EEE 391
Basics of Signals and Systems

MATLAB Assignment 2
28.12.2018

Sabit Gökberk Karaca
21401862

Introduction

Grayscale Image

The original image is read from the given ‘fruits.png’ file and converted to a grayscale image by using the code below. The resulting image is shown in *Figure 1*.

```
% Read the image from the file
image_rgb = imread('fruits.png');
image = im2double(rgb2gray(image_rgb));
figure('Name','Gray Scale Image');
imshow(image, []);
```

Figure 1: Grayscale Image

Fourier Transform

After reading the image and converting it to grayscale, I have performed the Fourier Transform operation on that image by using the given function and created FT_image variable by using the code below. Then, I calculated the magnitude and phase matrices of that image. The results are shown in *Figure 2* and *Figure 3.*

```
% Apply Fourier Transform to the image
FT_image = FourierTransform(image);
% Plot magnitude and phase of the image
FT_magnitude = log(1+abs(FT_image));
FT_angle = angle(FT_image);
figure('Name', 'Magnitude');
imshow(FT_magnitude, []);
figure('Name', 'Angle');
imshow(FT_angle, []);
```

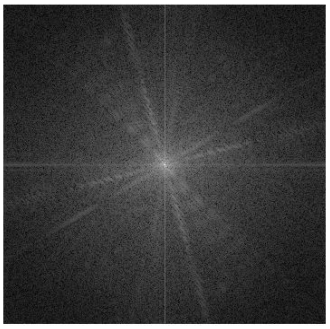


Figure 2: Magnitude

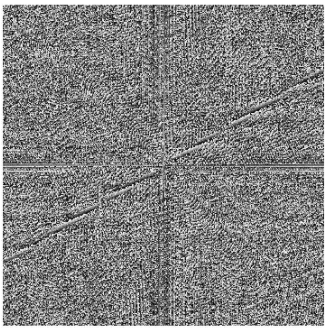


Figure 3: Phase

2D Frequencies

In the introduction part, some frequency values are given. These values are;

$$\begin{aligned} wx &= -\pi && for \quad n = 1 \\ wx &= \pi - 2\pi/N && for \quad n = N \\ wy &= -\pi && for \quad m = 1 \\ wy &= \pi - 2\pi/M && for \quad m = M \end{aligned}$$

By using the given values and assuming that frequency values are uniformly distributed, the equation that satisfies all conditions can be found as;

$$wy = -\pi + 2\pi(m-1)/M \quad \text{and} \quad wx = -\pi + 2\pi(n-1)/N$$

Thus, the 2D frequency for an arbitrary [m, n] pair is;

$$[-\pi + 2\pi(m-1)/M,\ -\pi + 2\pi(n-1)/N]$$

High Pass & Low Pass Filters

Low Pass Filter

Considering the given equation for the low pass filter $LP = \begin{cases} 1 & \text{if } \frac{-\pi}{4} \le \hat{\omega}_x \le \frac{\pi}{4}, \frac{-\pi}{4} \le \hat{\omega}_y \le \frac{\pi}{4} \\ 0 & \text{otherwhise} \end{cases}$

The low pass filter can be implemented and applied to the grayscale image in MATLAB by the following code

```
% Creating and appliyng low pass filter
LP = zeros(size(image));
bandwidth = pi/4;
for i = 1:size(image,1)
   for j = 1:size(image,2)
       wx = -pi + 2*pi*(j-1)/512;
       wy = -pi + 2*pi*(i-1)/512;
       wx_condition = -bandwidth <= wx && wx <= bandwidth;
       wy_condition = -bandwidth <= wy && wy <= bandwidth;
       LP(i,j) = wx_condition && wy_condition;
   end
end
Filtered_Image = InverseFourierTransform(LP.*FT_image);
figure('Name','Low Pass Filter');
imshow(real(Filtered_Image),[]);
```

Since high pass filter is defined as logically opposite of low pass filter;

$$HP = \begin{cases} 0 & \text{if } \frac{-\pi}{4} \leq \hat{\omega}_x \leq \frac{\pi}{4}, \frac{-\pi}{4} \leq \hat{\omega}_y \leq \frac{\pi}{4} \\ 1 & \text{otherwhise} \end{cases}$$

It can be generated and applied to the grayscale image by subtracting the low pass filter matrix from ones matrix of same size;

```
% Creating and appliyng high pass filter
HP = ones(512,512) - LP;
Filtered_Image = InverseFourierTransform(HP.*FT_image);
figure('Name','High Pass Filter');
imshow(real(Filtered_Image),[])
```

Outputs of low pass filter and high pass filter are shown in *Figure 4* and *Figure 5*

Figure 4: LP Output

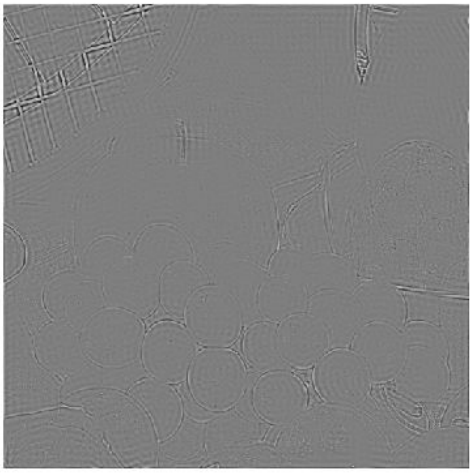


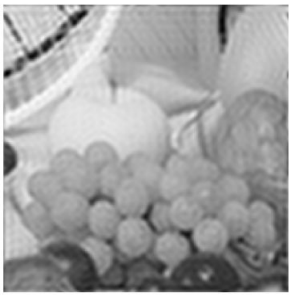
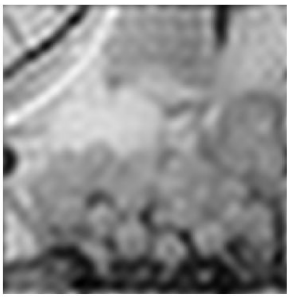
Figure 5: HP Output

These outputs actually show that a low pass filter blurs the image whereas a high pass filter sharpens the image, they are opposite of each other.

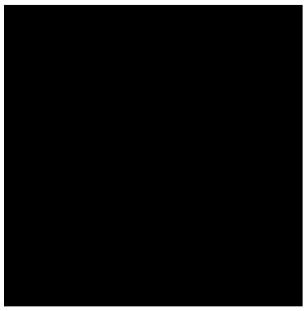
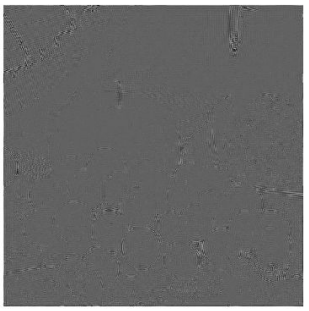
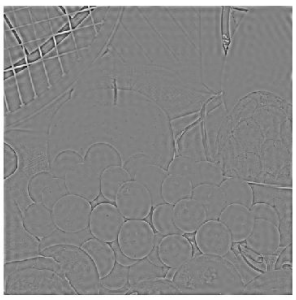
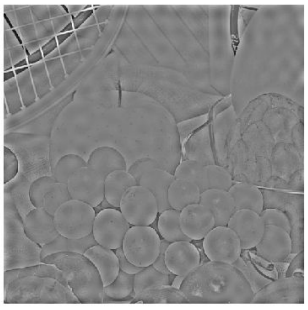
However, it is not possible to observe these behavior on the images because the bandwidth $\pi/4$ is not a good value. To observe the effect of bandwidth change, I chose bandwidth to be $[\pi, \pi/2, \pi/8, \pi/16]$. Outputs of these filters are shown below for both low pass filter and high pass filter.

Low Pass Filter Outputs



High Pass Filter Outputs

When the outputs of filters with different bandwidth are observed, it can be seen that the images filtered with the low pass filter becomes more blurred and images filtered with the high pass filter becomes sharper while bandwidth goes from π to $\pi/16$.

Custom Filters

conv2 Function

In that part, I have implemented the conv2 function and its helper functions to be used in the following parts of the homework. The function can be found below;

```
function result = my_conv2(A, B)
    % Assume that A is an axb array
    row_a = size(A,1);
    column_a = size(A,2);

    % Assume that B is an cxd array
    row_b = size(B,1);
    column_b = size(B,2);

    % Rotate B 180 degrees
    B = rot180(B);

    % Add zero paddings to the original matrix
    A = add_zero_padding(A, row_b-1, column_b-1);

    % Create and fill the result matrix
    result = zeros(row_a , column_a);
    for i = 1:row_a
        for j = 1:column_a
            filtered_area_of_A = A(i:i+row_b-1, j:j+column_b-1);
            result(i, j) = result(i, j) + sum(filtered_area_of_A.*B, 'all');
        end
    end
end
```

```
function result = add_zero_padding(A, row_padding, column_padding)
    % Calculate new dimensions and create the matrix
    new_row_size = size(A,1) + row_padding;
    new_column_size = size(A,2) + column_padding;
    result = zeros(new_row_size, new_column_size);
    % Calculate left and top paddings
    top_padding = floor(row_padding/2);
    left_padding = floor(column_padding/2);
    % Copy the matrix into its field in new one
    result(top_padding + 1:top_padding + size(A,1), left_padding + 1:left_padding +
size(A,2)) = A;
end

% Improves readability
function result = rot180(A)
    result = rot90(A,2);
end
```

hx Filter

The impulse response hx is defined as $h_x[m,n] = -\delta[m,n-1] + \delta[m,n+1]$

This corresponds to the filter

0	0	0
1	0	-1
0	0	0

By using the 2D convolution function I wrote above, I have calculated the A*hx and got the output that is shown in *Figure 6.*

```
A = image;

% Calculate A*hx
hx = [0, 0, 0; 1, 0, -1; 0, 0, 0];
res = my_conv2(A, hx);
figure('Name','A*hx');
imshow(res, []);
```

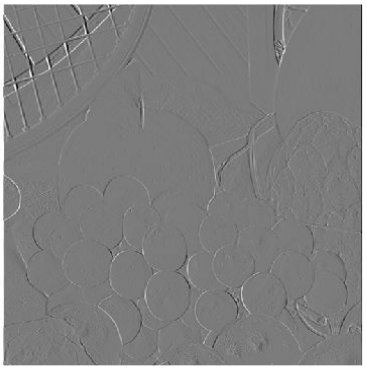


Figure 6: A*hx

Resulting image shows that hx filter makes the edges on the vertical axis more obvious.

hy Filter

The impulse response hy is defined as $h_y[m,n] = -\delta[m-1,n] + \delta[m+1,n]$

This corresponds to the filter

0	1	0
0	0	0
0	-1	0

By using the 2D convolution function I wrote above, I have calculated the A*hy and got the output that is shown in *Figure 7.*

```
A = image;

% Calculate A*hy
hxy = [0, 1, 0; 0, 0, 0; 0, -1, 0];
res = my_conv2(A, hy);
figure('Name','A*hy');
imshow(res, []);
```

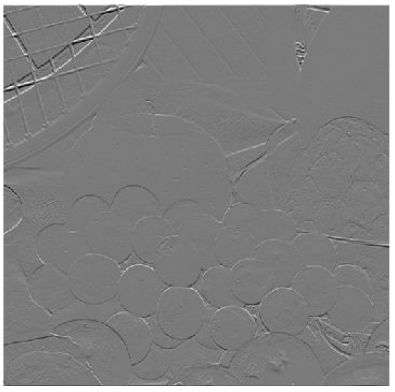


Figure 7: A*hy

Resulting image shows that hy filter makes the edges on the horizontal axis more obvious.

hxy Filter

When analytically calculated, the hxy filter can be found as

1	0	-1
0	0	0
-1	0	1

Thus, the impulse response equation for hxy filter is

$$hxy = \delta[m+1,\ n+1] - \delta[m+1,\ n-1] - \delta[m-1,\ n+1] + \delta[m-1,\ n-1]$$

When this filter applied to the image, the output shown in *Figure 8* is obtained. This figure shows that the hxy filter makes the diagonal edges more obvious. This was the expected result because the hxy filter is the convolution of hx and hy.

```
% Calculate hxy analytically then calculate A*hxy
hxy = [1, 0, -1; 0, 0, 0; -1, 0, 1];
res = my_conv2(A, hxy);
figure('Name','A*hxy');
imshow(res, []);
```

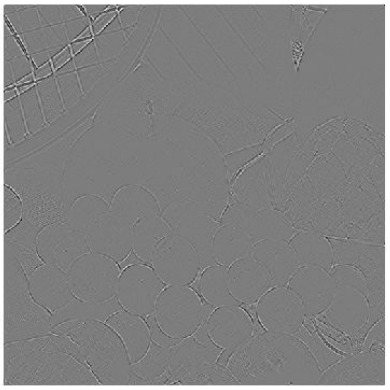


*Figure 8: A*hxy*

hx*hy Filter

When hx and hy filters are applied to the input image consecutively, the output shown in the *Figure 9* is obtained. The *Figure 8* and *Figure 9* are the same. This shows that the result does not depend on whether if we convolved the filters first and then applied the resulting filter to the image or we apply the filters to the image consecutively. The figure and the code section is shown below.

```
% Calculate A*hx*hy
hxhy = my_conv2(hx, hy);
res = my_conv2(A, hxhy);
figure('Name','A*hx*hy');
imshow(res, []);
```

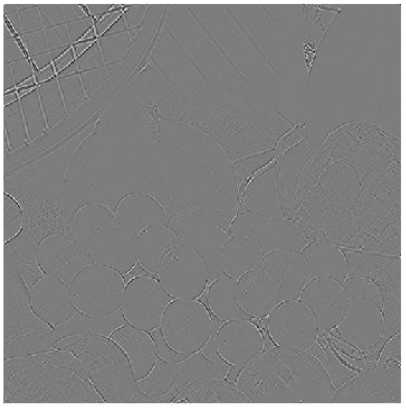


*Figure 9: A*hx*hy*

Sobel Operator

In this part, I have created the matrix G from the given impulse responses and given operations. When displayed the G, I got the output shown in Figure 10. This figure is similar to the ones that I got in the previous parts. It also makes the edges more obvious. The difference is that it makes all horizontal, vertical and diagonal edges more obvious and the contrast between edges and other fields are higher. So, it is easier to see the edges.

```
hx = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
gx = my_conv2(A, hx);
hy = [-1, -2, -1; 0, 0, 0; 1, 2, 1];
gy = my_conv2(A, hy);
g = sqrt(gx.^2 + gy.^2);
figure('Name','G');
imshow(g, []);
```

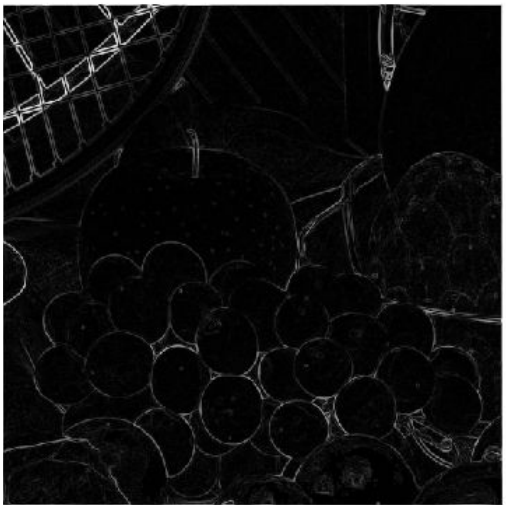


Figure 10: G matrix