Lab 5 Frequency Domain Filtering

by Wang Zimeng
12011711@mail.sustech.edu.cn
Southern University of Science and Technology

1 Introduction

1.1 Overview

Frequency domain filtering is a valuable technique utilized in digital image processing to enhance or manipulate images. Initially, the image is transformed from the spatial domain into the frequency domain via the Fourier transform, which facilitates the analysis of the image's frequency content, enabling the identification of essential information regarding its structure and features.

1.2 Sobel kernal,Gaussian and Ideal and Butterworth filter

Sobel kernal is a designed kernal used for edge detection which is simple and effective. It is a 3*3 matrix that is applied to each pixel in the image to compute the gradient of intensity. There are two componets, vertical and horizontal, in Sobel kernal that oriented to dectect vertiacal and horizontal edges in image. It is defined as: $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$. To compute the magnitude of gradient, the vertical and horizontal detector is applied seperately and then add together.

The key idea of GaussianãĂĄIdeal and Butterworth filter are the same : to suppress unwanted frequency band and pass the other band. The only difference of these thress is the edge of cutoff frequency. When appling ideal filter with a sharp edge, ringing effect will appear due to insufficient sample rate. Gaussian and Butterworth filter, on the othwewise, have a more smooth turning point and will attenuate this ringing effect.

1.3 Brief view of result

In this lab, the frequency sobel filter is compared with sobel kernal in time domain. Through the result, there is no conspicuous difference as expected.

Both lowpass and highpass Gaussian Ideal and Butterworth filters are operated to seperate rough image and details. It's another perspective of edge detection in frequency domain.

Last of all, the notch filter is designed depending on the image input to denoise preiodic noises in frequency domain.

2 Experiment

Overall steps for spatial-frequency transformation

i. Given an input image $f(x, y)$ of size $M \times N$, obtain the parameters P and Q. Typically, $P = 2M$ and $Q = 2N$.

ii. Form a padded image, $f_p(x, y)$ of size $M \times N$ by appending the necessary number of zeros to $f(x, y)$.

iii. Multiply $f_p(x, y)$ by -1^{x+y} to center its transform.

iv. Compute the DFT, $F(\mu, \nu)$ of the image from step 3.

v. Generate a real, symmetric filter function, $H(\mu, \nu)$, of size $P \times Q$ with center at coordinates (P/2, Q/2).

vi. Form the product$G(\mu,\nu)=H(\mu,\nu)F(\mu,\nu)$ using array multiplication.

vii. Obtain the processed image $g_p(x,y)=\text{real}[\mathcal{F}^{-1}(G(\mu,\nu))](-1)^{x+y}$.

Explain why $H(\mu,\nu)$ has to be real and symmetric in the Step 5 on slide 69 of Lecture 4, which is also the case for most of the filters used in this laboratory. However, there is an exception. Explain the exception.

While filtering, we don't want to distort the shape and intensity of image. So a zero pahse shift filter is necessary which indecates that $H(\mu,\nu)$ is **real**.

Besides, the imaginary part means nothing in real world, and we don't want a convolution with complex kernal. Thus the mask needs to be **symmetric** to be a real signal in spatial domain.

Sometimes, we need to rotate the image while processing i.e. QR code colliboration. To attain this, we need a complex mask to manipulate the phase angle in frequency domain which is related to image rotation.

2.1 Sobel filter in spatial and frequency domain

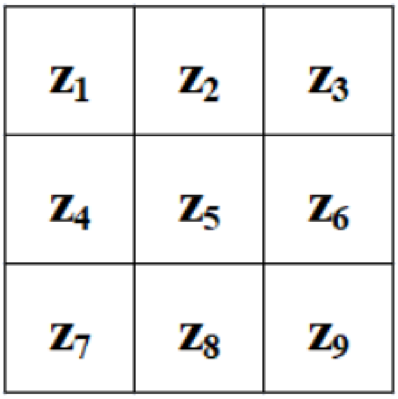
2.1.1 Mathmatical explanation

Based on the idea of using gradient to detect edges, we calclutae the second order gradient on both x and y directions.

$$g_x=\frac{\partial f}{\partial x}=|(z_7+2z_8+z_9)-(z_{31}+2z_2+z_3)|$$

$$g_y=\frac{\partial f}{\partial y}=|(z_3+2z_6+z_9)-(z_1+2z_4+z_7)|$$

The weight of $z_2\, z_4\, z_6$ and z_8 is larger since the Euclidean distance of this four is smaller than the other 4.



z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Figure 1. Pixels in Image

In geometric, the magnitude of $\nabla f=\sqrt{g_x^2+g_y^2}$. But for computation, the magtitude of gradient can be approximatly represented as $\nabla f=|g_x|+|g_y|$ using Manhattan Distance.

In frequency domain, we first need to construct the sobel kernal in the same size. According to the demand of symmetric, the sobel kernal is put in the middle of padded $P\times Q$ image. After fourier transformation, we simply multiply these two together and bring the image back to spatial domain.

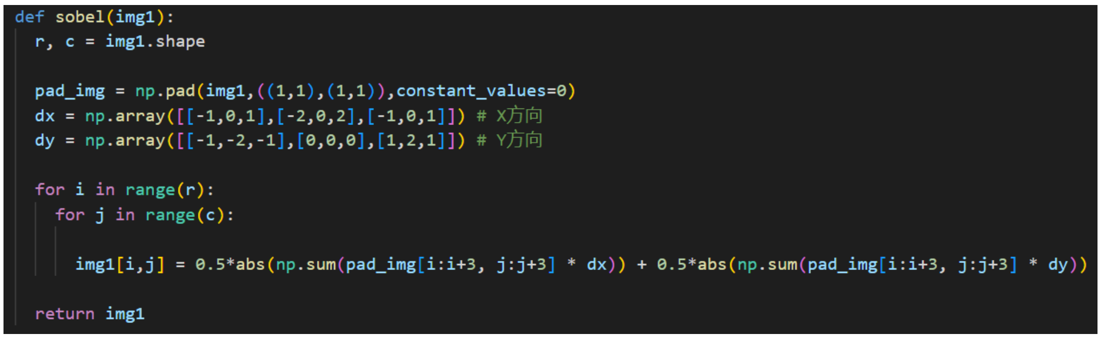
2.1.2 Implement

As introduced above, the pseudo code is:

Pseudo Code:

In spatial domain:

```
sobel_x, sobel_y  <= sobel kernal
img <= input image
pad_img <= padded image with P*Q size
h, w <= image size

for i in h:
    for j in w:
        img[i,j] <= |pad_img[i:i+3,j:j+3]*sobel_x| +
|pad_img[i:i+3,j:j+3]*sobel_y|
```

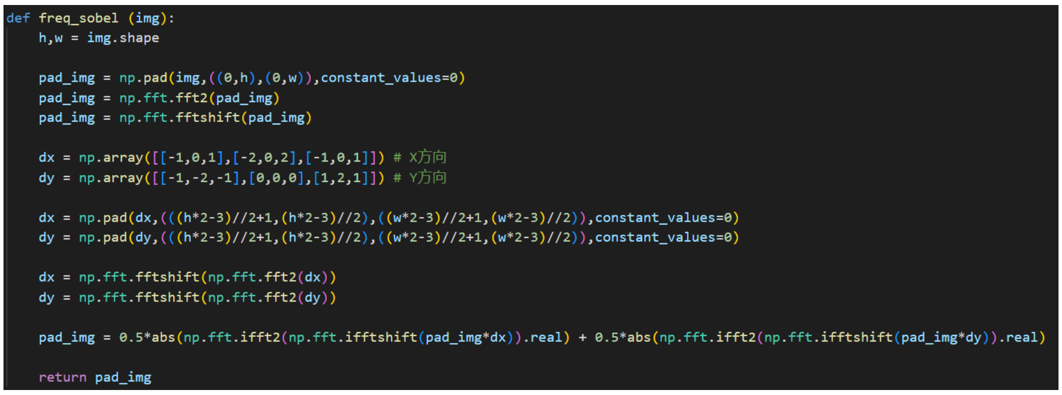


```python
def sobel(img1):
    r, c = img1.shape

    pad_img = np.pad(img1,((1,1),(1,1)),constant_values=0)
    dx = np.array([[-1,0,1],[-2,0,2],[-1,0,1]]) # X方向
    dy = np.array([[-1,-2,-1],[0,0,0],[1,2,1]]) # Y方向

    for i in range(r):
        for j in range(c):

            img1[i,j] = 0.5*abs(np.sum(pad_img[i:i+3, j:j+3] * dx)) + 0.5*abs(np.sum(pad_img[i:i+3, j:j+3] * dy))

    return img1
```

Figure 2.

In frequency domain:

```
sobel_x, sobel_y <= padded sobel kernal
freq_img <= Fourier transform and shift of input image
freq_sobel_x, freq_sobel_y <= Fourier transform and shift of sobel kernal

freq_img = freq_sobel_x*freq_img + freq_sobel_y*freq_img
img_back <= inverse Fourier transform of freq_img
```



```python
def freq_sobel (img):
    h,w = img.shape

    pad_img = np.pad(img,((0,h),(0,w)),constant_values=0)
    pad_img = np.fft.fft2(pad_img)
    pad_img = np.fft.fftshift(pad_img)

    dx = np.array([[-1,0,1],[-2,0,2],[-1,0,1]]) # X方向
    dy = np.array([[-1,-2,-1],[0,0,0],[1,2,1]]) # Y方向

    dx = np.pad(dx,(((h*2-3)//2+1,(h*2-3)//2),((w*2-3)//2+1,(w*2-3)//2)),constant_values=0)
    dy = np.pad(dy,(((h*2-3)//2+1,(h*2-3)//2),((w*2-3)//2+1,(w*2-3)//2)),constant_values=0)

    dx = np.fft.fftshift(np.fft.fft2(dx))
    dy = np.fft.fftshift(np.fft.fft2(dy))

    pad_img = 0.5*abs(np.fft.ifft2(np.fft.ifftshift(pad_img*dx)).real) + 0.5*abs(np.fft.ifft2(np.fft.ifftshift(pad_img*dy)).real)

    return pad_img
```

Figure 3. Frequency Sobel

As we can see in the result part, the two images are identical.

2.2 Mathmatical Explanation for Frequency Filter

Indeed, the core of frequency filter is to onstruct a function to cutdown some noise frequency yet leaving the image unchanged.

For Ideal Lowpass Filter, $H(\mu,\nu)$ is:

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

For Gaussian Lowpass Filter, $H(\mu,\nu)$ is :

$$H(\mu,\nu)=e^{-\frac{D(\mu,\nu)^2}{2D_0^2}}$$

For Butterwoth Highpass Filter with the order n, $H(\mu,\nu)$ is :

$$H(\mu,\nu)=\frac{1}{1+[D_0/D(\mu,\nu)]^{2n}}$$

where

$$D(\mu,\nu)=\left[\left(\mu-\frac{P}{2}\right)^2+\left(\nu-\frac{Q}{2}\right)^2\right]^{\frac{1}{2}}$$

D_0 is the cut off frequency.

For Highpass Filter, we just need to adjust a bit. For a lowpass filter, we can also derive the lowpass filter from the equation.

$$H_{\text{high}}(\mu,\nu)=1-H_{\text{low}}(\mu,\nu)$$

Then we can make the low frequency band blocked and high frequency passed.

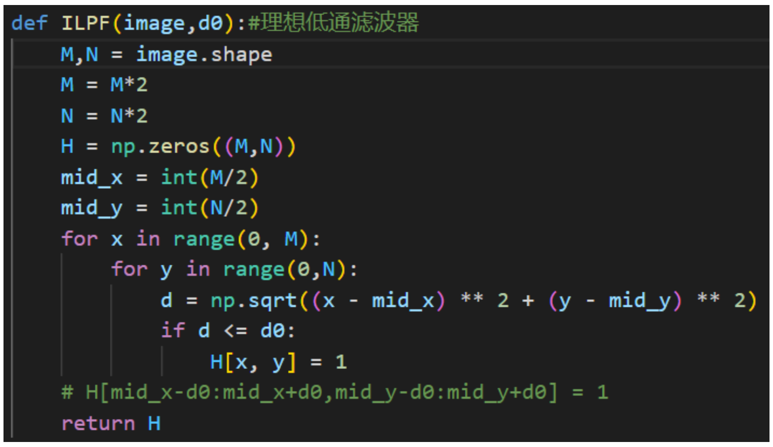
Implement of Frequency Filter

Pseudo Code:

```
H(m,n) <= mask of filter
img(m,n) <= input image

freq_img = Fourier Transformation and shift of img
freq_img = freq_img * H
back_imf = Inverse Fourier Transformation and shift of freq_img
```

For each filter, I define a related kernal function. Then, use **Conv** function to do convolution and inverse transformation.

The construction of filters are trivial, so only ideal low pass filter is presented.



```
def ILPF(image,d0):#理想低通滤波器
    M,N = image.shape
    M = M*2
    N = N*2
    H = np.zeros((M,N))
    mid_x = int(M/2)
    mid_y = int(N/2)
    for x in range(0, M):
        for y in range(0,N):
            d = np.sqrt((x - mid_x) ** 2 + (y - mid_y) ** 2)
            if d <= d0:
                H[x, y] = 1
    # H[mid_x-d0:mid_x+d0,mid_y-d0:mid_y+d0] = 1
    return H
```

Figure 4. Idea Lowpass Filter

Optimization

More than that, the commented-out code in the function suggests an alternative way of generating the mask by array slicing while using approximate the Manhanton distance from each pixel to the center of the matrix. this method can be computationally cheap for large images and is necessary sometimes since the mask can be created more efficiently and the difference is not that large.

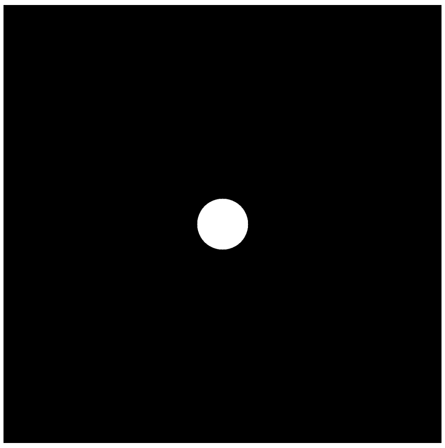


Figure 5. Euclidean

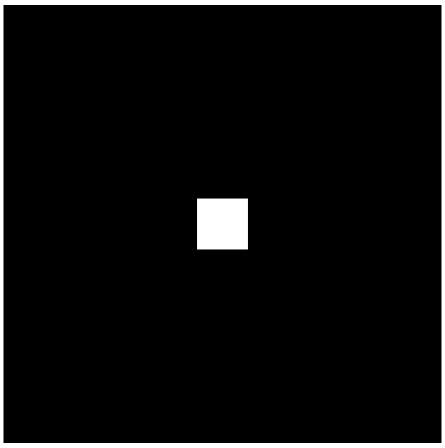


Figure 6. Square

The Time expenses for round filter is way much larger than square or array slicing.

$10-\text{times run Euclidean}$	$10-\text{times run array}-\text{slicing}$	efficiency
$34.24s$	$1.922s$	1781%

As we can see from figures below, appling square or euclidean for filtering didn't cause any notable difference.

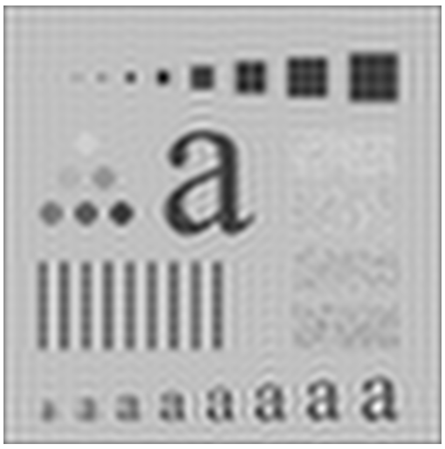


Figure 7. Euclidean $D_0 = 60$

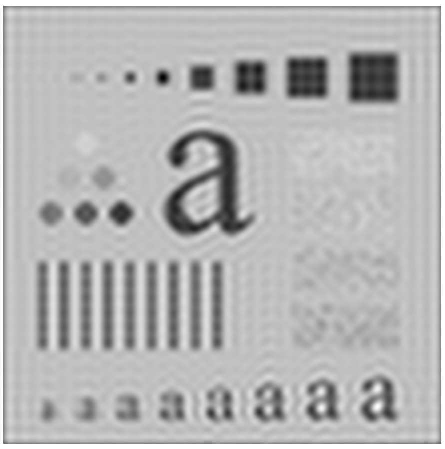
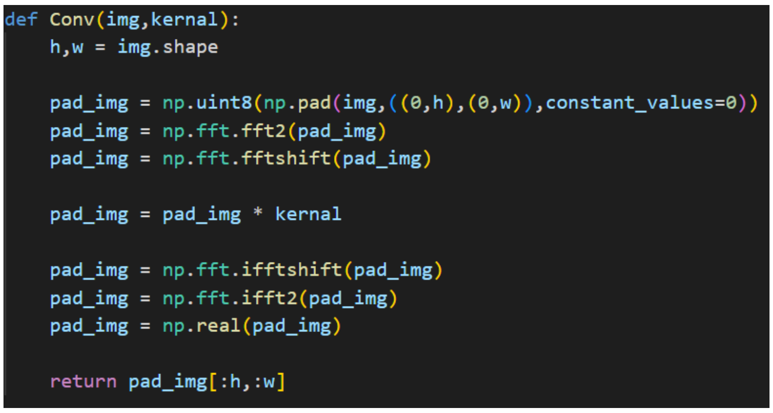


Figure 8. Square $D_0 = 60$



```python
def Conv(img,kernal):
    h,w = img.shape

    pad_img = np.uint8(np.pad(img,((0,h),(0,w)),constant_values=0))
    pad_img = np.fft.fft2(pad_img)
    pad_img = np.fft.fftshift(pad_img)

    pad_img = pad_img * kernal

    pad_img = np.fft.ifftshift(pad_img)
    pad_img = np.fft.ifft2(pad_img)
    pad_img = np.real(pad_img)

    return pad_img[:h,:w]
```

Figure 9. Convolution

2.3 Butterworth Notch Filter

2.3.1 Mathematical Explanation

For notch filter, it can be represented as the combination of low-pass filter or high-pass filter.
For High-pass Filter $H(\mu,\nu)$, notch filter is :

$$H_{\text{notch}}(\mu,\nu)=\sum_{k=1}^{N}\left(H_k(\mu,\nu)+H_{-k}(\mu,\nu)\right)$$

For Low-pass Filter $H(\mu,\nu)$, notch filter is :

$$H_{\text{notch}}(\mu,\nu)=\prod_{k=1}^{N}H_k(\mu,\nu)H_{-k}(\mu,\nu)$$

where H_{-k} represent the filter with the center point symmetric about H_k , D_k is :

$$D_k(\mu,\nu)=\left[\left(\mu-\frac{P}{2}-\mu_k\right)^2+\left(\nu-\frac{Q}{2}-\nu_k\right)^2\right]^{\frac{1}{2}}$$

(μ_k,ν_k) is the center of filter $H_k(\mu,\nu)$.
Finally the band reject notch filter can be represented :

$$H_{\text{notch}}(\mu,\nu)=\prod_{k=1}^{N}\left(\frac{1}{1+[D_0/D_k(\mu,\nu)]^{2n}}\right)\left(\frac{1}{1+[D_0/D_{-k}(\mu,\nu)]^{2n}}\right)$$

Band pass filter can be represented as :

$$H_{\text{notch}}(\mu,\nu)=\sum_{k=1}^{N}\left(\frac{[D_0/D_k(\mu,\nu)]^{2n}}{1+[D_0/D_k(\mu,\nu)]^{2n}}+\frac{[D_{-k}(\mu,\nu)/D_0]^{2n}}{1+[D_{-k}(\mu,\nu)/D_0]^{2n}}\right)$$

However, there is another high pass Butterworth filter :

$$H(\mu,\nu)=\frac{1}{1+[D(\mu,\nu)/D_0]^{2n}}$$

In the lab, I used this butterworth filter and the corresponding notch is :
band reject:

$$H_{\text{notch}}(\mu,\nu)=\sum_{k=1}^{N}\left(\frac{1}{1+[D_k(\mu,\nu)/D_0]^{2n}}+\frac{1}{1+[D_{-k}(\mu,\nu)/D_0]^{2n}}\right)$$

band pass:

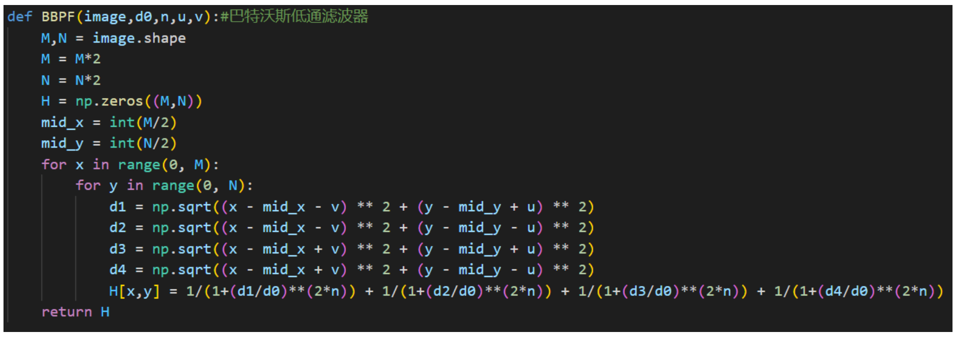
$$H_{\text{notch}}(\mu,\nu) = \sum_{k=1}^{N} \left(\frac{[D_k(\mu,\nu)/D_0]^{2n}}{1+[D_k(\mu,\nu)/D_0]^{2n}} \right) \left(\frac{[D_{-k}(\mu,\nu)/D_0]^{2n}}{1+[D_{-k}(\mu,\nu)/D_0]^{2n}} \right)$$

2.3.2 Implement

Based on the idea that the noise is periodic, we only need to specify one of the noise center and then use symmetric property to construct the denoise mask.

Pseudo Code:

```
u,v <= noise frequency size
H(u,v) <= butterworth lowpass filter with center in (u,v)

H = H(u,v) + H(-u,v) + H(-u,-v) + H(u,-v)
```



```
def BBPF(image,d0,n,u,v):#巴特沃斯低通滤波器
    M,N = image.shape
    M = M*2
    N = N*2
    H = np.zeros((M,N))
    mid_x = int(M/2)
    mid_y = int(N/2)
    for x in range(0, M):
        for y in range(0, N):
            d1 = np.sqrt((x - mid_x - v) ** 2 + (y - mid_y + u) ** 2)
            d2 = np.sqrt((x - mid_x - v) ** 2 + (y - mid_y - u) ** 2)
            d3 = np.sqrt((x - mid_x + v) ** 2 + (y - mid_y + u) ** 2)
            d4 = np.sqrt((x - mid_x + v) ** 2 + (y - mid_y - u) ** 2)
            H[x,y] = 1/(1+(d1/d0)**(2*n)) + 1/(1+(d2/d0)**(2*n)) + 1/(1+(d3/d0)**(2*n)) + 1/(1+(d4/d0)**(2*n))
    return H
```

Figure 10.

After mask generation, we just need to do convolution using **Conv** function mentioned above.

3 Result and Analysis

3.1 Sobel filter

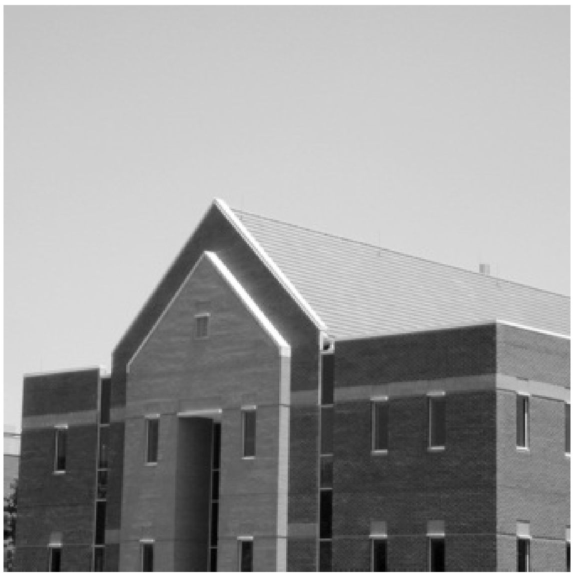


Figure 11. Q5_1

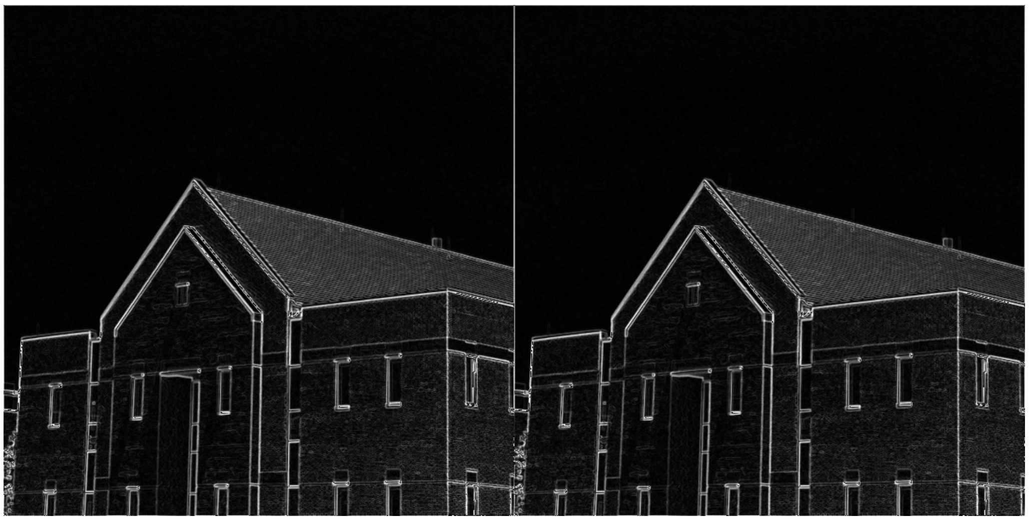


Figure 12. Spatial and Frequency Sobel

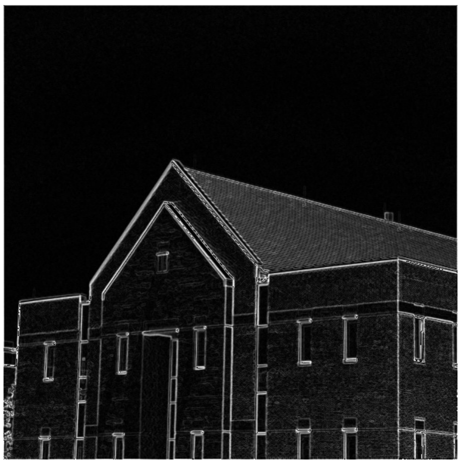


Figure 13. Edge

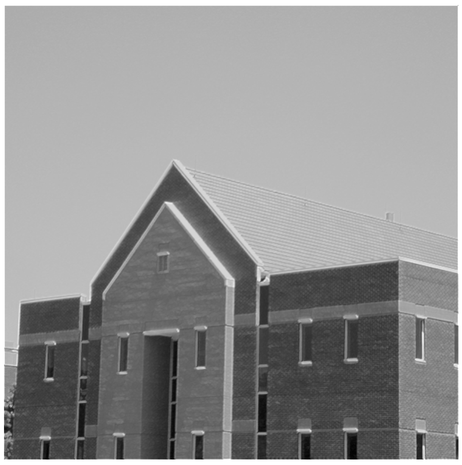


Figure 14. Edge enhancement

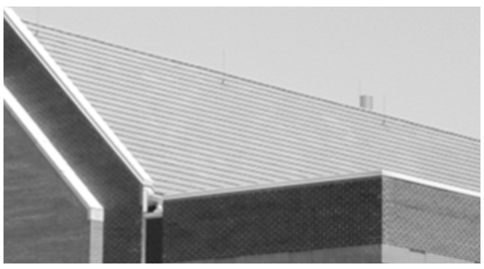


Figure 15. Roof of origin image

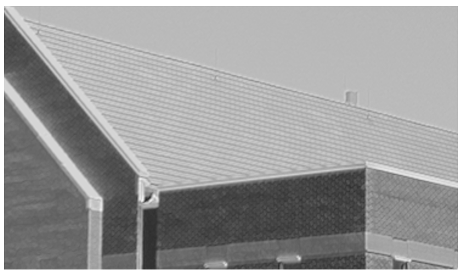


Figure 16. Roof after enhancement

Fig 8. clears shows that the spatial domain sobel is actually the same as sobel mask in frequency domain. So this experiment successfully proved that the two is the same.Fig 10. which is the image after gamma transformation have an enhance edge in the roof.

Explain why perform a shift in Step 4 on slide 79 of Lecture 4 in the first Exercise.

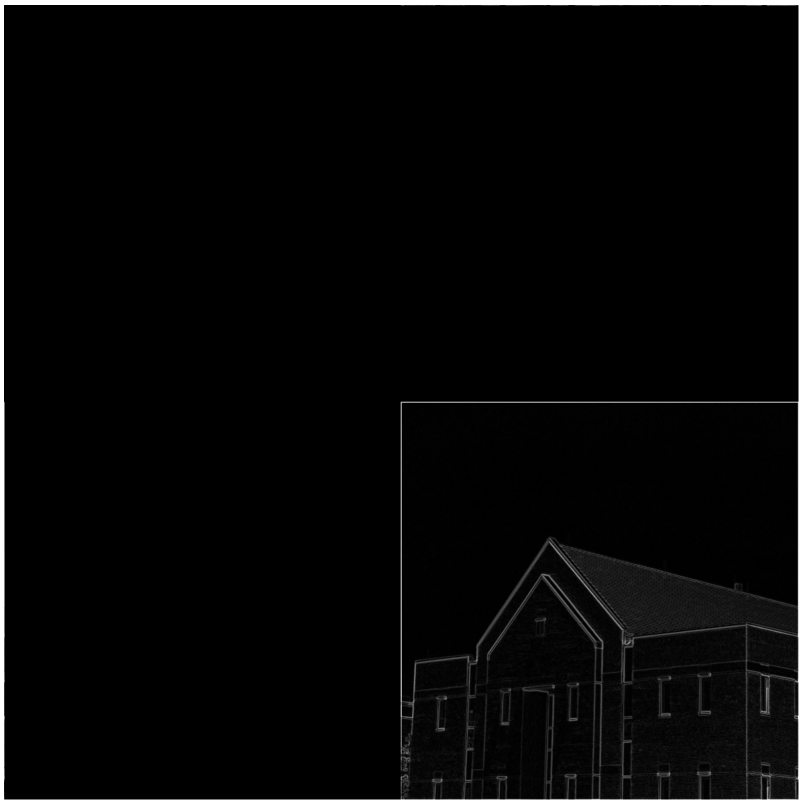


Figure 17.

As I pad the image with same length of original image, the period of the padded image is doubled causing $M' = 2M$ and $N' = 2N$. While putting the keranl in the middle of the image, we actually do a shift in time domain which indicates that :

$$f(x - M, y - N) \Leftrightarrow F(\mu, \nu) e^{-j2\pi\left(\frac{M\mu}{M'} + \frac{N\nu}{N'}\right)} = F(\mu, \nu) \times (-1)^{\mu+\nu}$$

In time domain, it is a shift effect making the original image moves half a period of padded image. So the image will be on the right-down instead of left-up side. To avoid this, we need to multiply the $(-1)^{\mu+\nu}$ again to move the image back.

3.2 Gaussian Filter and Ideal Filter

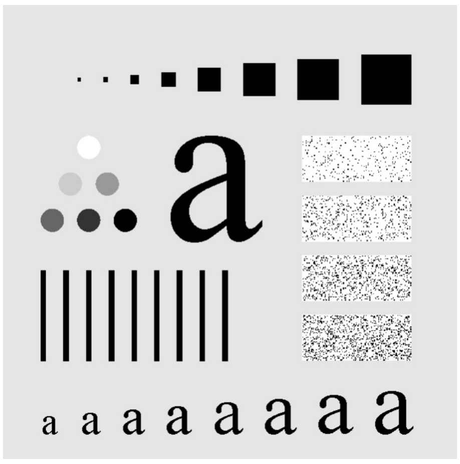


Figure 18. Q5_2

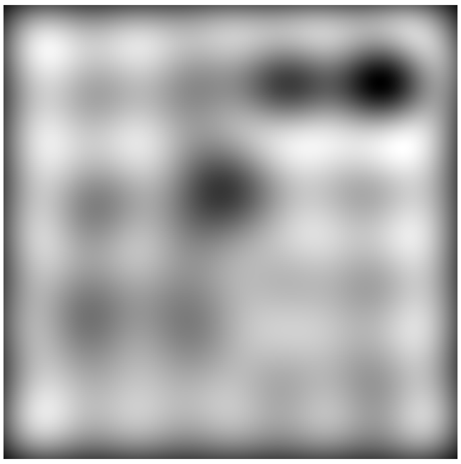


Figure 19. ILPF $D_0 = 10$

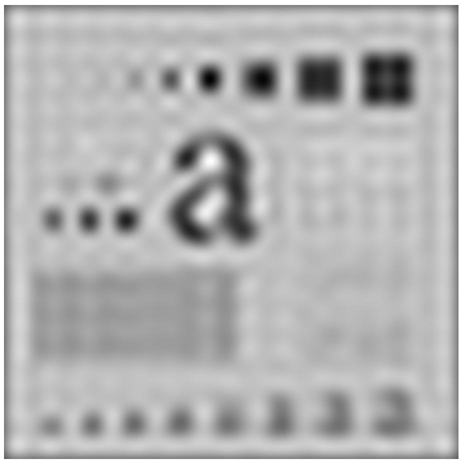


Figure 20. ILPF $D_0 = 30$

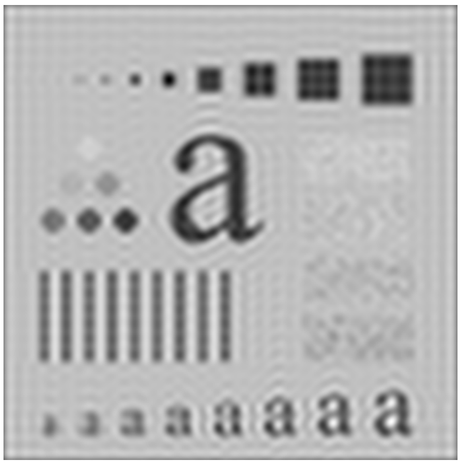


Figure 21. ILPF $D_0 = 60$

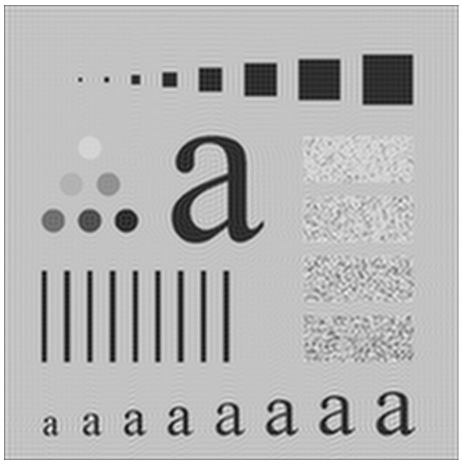


Figure 22. ILPF $D_0 = 160$

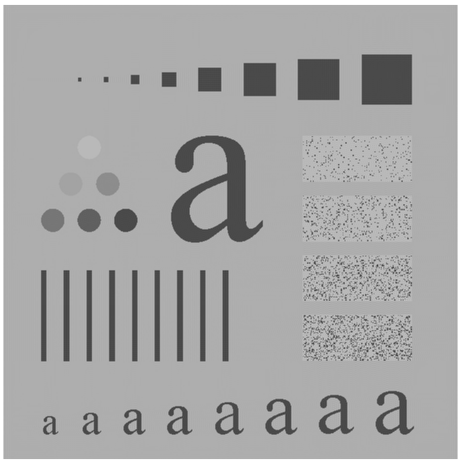


Figure 23. ILPF $D_0 = 460$

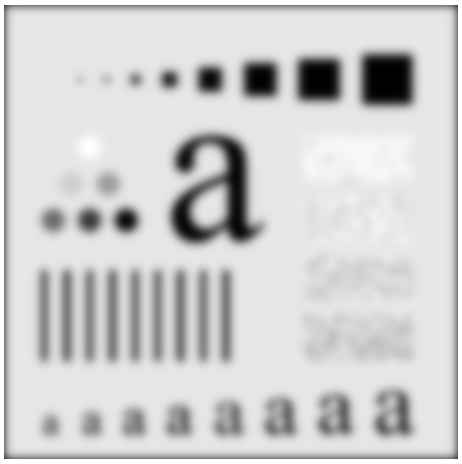


Figure 24. GLPF $D_0 = 30$

From the figure above, we can see that both ILPF and GLPF is getting more and more clear and detailed as the cutoff frequency D_0 increases. Actually the 460 cutoff frequency is almost the original image.

However, as the sample rate of computer could never satisfy the ideal Fourier transformation, the ringing effect appears due to sharp edge of ideal low-pass filter. However, the smooth edge of Gaussian filter would greatly reduce this effect.

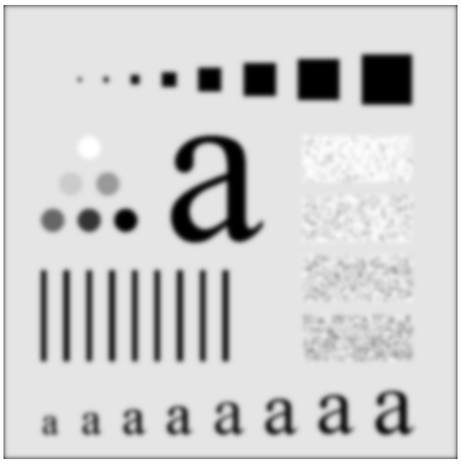


Figure 25. GLPF $D_0 = 60$

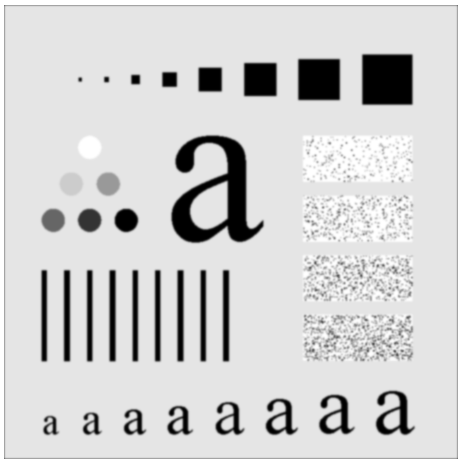


Figure 26. GLPF $D_0 = 160$

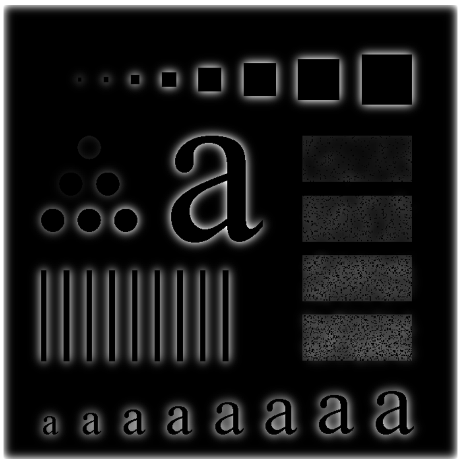


Figure 27. GHPF $D_0 = 30$

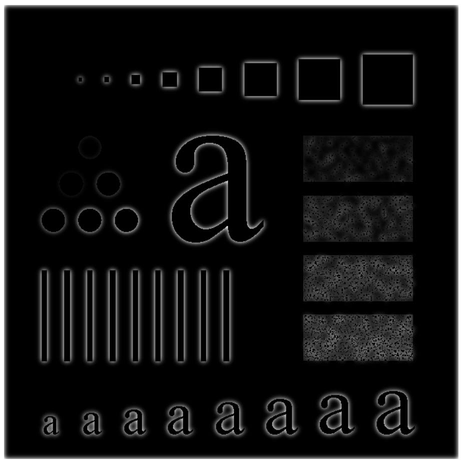


Figure 28. GHPF $D_0 = 60$

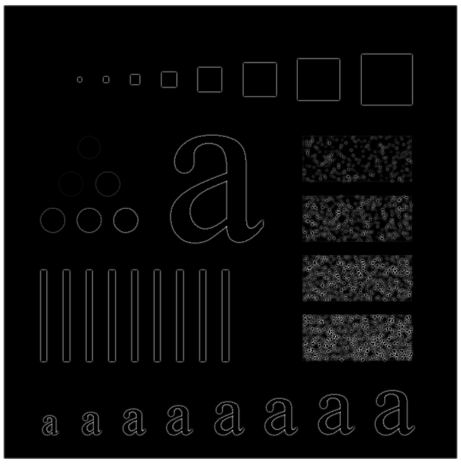


Figure 29. GHPF $D_0 = 160$

In high-pass filtering, as the cut-off frequency D_0 increases, the edges are clearer, the distortion is less, and noises have been correctly filtered out. The high-pass filter is similar to differentiate.

3.3 Butterworth Notch Filter

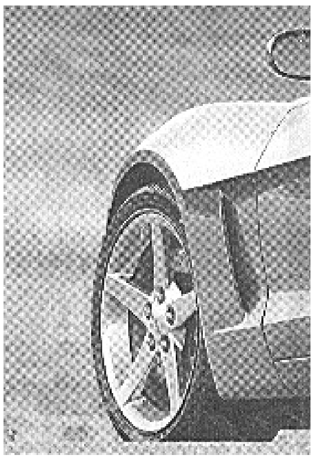


Figure 30. Q5_3

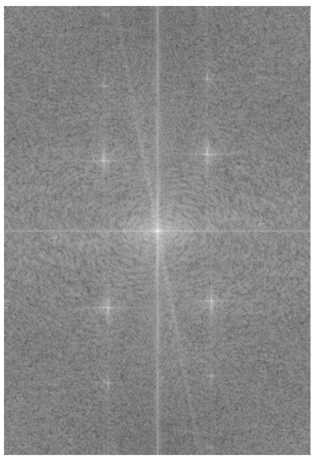


Figure 31. spectrum

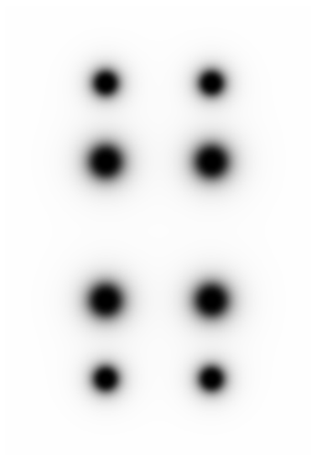


Figure 32. mask

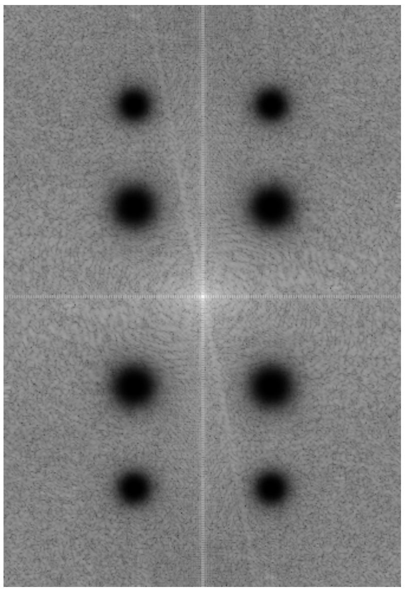


Figure 33.

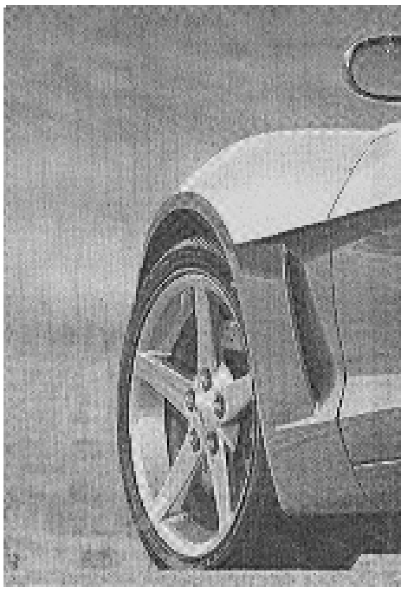


Figure 34.

As we can see from above, the images shows the spectrum of original image, spectrum of mask, spectrum of processed image and final output. In this section, the filter is handcrafted since the noise frequency can be easily spotted from frequency domain. After filtering, the image becomes clear and vivid but the details are maintained.

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

The noise spectrum is irregular and conspicuous in the spectrum. Since there are 8 noise source, we construct 8 mask to denoise.

To select u_k and v_k , I make a iteration and measured roughly by the spectrum and processed image. Finally, the center of noise is found in $(58, 76), (58, 162), (-58, 76), (-58, 162), (58, -76)$, $(58, -162), (-58, -76), (-58, -162)$. The cutoff frequency depends on the size of noise within (15,25).

4 Conclusion

This lab we reproduce the Sobel filter in the frequency domain and compares its results with those obtained from the Sobel filter in the spatial domain. The results show that the two outputs are fully consistent, demonstrating that the result obtained by multiplying the Fourier transform of the input image in the frequency domain and applying the inverse Fourier transform is the same as the result obtained from convolution in the spatial domain.

In addition to the Sobel filter, the lab also implements a Gaussian filter in the frequency domain. The results demonstrate that as the filter radius D_0 of the Gaussian low-pass filter increases, the images becomes obscured, while an increase in the filter radius D_0 of the Gaussian high-pass filter leads to greater details and the enhancing more noises at the same time. The ideal filter creates ringing, while the Gaussian and Butterworth filters do not because of different edges. The second-order BLPF is a good choice for balancing effective low-pass filtering and acceptable levels of ringing while the Butterworth filter has larger derivatives,sharpened edges in another word ,as the orders go up .

Finally, the lab implements the Butterworth notch filter, a man-made filter that uses four notch pairs to accurately eliminate the noises in the input image. The filter and its results are displayed through the frequency spectrum conspicuously.