        ' thus preventing a bias from being introduced by filtering'])
else
    disp(['the sum of the pixels in the original and filtered images were not the same' ...
        ', thus have a bias from being introduced by filtering'])
end
```

Figure 11

I wrote a MATLAB script to find the convolution of the f and w. In this code, I also check whether the result has a bias or not by comparing the sum of the original image and the filtered image. Here are the results:

```
conv_result =

     1     3     4     4     4     3     1
     3     9    12    12    12     9     3
     4    12    16    16    16    12     4
     4    12    16    16    16    12     4
     4    12    16    16    16    12     4
     3     9    12    12    12     9     3
     1     3     4     4     4     3     1


sum_f =

    25


sum_conv_result =

    16

the sum of the pixels in the original and filtered images were not the same, thus have a bias from being introduced by filtering
```

Figure 12

So according to Figure **12,** the sum of the pixels in the original and filtered images were not the same, thus the result has a bias from being introduced by filtering.

Problem 7 (G&W 3.27)

An image is filtered four times using a Gaussian kernel of size 3 × 3 with a standard deviation of 1.0. Because of the associative property of convolution, we know that equivalent results can be obtained using a single Gaussian kernel formed by convolving the individual kernels.

a) What is the size of the single Gaussian kernel?

b) What is its standard deviation?

Answer:

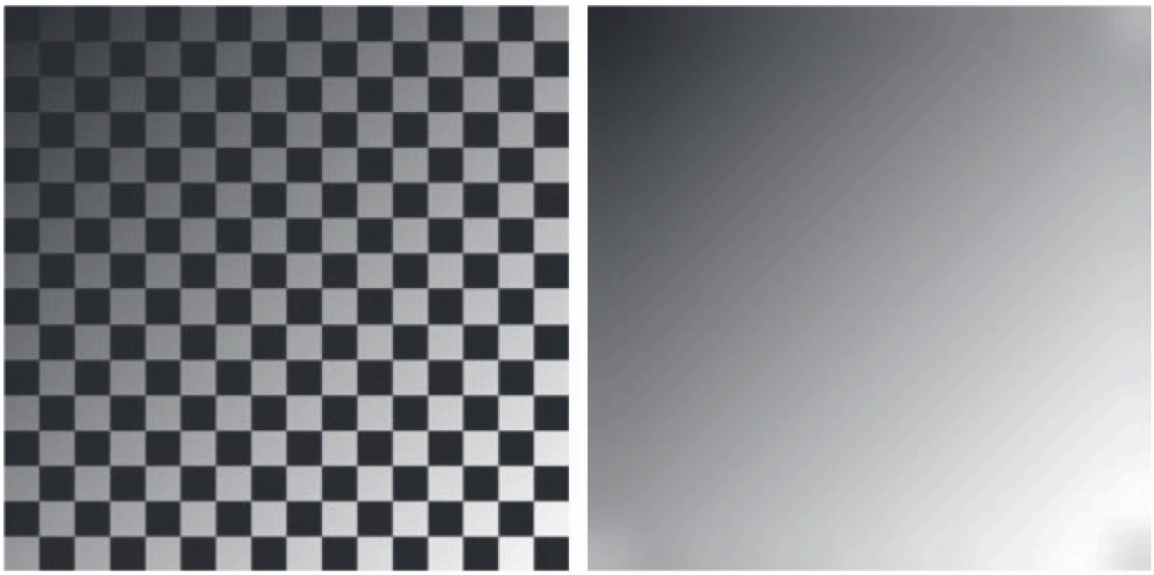
a) We know that every time we convolve two matrices with size MxN and mxn, the resulting "full" convolution will have the size m+M-1 x n+N-1. In our situation we are convolving the 3x3 Gaussian filters four times firstly to get a single Gaussian kernel.
   - First convolution of kernel 1 with size 3x3 and kernel 2 with size 3x3 kernels will result 5x5 kernel
   - The second convolution of kernel 3 with size 3x3 and kernel 4 with size 3x3 kernels will result 5x5 kernel
   - If we convolve the results of 5x5 and 5x5 kernels, we will have one single 9x9 gaussian kernel.

b-) Convolving two times with a Gaussian kernel with standard deviation σ is same as convolving once with a kernel with standard deviation $\sigma\sqrt{2}$ [1]. In our situation,

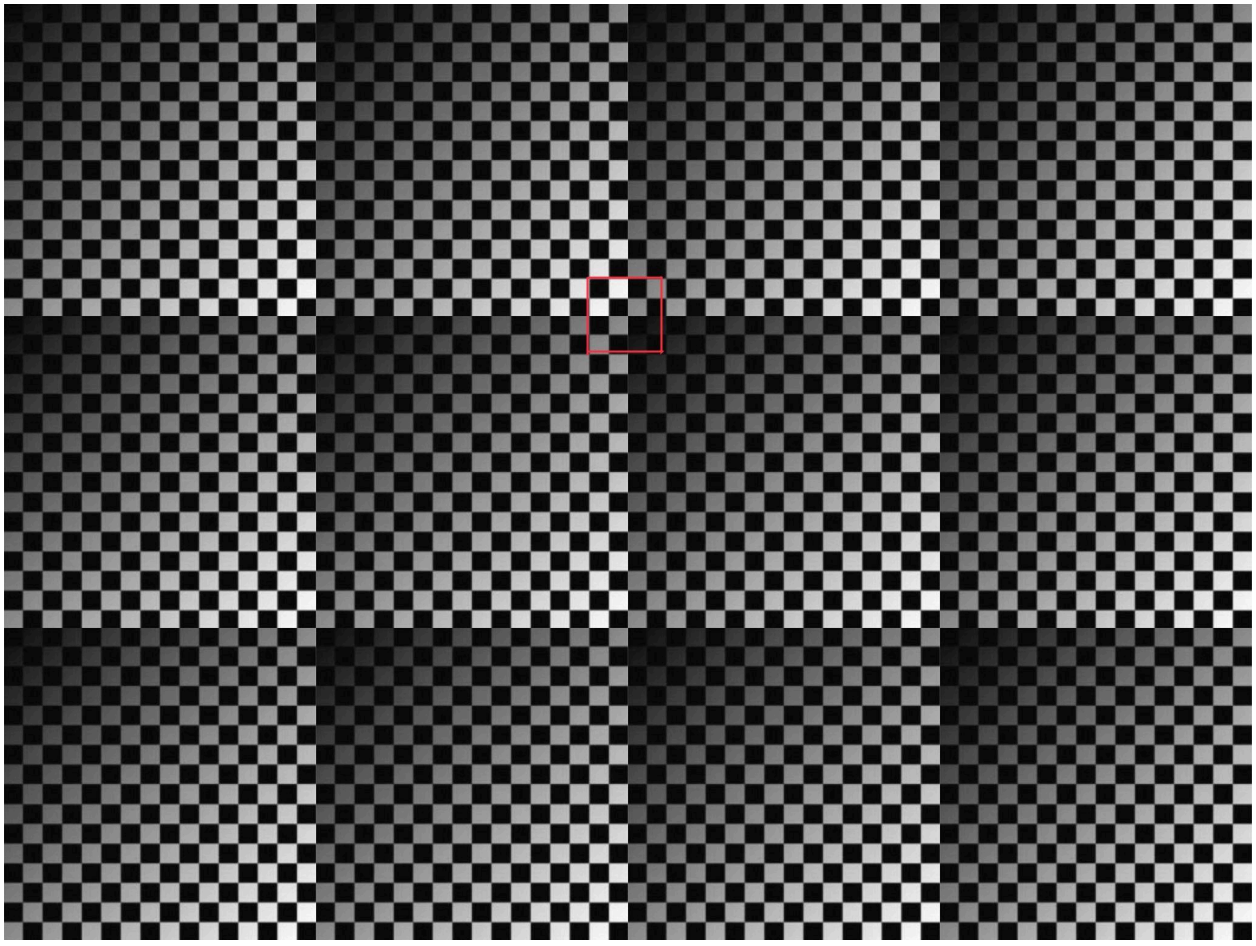
- Kernel 1 and kernel 2 have σ =1, and if we convolve them we get a kernel with standard deviation $\sqrt{2}$
- Kernel 3 and kernel 4 have σ =1, and if we convolve them we get a kernel with standard deviation $\sqrt{2}$

- If we will convolve the resulting kernels that have σ =$\sqrt{2}$, we will get a 9x9 Gaussian kernel with σ =2.

Problem 8 (G&W 3.30)
The image below (G&W's Fig. 3.42) features a shaded image, and a shading pattern extracted from it by low-pass filtering. The corners of the extracted shading pattern appear darker or lighter than their surrounding areas. Explain the reason for this.



Answer:
The corners of the extracted shading pattern appear darker or lighter than their surrounding area due to 2 reasons. Firstly, the image is shaded by a shading pattern oriented in the -45° direction.When the low pass filtering is performed, instead of zero-padding if one uses tiled padding as in the following image, the corners of the extracted shading pattern appear darker or lighter than their surrounding areas because the corners have different brightness levels as seen from the image. If we focus on the red box part, we can understand why the corners of the extracted shading pattern appear darker or lighter than their surrounding areas better.



Problem 9 (G&W 3.38)
In a given application, a smoothing kernel is applied to input images to reduce noise, then a Laplacian kernel is applied to enhance fine details. Would the result be the same if the order of these operations is reversed?

Answer: The result will not be the same if the order of these operations is reversed. I wrote the following MATLAB code to find whether the result will be the same if the order of these operations is reversed or not.

```
clear; clc; close all;
%Read gaussian noise image
edges_gnoise = imread("noisy.png");
%Convert the image to the grayscale
if size(edges_gnoise, 3) == 3
    edges_gnoise = rgb2gray(edges_gnoise);
end
% Create large gaussian kernel
h_noise = fspecial('gaussian', [21 21],3.5);
h_laplacian = fspecial('laplacian', 0);

%Filter image
filtered_img_noise_1 = conv2(edges_gnoise, h_noise);
filtered_img_laplacian_1 = conv2(filtered_img_noise_1, h_laplacian );

%Filter image
filtered_img_laplacian_2 = conv2(edges_gnoise, h_laplacian );
filtered_img_noise_2 = conv2(filtered_img_laplacian_2, h_noise );


images = {edges_gnoise,filtered_img_laplacian_1,filtered_img_noise_2,abs(filtered_img_laplacian_1-filtered_img_noise_2)};
labels = {"Original Image", "First Noise Then Laplacian", "First Laplacian Then Noise", "Difference"};

showImages(images, labels, "Q9_Resulting_Images", "Order of Filterings", 2,2)

if filtered_img_laplacian_1 == filtered_img_noise_2
    disp(" the result be the same if the order of these operations is reversed")
else
    disp(" the result be not the same if the order of these operations is reversed")
end
```

```
%% This function will display the images and their histograms
function showImages(images, labels, file_name, final_image_name, subplot_x,subplot_y)
%Display the Images
    figure
    for i=1:length(images)
        currentImage = images{i};
        % Create a subplot
        subplot(subplot_x, subplot_y, i);
        % Display the image with its label
        imshow(currentImage, []);
        % Compute the noise standard deviation and mean before and after filtering.
        title(labels{i} + " mean:" + mean2(currentImage) + " std: " + std2(currentImage));
        imwrite(mat2gray(currentImage),fullfile(file_name, labels{i}+".jpg"));
    end
    % Adjust layout
    sgtitle(final_image_name + "-Images");
    set(gcf, 'Position', [100, 100, 800, 600]);
    saveas(gcf, fullfile(file_name, final_image_name+ ".jpg"));
end
```

Figure 13

The results:

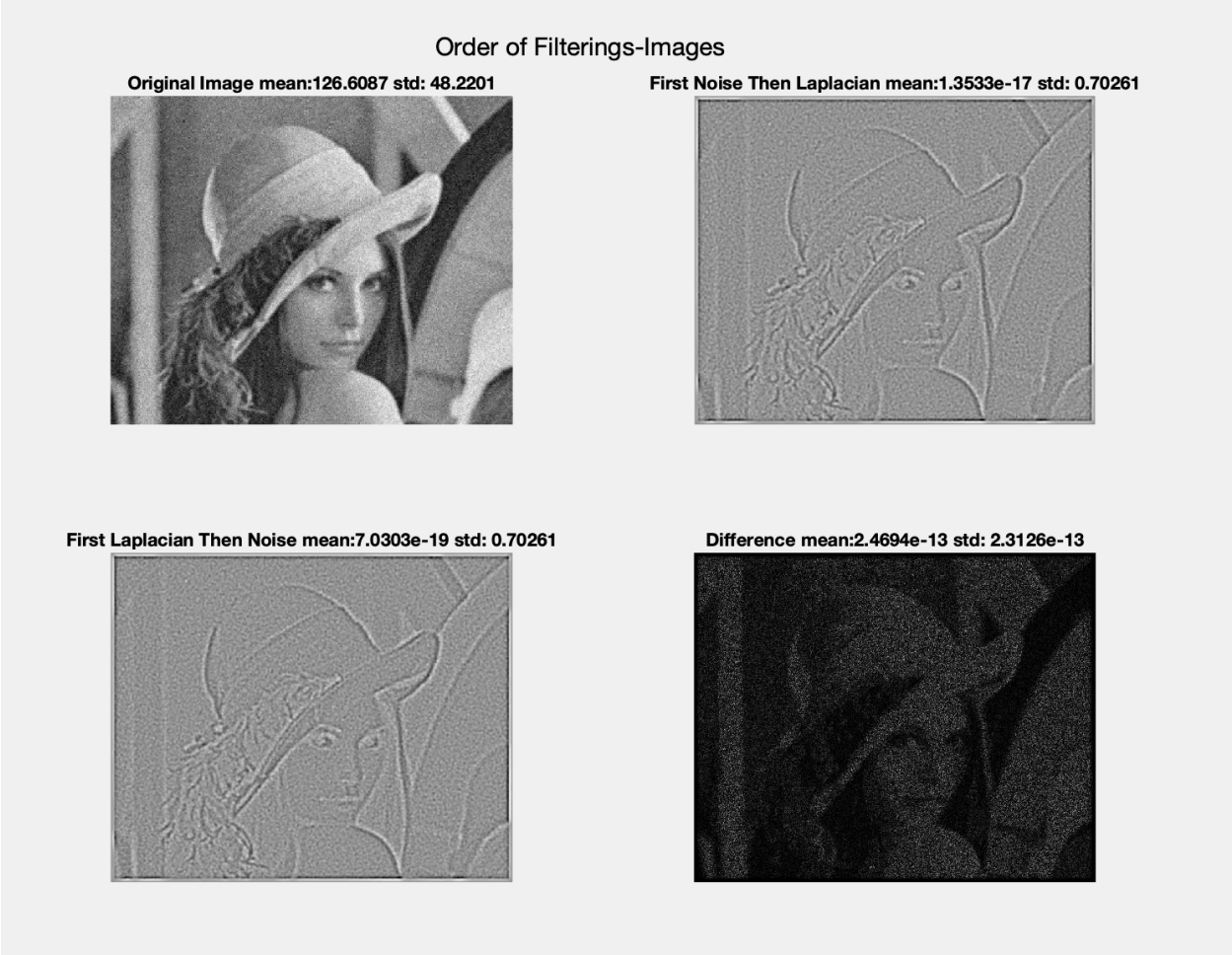


Figure 14

In **Figure 14,** It is hard to see whether the order matters for smoothing and laplacian by only looking at the “First Noise Then Laplacian” and “First Laplacian Then Noise” images. Therefore, I put their differences to see whether they have the same result. As we can see, we do not have the same result. The following code segment also does the same job and results as **Figure 16.**

```
if filtered_img_laplacian_1 == filtered_img_noise_2
    disp(" the result be the same if the order of these operations is reversed")
else
    disp(" the result be not the same if the order of these operations is reversed")
end
```

Figure 15

```
 the result be not the same if the order of these operations is reversed
>>
```

Figure 16

Why is this happening? When we applied the Laplacian first, since it enhances the edges by amplifying the details, it will also increase the noise in the image. After that, if we try to filter the

noise by gaussian (I choose this one), It will also blur the edges and since some of the noises are enhanced it will be harder to filter them. However, in the reverse order, a gaussian filter will remove the noise first, then the edges will be enhanced by the Laplacian.

Problem 10 (G&W 3.41)
Give a 3 × 3 kernel for performing unsharp masking in a single pass through an image. Assume that the average image is obtained using a box filter of size 3 × 3.

Answer:
I would solve this by using the Fourier transforms. Firstly when we perform unsharp masking, we know that the value of k =1 in this equation:

$$\begin{aligned} g(x,y) &= f(x,y) + k\,.\,(f(x,y) - \overline{f}(x,y)) \\ &= f(x,y) + (f(x,y) - \overline{f}(x,y)) \\ &= 2f(x,y) - \overline{f}(x,y) \qquad where\ \overline{f}(x,y) = conv(f(x,y), h) \end{aligned}$$

If I took the Fourier transform of g(x,y) from the linearity property of Fourier Transform:

$$\begin{aligned} G(x,y) &= F(x,y) + (F(x,y) - \overline{F}(x,y)) \\ &= 2F(x,y) - \overline{F}(x,y) \qquad where\ \overline{F}(x,y) = F(x,y).H(X,Y) \\ &= 2F(x,y) - F(x,y).H(X,Y) \\ &= F(x,y)(2.I - H(X,Y)) \end{aligned}$$

So, the wanted kernel in frequency domain is $2.I - H(X,Y)$. If we take the inverse fourier transform of this, we will find our kernel.
To find it numerically, I wrote the MATLAB script in **Figure 17** which computes the above functions. I found the values in **Figure 18.** The matrix in the red box gives us a 3 × 3 kernel for performing unsharp masking in a single pass through an image.

```
clear; clc; close all;
%Read gaussian noise image
img = imread("unsharp_masking.png");
if size(img, 3) == 3
    img = rgb2gray(img);
end
% Normal unsharp masking
% Blur image
h = createAveragingKernel(3)
blurred_img = conv2(img,transpose(h), 'same');

%Add the mask to the original image
unsharpening = img+(img-uint8(blurred_img));

figure
imshow(unsharpening);

% find the fft of the kernel
fft_h = fft2(h)
I = [1 0 0; 0 1 0; 0 0 1]
h_one_pass_fft = 2*I-fft_h
f_one_pass = ifft2(h_one_pass_fft)

%perform Unsharpening in single pass
unsharpening_single = imfilter(img,f_one_pass);

figure
imshow(unsharpening_single,[]);

%% This function will take kernel size and create averaging kernel
function averaging_kernel = createAveragingKernel(kernel_size)
    averaging_kernel = ones(kernel_size,kernel_size) * (1/(kernel_size^2));
end
```

Figure 17

```
h =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111


fft_h =

     1     0     0
     0     0     0
     0     0     0


I =

     1     0     0
     0     1     0
     0     0     1


h_one_pass_fft =

     1     0     0
     0     2     0
     0     0     2


f_one_pass =

    0.5556   -0.1111   -0.1111
   -0.1111   -0.1111    0.5556
   -0.1111    0.5556   -0.1111
```

Figure 18

References

[1] S. Maji, Gaussian filters separability of the ..., http://www-edlab.cs.umass.edu/~smaji/cmpsci370/slides/hh/lec02_hh_advanced_edges.pdf (accessed Feb. 26, 2024).