Frequency Domain Filtering Report

By Zhang Hewen, 12010342

1. Objective and Principles

The objective of this laboratory is to process images in frequency domain. The advantages of using frequency domain include multiplication replacing convolution, making it more convenient to design filters with different usages and no need to design complicated spatial filters when they have a large size. Consequently, processing images in frequency domain is very practical. For example, it can be used to preprocess medical images, enhance fingerprint or assist geographical research like finding a certain place.

The basic principle of using frequency domain of an image is Fourier Transform theory. The formulas are:

$$DFT: F(\mu,\nu) = \sum_{x=0}^{M-1}\sum_{y=1}^{N-1} f(x,y)e^{-j2\pi(\mu x/M+\nu y/N)}$$

$$IDFT: f(x,y) = \frac{1}{MN}\sum_{\mu=0}^{M-1}\sum_{\nu=0}^{N-1} F(\mu,\nu)e^{j2\pi(\frac{\mu x}{M}+\frac{\nu y}{N})}$$

The 2D convolution theorem is also used:

$$f(x,y)\star h(x,y) \Leftrightarrow F(\mu,\nu)H(\mu,\nu)$$

Through the laboratory, I find that generating many dfferent types of filters by using frequency domain is convenient and it may be faster than using filters in spatial domain. I learn that filters also have their limits, for example, unlike what we learn in communication principles, ideal filters may not be the best choice. Additionally, I find that writing an algorithm that can perform generally is difficult and different image need different algorithms to enhance them in different aspects.

2. Formulas for each Filters

The tasks in this laboratory are implementing Sobel filters in spatial domain and frequency domain, ideal lowpass filter, Gaussian lowpass filter and Butterworth notch filter.

1. Sobel Filter

The Sobel mask in spatial domain used in this laboratory is:

-1	0	1
-2	0	2
-1	0	1

It is real and odd symmetric, so the mask in frequency domain is imaginary and odd. It is a zero-phase-shfit filter because the information of the result image after passing it is completely restored.

2. Ideal Lowpass Filter

The ideal lowpass filter in frequency domain is:

$$H(\mu,\nu)=\begin{cases}1 & if\ D(\mu,\nu)\le D_0\\ 0 & if\ D(\mu,\nu)>D_0\end{cases}$$

$$D(\mu,\nu)=[(\mu-\frac{M}{2})^2+(\nu-\frac{N}{2})^2]^{\frac{1}{2}}$$

3. Gaussian Lowpass Filter

The Gaussian lowpass filter in frequency domain is:

$$H(\mu,\nu)=e^{\frac{-D^2(\mu,\nu)}{2\sigma^2}}$$

$$D(\mu,\nu)=[(\mu-\frac{N}{2})^2+(\nu-\frac{N}{2})^2]^{\frac{1}{2}}$$

4. Butterworth Notch Filter

The Butterworth notch reject filter in frequency domain is:

$$H_{NR}(\mu,\nu)=\prod_{k=1}^{Q}[\frac{1}{1+[\frac{D_{0k}}{D_k(\mu,\nu)}]^{2n}}][\frac{1}{1+[\frac{D_{0k}}{D_{-k}(\mu,\nu)}]^{2n}}]$$

$$D_k(\mu,\nu)=[(\mu-\frac{M}{2}-\mu_k)^2+(\nu-\frac{N}{2}-\nu_k)^2]^{\frac{1}{2}}\quad D_{-k}(\mu,\nu)=[(\mu-\frac{M}{2}+\mu_k)^2+(\nu-\frac{N}{2}+\nu_k)^2]^{\frac{1}{2}}$$

3. Pseudo Code

1. Sobel Filter

In Spatial Domain

```
filter = [[-1,0,1],[-2,0,2],[-1,0,1]]
img0 = np.pad(input,(padding_coefficients))  /*pad the input image*/
output = np.zeros(input.shape)
for i in (0,output.shape[0]):
    for j in (0,output.shape[1]):
        subimg0 = img0[(i):(i+3),(j):(j+3)]
        /*select the patch with the same size as the filter*/
        output[i][j] = np.abs(np.sum(np.multiply(filter,subimg0)))
 /*convolution*/
output = input + output  /*g(x,y)=f(x,y)+M(x,y), M(x,y)=grad(f)*/
```

In Frequency Domain

```
filter = [[-1,0,1],[-2,0,2],[-1,0,1]]
img0 = np.pad(input,(padding_coeffcients))
freq_filter = np.pad(filter,(padding_coeffcients))
for i,j in (0,img0.shape[0]), (0,img0.shape[1]):
    img0[i][j] = img0[i][j]*np.power(-1,i+j)  /*get f(x,y)*[(-1)^(x+y)]*/
...   /*using the same way to get h(x,y)*[(-1)^(x+y)]*/
freq_filter = np.fft.fft2(freq_filter)  /*get the Fourier Transform of the
filter*/
freq_img = np.fft.fft2(input)
```

```
...  /*using the way above to get H(u,v)*[(-1)^(u+v)]*/
freq_filter = np.imag(freq_filter)*1j  /*get the imaginary part of H(u,v)*/
output = np.real(np.fft.ifft2(np.multiply(freq_img, freq_filter)))
/*the multiplication of the FT of two images*/
...  /*using the way above to get M(x,y)*[(-1)^(x+y)]*/
output = output[0:(input.shape[0]),0:(input.shape[1])]
/*crop the output to let it have the same size as the input*/
output = input + np.abs(output)  /*g(x,y)=f(x,y)+M(x,y), M(x,y)=grad(f)*/
```

2. Ideal Lowpass Filter

```
img0 = np.pad(input,(padding_coeffcients))
freq_filter = np.zeros(img0.shape)
for i in (0,M):  /*M = freq_filter.shape[0]*/
    for j in (0,N):  /*N = freq_filter.shape[1]*/
        if cal_D(i,j,M,N]) <= D0:
            freq_filter[i][j] = 1
/*cal_D is used to calculate the distance between (u,v) and the midpoint of the filter; D0 is the cutoff frequency; 1 is the gain of the filter*/
for i,j in (0,img0.shape[0]), (0,img0.shape[1]):
    img0[i][j] = img0[i][j]*np.power(-1,i+j)  /*f(x,y)*[(-1)^(x+y)]*/
freq_img = np.fft.fft2(img0)
output = np.real(np.fft.ifft2(np.multiply(freq_img, freq_filter)))
/*the multiplication of the FT of two images*/
...  /*using the way above to get M(x,y)*[(-1)^(x+y)]*/
output = output[0:input.shape[0],0:input.shape[1]]  /*cropping*/
```

3. Gaussian Lowpass Filter

```
/*The process of using GLPF is the same as that of ILPF, except the process of generating the filter which is written below*/
for i in (0,M):  /*M = freq_filter.shape[0]*/
    for j in (0,N):  /*N = freq_filter.shape[1]*/
        freq_filter[i][j] = np.exp(-cal_D2(i,j,M,N)/(2*np.power(D0,2)))
/*cal_D2 is used to calculate the square distance between (u,v) and the midpoint of the filter*/
```

4. Butterworth Notch Filter

```
img0 = np.pad(input,(padding_coeffcients),'constant')
freq_filter = np.ones(img0.shape)
for i in (0,M):  /*M = freq_filter.shape[0]*/
    for j in (0,N):  /*N = freq_filter.shape[1]*/
        a = cal_D(i,j,M,N,u_1,v_1)
    /*calculate the distance between (u,v) and the center of the notch filter*/
        b = cal_D(i,j,M,N,-u_1,-v_1)
        c = cal_D(i,j,M,N,u_2,v_2)
        d = cal_D(i,j,M,N,-u_2,-v_2).../*8 varaiables in total*/
        freq_filter[i][j] = 1/(1+np.power(D0/a,2*n))* 1/(1+np.power(D0/b,2*n))*1/(1+np.power(D0/c,2*n))* 1/(1+np.power(D0/d,2*n))...
for i,j in (0,img0.shape[0]), (0,img0.shape[1]):
    img0[i][j] = img0[i][j]*np.power(-1,i+j)  /*get f(x,y)*[(-1)^(x+y)]*/
freq_img = np.fft.fft2(img0)
output = np.real(np.multiply(freq_img,freq_filter))
...  /*using the way above to get g(x,y)*[(-1)^(x+y)]*/
output = output[0:input.shape[0],0:input.shape[1]]  /*cropping*/
```

4. Result & Analysis

It has to be mentioned that, to facillate comparison, I change the size of the images used in this report. The actual results are appended in the compressed package.

1. Sobel Filter

> Fig. 1. The left image is the original image. The center image is the result plus the input image in the spatial domain, the right is that in the frequency domain

> Fig. 2. The left image is the result gradient image in the spatial domain, the right is that in the frequency domain

The sharpening effect of Sobel filter is apparent. Also, it can be found that the two results resemble each other except a small difference appearing at the second window from the right. There are some disordered points in the result from the frequency filter. However, it could still be confirmed that using both methods has same function.

Despite the same function, the method using frequency domain is better. The algorithm of spatial filter I write takes 3.4s while that of frequency filter takes 2.4s. Using filter frequency domain is able to reduce the amount of calculation.

2. Ideal Lowpass Filter

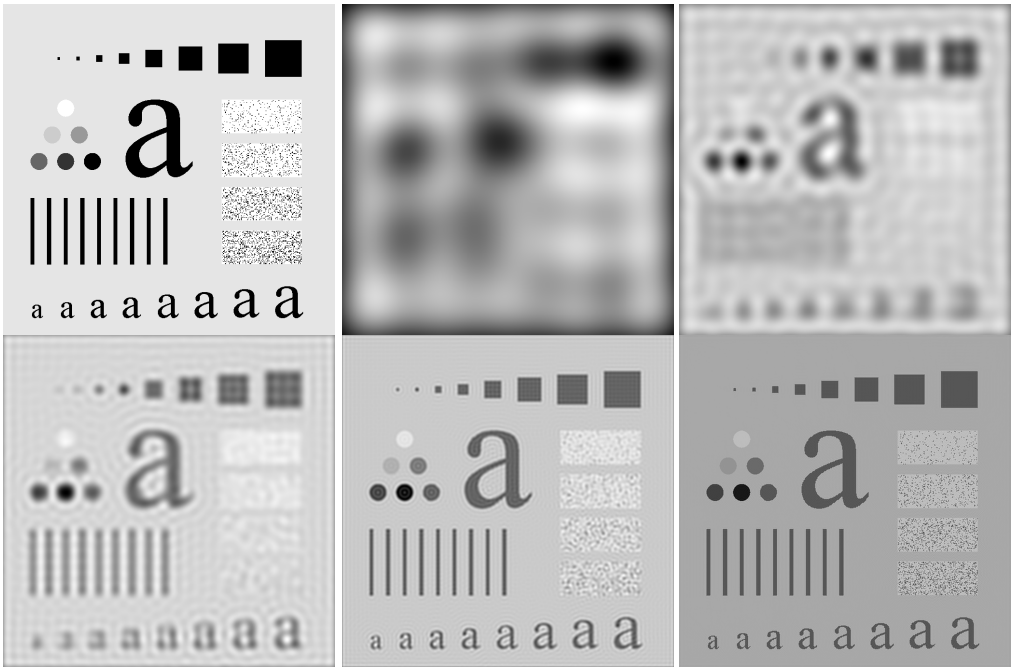


Fig. 3. From left to right, up to down: the original image, result of filtering with ILPF with cutoff frequencies set at radii values 10, 30, 60, 160, 460

The ILPF has great unsharpening effct. With the cutoff frequency decreasing, the noise becomes weaker and the letters become vague more slowly than the noise. This is because the distribution of intensity of the letters are denser. Additionally, many rings around the patterns appear in the unsharpened images, which is called the ring effect. The explanation will be included in later analysis.

3. Gaussian Lowpass Filter

Fig. 4. From left to right, up to down: the original image, result of filtering with GLPF with cutoff frequencies set at radii values 30, 60, 160

The unsharpening function of GLPF is similar to that of ILPF. However, the ring effct does not appear. The visual effect of the result of GLPF are better than that of ILPF.

4. Butterworth Notch Filter

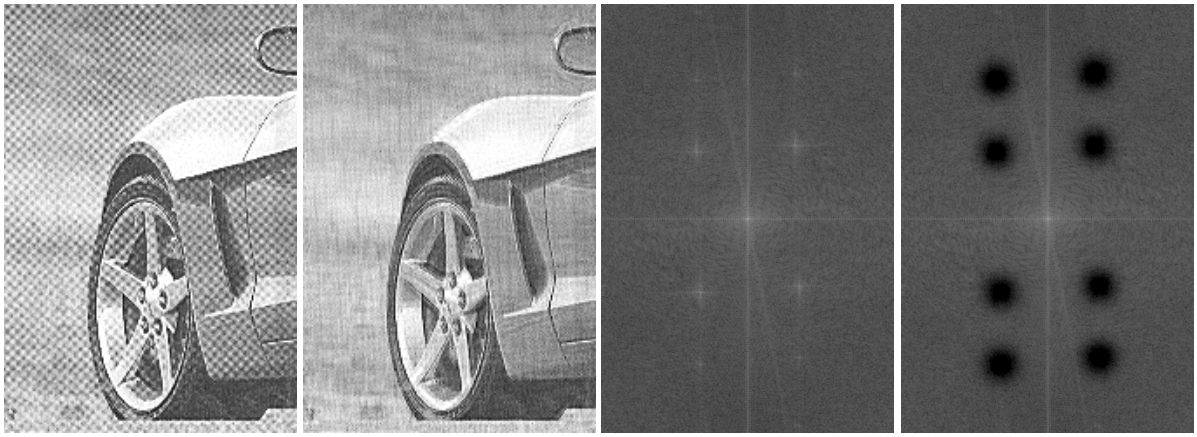


Fig. 5. From left to right: the original image, result of filtering with Butterworth notch reject filter with cutoff frequencies set at radii value 30, the spectrum of the originale image, Butterworth notch reject filter multiplied by the Fourier Transform of the original image

The initial image shows a moiré pattern which is unwanted. Processing it in frequency domain is convenient and efficient. It just need to block 8 frequency components in the spectrum. Using Butterworth notch reject filter reduces the noise a lot.

At first, I want to write a general algorithm to finish the process of reducing noise. I try to find the local extrema of the spectrum, but the method `scipy.signal.argrelextrema()` tends to give strange answers and the algorithm cannot fix which extremum is right. Finally, I failed. It needs much effort to find a general method.

5. Four Questions

1. Explain why perform a shift in Step 4 on slide 81 of Lecture 4 in the first Exercise.

In this case, the initial Sobel mask is translated to the center of an array of zeros to generate an odd symmetric mask. So after getting the Fourier Transform of it, the reault need to be multiplied by $(-1)^{(\mu+\nu)}$ to translate the nonzero part of the mask to the origin.

2. Explain what cause the ring effect in the ideal filtering. Design an experiment to verify your reasoning.

I think the reason is that the IDFT of ILPF is a 2D sinc function, in which many positive and negative values appear in turn. The convolution of a 1D sinc function and another signal has many ups and downs across the horizontal axis. So in 2D, this phenomenon may also occur. In image, those ups and downs are represented as white and black rings around a center. The image expression of a 2D sinc function is shown below.

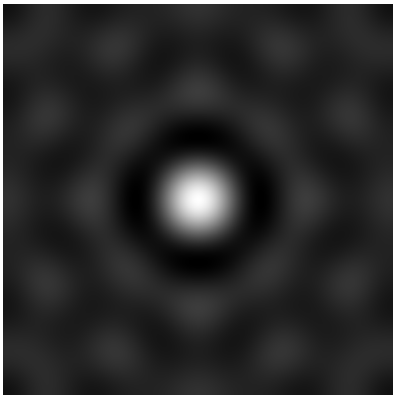
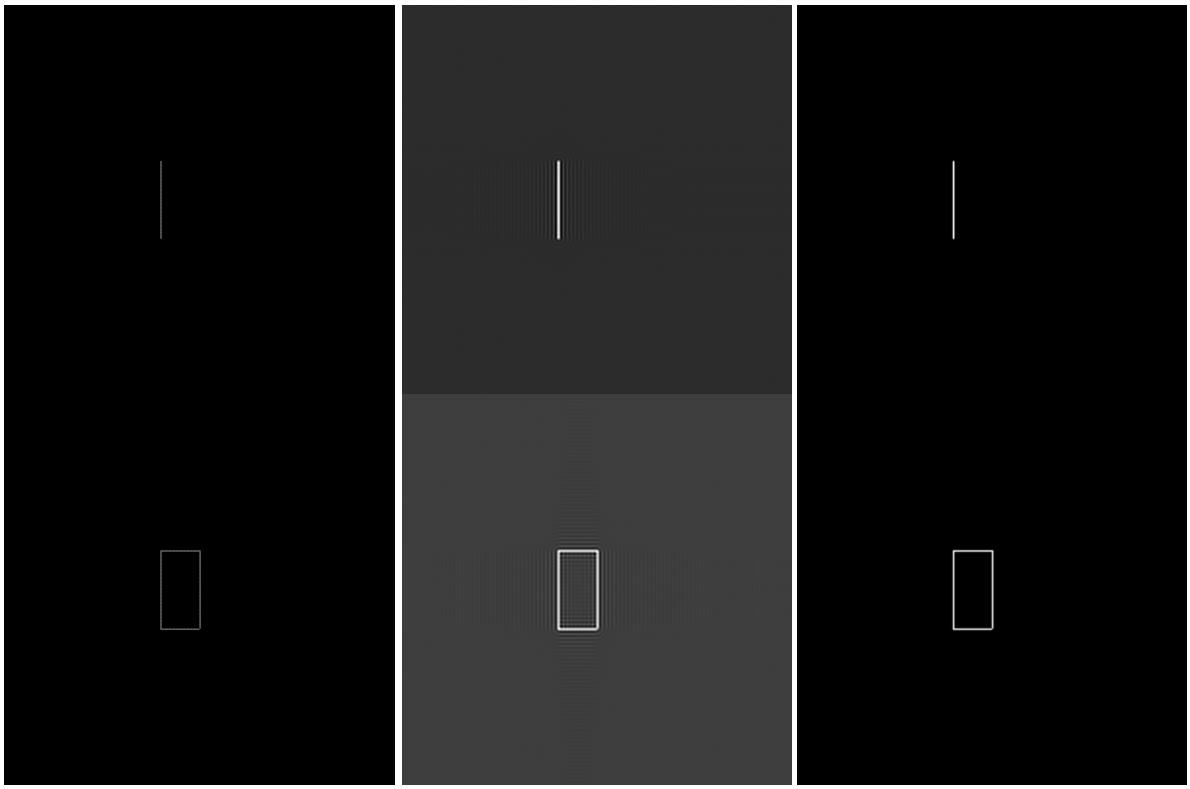


Fig. 6. Representation in the spatial domain of an ILPF of radius 5 and size 1000x1000

It can be seen that many rings appear in the image.

Take a line or a square in an image as an example. The results of the convolution of an image with a single line and the ILPF and that of an image with a single square and ILPF or GLPF are shown below. The lowpass filters have size of 1000x1000 and radius of 100.



> Fig. 7. From left to right, up to down: the original image with a single line, the image before filtered by an ILPF and a GLPF, the original image with a single square, the image before filtered by an ILPF and a GLPF

The patterns in the two images after ILPF have many rings around their edges but those after GLPF have no rings. The IDFT of the ILPF has positive and negative values while the IDFT of the GLPF is always positive. So it can be further deduced that the negative part of the sinc function causes the ring effect.

3. In the above implementation 4, how the parameters in the notch filters are selected, and why.

I have mentioned above that I failed to find a general method to block the noise, then I wanted to just locate the four points on the top and calculate their symmetric points. This still bothered me because I did not find a useful algorithm to get the accurate location of them. At last, I used `plt.imshow()` to show the spectrum and use my cursor to find the accurate coordinates of the 8 points. I think this is difficult and a waste of time but I have not figured out a better way.

The radius of the notch is 30. I first used 30 and thought the noise is lowered and the image is clearer. By increasing the radius, I found the image becomes vague. By decreasing it, the noise becomes apparent.

4. Explain why $H(\mu, \nu)$ has to be real and symmetric in the Step 5 on slide 71 of Lecture 4, which is also the case for all the filter used in this laboratory.

$H(\mu, \nu)$ being symmetric guarantees $h(x, y)$ to be real, which is a common sense.

$H(\mu, \nu)$ being real guarantees no effect on the phase of the original image. When $H(\mu, \nu)$ is real, $g(x, y) = \mathfrak{F}^{-1}[H(\mu, \nu)|F(\mu, \nu)|e^{j\phi(\mu,\nu)}]$. When $H(\mu, \nu)$ is not real, the phase of the original changes a lot, $g(x, y) = \mathfrak{F}^{-1}[|H(\mu, \nu)||F(\mu, \nu)|e^{j\phi_f(\mu,\nu)+j\phi_h(\mu,\nu)}]$. Since phase stores much information about the image like edges and shapes, we must keep it unchanged during processing an image.

5. Conclusion

In this laboratory, I process images in the frequency domain. I find both the efficiency and the limit of using frequency domain. In addition, I encounter difficulties to create a general method that can be applied to many images. Maybe deep learing with trained nueral network can figure out this problem and so many other problems. I think this is why it continues to thrive.