Lab5 Filtering in the frequency domain

张旭东 12011923

1. Introduction

The frequency of an image is an indicator of the intensity of the change of gray in an image. The Fourier transform has a very obvious physical meaning in practice. In a mathematical sense, the Fourier transform transform the image from the spatial domain to the frequency domain, and its inverse transform transforms the image from the frequency domain to the spatial domain. In other words, the physical meaning of Fourier transform is to transform the grayscale distribution function of an image to the frequency distribution function of an image. The significance of the light and dark points seen on the Fourier spectrum refers to the strength of difference between a point on the image and the point in the neighborhood, which is also called the magnitude of gradient. Generally speaking, the larger the magnitude of gradient, the stronger the brightness of the point. In this way, it is easy to know the distribution of energy of the image by observing the Fourier spectrum. If there are more dark points in the spectrum, the image in spatial domain is relatively soft, which means the difference between each point and the points in the neighborhood and the gradient is relative small. On the other hand, if there are many bright points in the spectrum, the image spectrum in the spatial domain is sharp with sharp boundaries and large difference of pixels on both sides of the boundary.

The filters in frequency domain discussed in this experiment is Sobel filter and Butterworth notch reject filter.

Sobel filter is a kind of high-pass filter, which is also called directional filter because of the direction of it. It can remove the low frequency component in the image and keep the high frequency component in the image, making the the rate of change in the gradient direction more significant, which is used in the edge detection of an image.

Butterworth notch reject filter can be used to remove periodic noise, which means processing small region of the frequency rectangle. Although band-stop filter is also used to remove periodic noise, it attenuate components other than noise. The Butterworth notch reject filter mainly attenuates a certain point and does not have an effect to other components.

2. Analysis and result

2.1Theoretical knowledge

2-D Discrete Fourier Transform and Its Inverse:

$$\begin{aligned} F(u,v) &= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} \\ f(x,y) &= \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})} \end{aligned} \tag{1}$$

Periodicity of the DFT:

For $f(x,y)$ and its DFT $F(u,v)$, we have:

$$\begin{aligned} f(x,y)e^{j2\pi(\frac{u_0x}{M}+\frac{v_0y}{N})} &\Longleftrightarrow F(u-u_0,v-v_0) \\ f(x-x_0,y-y_0) &\Longleftrightarrow F(u,v)e^{-j2\pi(\frac{ux_0}{M}+\frac{vy_0}{N})} \end{aligned} \tag{2}$$

2-D Fourier transform and its inverse are infinitely periodic,so

$$\begin{aligned} F(u,v) = F(u+k_1M,v) = F(u,v+k_2N) = F(u+k_1M,v+k_2N) \\ f(x,y) = f(x+k_1M,y) = f(x,y+k_2N) = F(x+k_1M,y+k_2N) \end{aligned} \tag{3}$$

However, if we just take one period, an uncentered spectrum will be gotten. According to `Formula(2)`, let $u_0 = \frac{M}{2}, v_0 = \frac{N}{2}$, then the spectrum will be moved to the center.

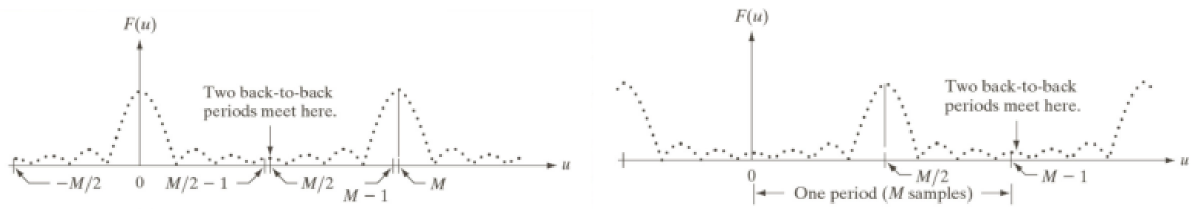


Fig.1 A 1-D DFT(left) and Shifted DFT obtained by multiplying f(x) by (-1)^x before computing F(u)(right)

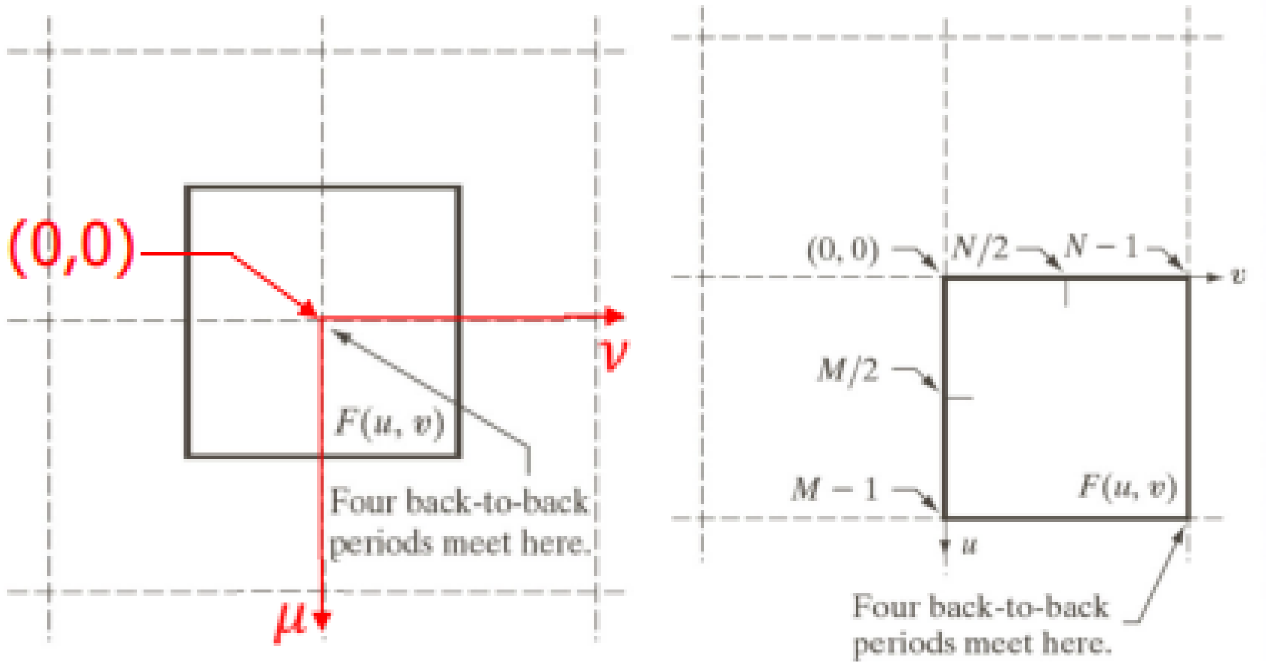

(0,0)
ν
μ
F(u, v)
Four back-to-back periods meet here.
(0, 0)
N/2
N − 1
v
M/2
M − 1
F(u, v)
u
Four back-to-back periods meet here.

Fig.2 A 2-D DFT(left) and Shifted DFT obtained by multiplying f(x,y) by (-1)^(x+y) before computing F(u,v)(right)

Zero padding:

As we all know, convolution operation in the spatial domain means product operation in the frequency domain. And product operation in the spatial domain means convolution operation in the frequency domain.

$$\begin{gathered} f(x,y) * h(x,y) \Longleftrightarrow F(u,v)H(u,v) \\ f(x,y)h(x,y) \Longleftrightarrow F(u,v) * H(u,v) \end{gathered} \tag{4}$$

However, the data from adjacent periods produce wraparound error, yielding an incorrect convolution result. To obtain the correct result, function padding must be used.

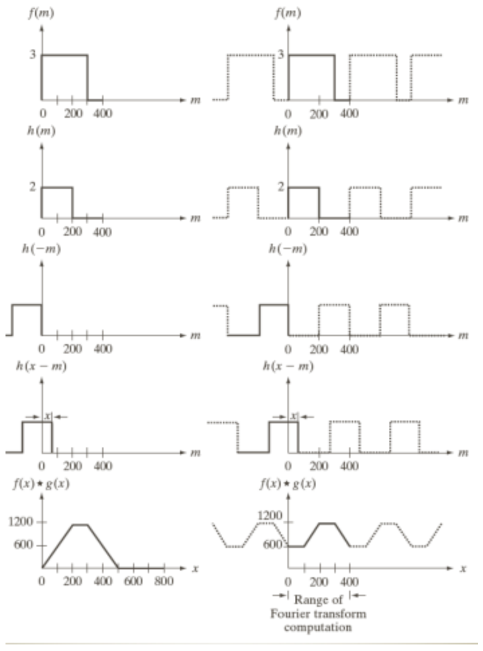


Fig.3 convolution of two discrete functions

Let $f(x,y)$ and $h(x,y)$ be two image arrays of sizes $A\times B$ and $C\times D$ pixels, respectively. Wraparound error in their convolution can be avoided by padding these functions with zeros.

$$f_p(x,y)=\begin{cases} f(x,y) & 0\leqslant x\leqslant A-1 \quad \textit{and} \quad 0\leqslant y\leqslant B-1 \\ 0 & A\leqslant x\leqslant P \quad \textit{or} \quad B\leqslant y\leqslant Q \end{cases} \tag{5}$$

$$h_p(x,y)=\begin{cases} h(x,y) & 0\leqslant x\leqslant C-1 \quad \textit{and} \quad 0\leqslant y\leqslant D-1 \\ 0 & C\leqslant x\leqslant P \quad \textit{or} \quad D\leqslant y\leqslant Q \end{cases} \tag{6}$$

$$\begin{gathered} p\geqslant A+C-1 \\ Q\geqslant B+D-1 \end{gathered} \tag{7}$$

Steps for Filtering in the Frequency Domain:

1. Given an input image $f(x,y)$ of size $M\times N$, obtain the padding parameters P and Q. Typically, $P=2M$ and $Q=2N$.
2. Form a padded image, $f_p(x,y)$ of size $P\times Q$ by appending the necessary number of zeros to $f(x,y)$
3. Multiply $f_p(x,y)$ to center its transform
4. Compute the DFT,$F(u,v)$ of the image from the step 3.
5. Generate a real, symmetric filter function, $H(u,v)$, of size $P\times Q$ with center at coordinates $(P/2,Q/2)$

6. Form the product $G(u,v) = H(u,v)F(u,v)$ using array multiplication
7. Obtain the processed image

$$g_p(x,y) = [real[\varsigma^{-1}[G(u,v)]]](-1)^{x+y} \tag{8}$$

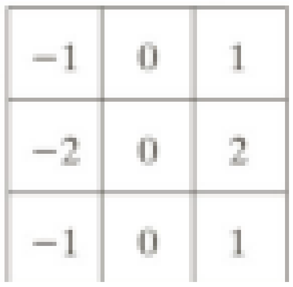
8. Obtain the final processed result, $g(x,y)$, by extracting the $M \times N$ region from the top,left quadrant of $g_p(x,y)$

2.2 Sobel filter

principle: Sobel operator is a discrete differential operator, which is used to calculate the approximate gradient of image gray function. That is, Sobel operator can be used to measure the change of image in vertical and horizontal directions. Due to the discrete characteristics of pixels, Sobel operator provides the approximation of image gradient through pixel difference in horizontal and vertical directions.

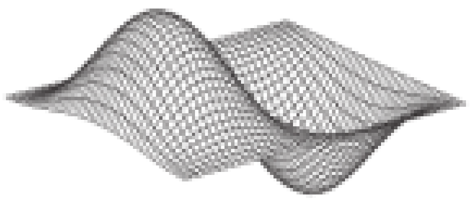
Question formulation:

The Sobel operator is as below:



−1	0	1
−2	0	2
−1	0	1

Fig.4 spatial mask of Sobel



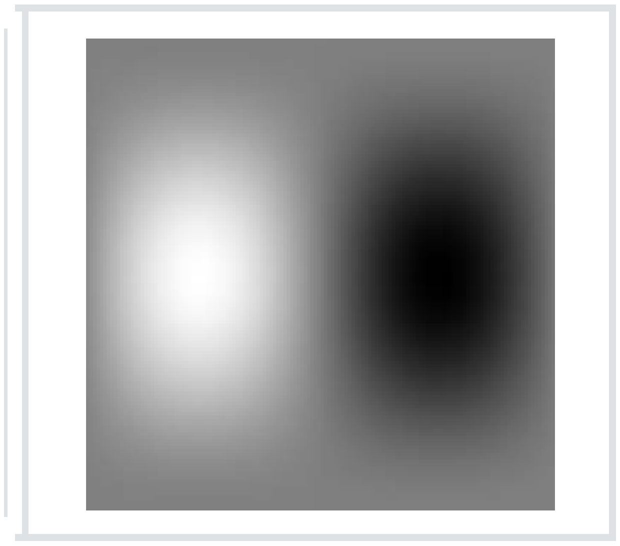


Fig.5 perspective plot of its corresponding frequency domain(left) and filter shown as an image(right)

The convolution of a filter $w(x,y)$ of size $m \times n$ with an image $f(x,y)$ with $M \times N$, denoted $w(x,y) \bigotimes f(x,y)$

$$w(x,y) \bigotimes f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{9}$$

After simplification,

$$w(x,y) \bigotimes f(x,y) = G_x \tag{10}$$

The operation in frequency domain is:

$$O(u,v) = W(u,v) F(u,v) \tag{11}$$

For generating from a given spatial filter, the steps of generating $H(u,v)$ is

1. pad the image and the filter

$$f_p(x,y) = \begin{cases} f(x,y) & 0 \leqslant x \leqslant M-1 & and & 0 \leqslant y \leqslant N-1 \\ 0 & M \leqslant x \leqslant M+m-1 & or & N \leqslant y \leqslant N+n-1 \end{cases} \tag{12}$$

$$h_p(x,y) = \begin{cases} h(x,y) & \frac{M}{2} \leqslant x \leqslant \frac{M}{2}+m-1 & and & \frac{N}{2} \leqslant y \leqslant \frac{N}{2}+n-1 \\ 0 & others \end{cases} \tag{13}$$

2. multiply $h_p(x,y)$ by $(-1)^{x+y}$ to center the frequency domain filter
3. compute the forward DFT of the result in (1)
4. set the real part of the result DFT to 0 to account for parasitic real parts
5. multiply the result by $(-1)^{u+v}$, which is implicit when $h(x,y)$ was moved to the center of $h_p(x,y)$.

pseudo code:

```
#spatial
padimage=np.pad(input_image,((W,W),(H,H)),'symmetric')
for i in range (H,2*H):
    for j in range(W,2*W):
        output_image[i-H,j-W]=sobel_operator
```

```
output_image=normalize(output_image)

#frequency domain
padimage=np.zeros([input_image.shape[0]+mask_H-1,input_image.shape[1]+mask_W-
1])
padimage[0:input_image.shape[0],0:input_image.shape[1]]=input_image
padimage=padimage*((-1)**(x+y))
padimage_DFT=FFT(padimage)

sobel=np.zeros([input_image.shape[0]+mask_H-1,input_image.shape[1]+mask_W-1])

sobel[input_image.shape[0]:input_image.shape[0]+mask_H,input_image.shape[1]:in
put_image.shape[1]+mask_W]=sobel_operator

sobel=sobel*((-1)**(x+y))
sobel_DFT=FFT(sobel)
sobel_DFT.real=0
sobel_DFT=sobel_DFT*((-1)**(u+v))

real_inverse_DFT=real(IFFT(padimage_DFT*sobel_DFT))*((-1)**(x+y))
output_image=real_inverse_DFT[0:input_image.shape[0],0:input_image.shape[1]]
output_image=normalize(output_image)
```

Python code:

```
def Sobel_spatial(input_image):
    H,W=input_image.shape
    output_image=np.zeros([H,W],dtype=np.int32)

    padimage=np.pad(input_image,((W,W),(H,H)),'symmetric')
    padimage=np.array(padimage,dtype=np.int32)

    for i in range(H,2*H):
        for j in range(W,2*W):
            #sobel_operator=padimage[i-
1,j+1]+2*padimage[i,j+1]+padimage[i+1,j+1]-padimage[i-1,j-1]-2*padimage[i,j-
1]-padimage[i+1,j-1]+padimage[i+1,j-1]+2*padimage[i+1,j]+padimage[i+1,j+1]-
padimage[i-1,j-1]-2*padimage[i-1,j]-padimage[i-1,j+1]
            #output_image[i-H,j-W]=sobel_operator+128
            #sobel_operator=padimage[i-
1,j+1]+2*padimage[i,j+1]+padimage[i+1,j+1]-padimage[i-1,j-1]-2*padimage[i,j-
1]-padimage[i+1,j-1]
            #output_image[i-H,j-W]=sobel_operator+128
            sobel_operator=padimage[i+1,j-1]+2*padimage[i,j-1]+padimage[i-1,j-
1]-padimage[i+1,j+1]-2*padimage[i,j+1]-padimage[i-1,j+1]
            output_image[i-H,j-W]=sobel_operator+128




    a=np.max(output_image)
    b=np.min(output_image)
    for i in range(0,H):
        for j in range(0,W):
            output_image[i,j]=int((output_image[i,j]-b)*255/(a-b))



    output_image=np.array(output_image,dtype=np.uint8)
    return output_image

def Sobel_frequency(input_image):
    H,W=input_image.shape
    output_image=np.zeros([H,W],dtype=np.int32)
    mask_H=3
    mask_W=3



    # pad input image and DFT
    P=H+mask_H-1
    Q=W+mask_W-1
    padimage=np.zeros([P,Q],dtype=np.int32)
    padimage[0:H,0:W]=input_image
    for i in range (0,P):
        for j in range (0,Q):
            padimage[i,j]=padimage[i,j]*((-1)**(i+j))

    padimage_DFT=np.fft.fft2(padimage)

    #pad the mask and DFT
    sobel_operator=[[-1,0,1],[-2,0,2],[-1,0,1]]

    pad_sobel_operator=np.zeros([P,Q],dtype=np.int32)
    pad_sobel_operator[int(1+(P-1-mask_H)/2):int(1+(P-1-
mask_H)/2+mask_H),int(1+(Q-1-mask_W)/2):int(1+(Q-1-
mask_W)/2+mask_W)]=sobel_operator

    for i in range (0,P):
        for j in range (0,Q):
            pad_sobel_operator[i,j]=pad_sobel_operator[i,j]*((-1)**(i+j))

    pad_sobel_operator_DFT=np.fft.fft2(pad_sobel_operator)
    pad_sobel_operator_DFT.real=0


    for u in range (0,pad_sobel_operator_DFT.shape[0]):
        for v in range (0,pad_sobel_operator_DFT.shape[1]):
            pad_sobel_operator_DFT[u,v]=pad_sobel_operator_DFT[u,v]*((-1)**
(u+v))


    #inverse
    product=padimage_DFT*pad_sobel_operator_DFT
    real_inverse_DFT=np.real(np.fft.ifft2(product))

    for i in range (0,real_inverse_DFT.shape[0]):
        for j in range (0,real_inverse_DFT.shape[1]):
            real_inverse_DFT[i,j]=real_inverse_DFT[i,j]*((-1)**(i+j))
    output_image=real_inverse_DFT[0:H,0:W]

    a=np.max(output_image)
    b=np.min(output_image)
    for i in range(0,H):
        for j in range(0,W):
            output_image[i,j]=int((output_image[i,j]-b)*255/(a-b))

    output_image=np.array(output_image,dtype=np.uint8)
    return output_image
```

result:

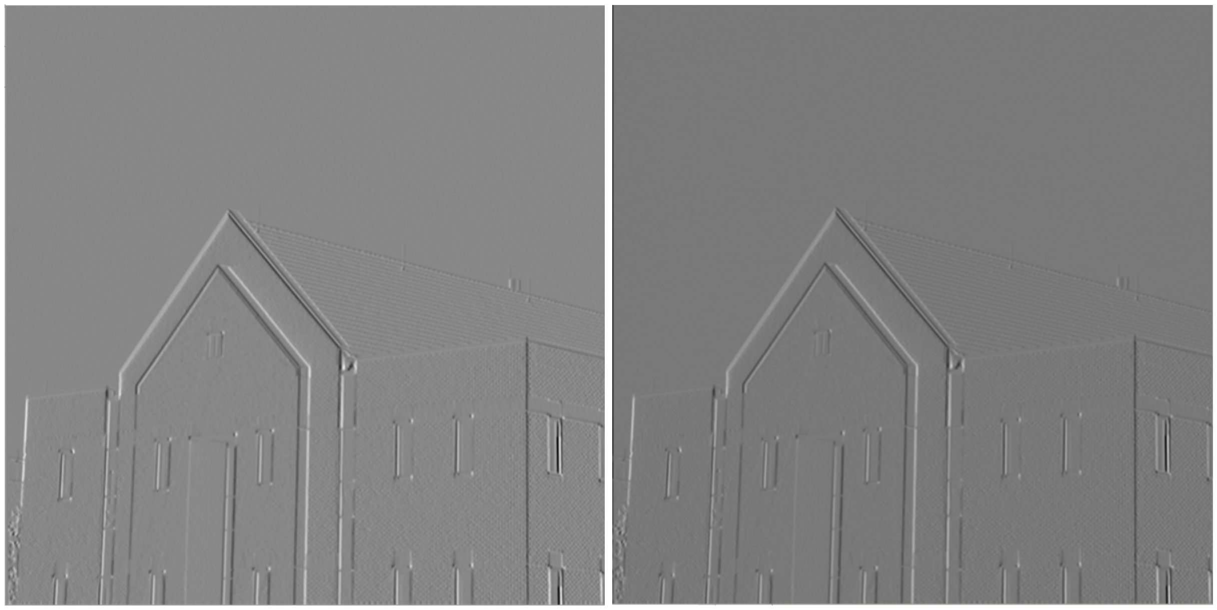


Fig.6 result of filtering in spatial domain(left) and result of filtering in frequency domain(right)

Analysis: From the above pictures, the sharpening effect of Sobel filter is apparent. The edge information of the object is better extracted. However, there is a little difference between them. The overall brightness of the image filtered in spatial domain is higher than the image filtered in frequency domain. I think two possible reasons behind this is the symmetric pad of the original image and Sobel operator added by 128 in frequency domain.

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

let $F(u,v)$ denote the spectrum of $h(x,y)$ and $F_p(u,v)$ denote the spectrum of $h_p(x,y)$, the relation between $h(x,y)$ and $h_p(x,y)$ and that between $F(u,v)$ and $F_p(u,v)$ is:

$$h_p(x,y) = h(x - \frac{M}{2}, y - \frac{N}{2}) \Longleftrightarrow F_p(u,v) = F(u,v)e^{-j\pi(u+v)} \tag{14}$$

Then

$$h_p(x,y)(-1)^{(x+y)} \Longleftrightarrow F_p(u,v) = F(u - \frac{M}{2}, v - \frac{N}{2})e^{-j\pi(u+v)} \tag{15}$$

The purpose of the shift in Step 4 is to make $F(u-\frac{M}{2}, v-\frac{N}{2})e^{-j\pi(u+v)}$ become $F(u-\frac{M}{2}, v-\frac{N}{2})$, which can eliminate frequency offset and ensure result we want at the top,left quadrant of IFFT of filtered result. What's more, this operation can keep the phase of image, which can store the profile information of the image.

$$e^{-j\pi(u+v)}(-1)^{u+v} = 1 \tag{16}$$

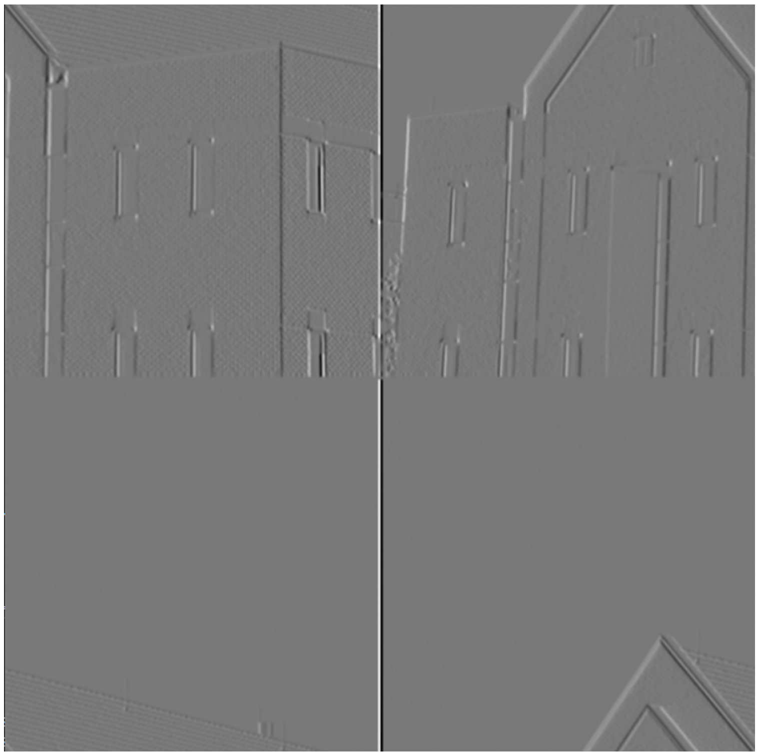


Fig.7 output without multiplying the result by $(-1)^{u+v}$

algorithm complexity:

	SPATIAL	FREQUENCY
average time/s	0.890	1.197

It is obvious that the average time used by spatial operation is less than that used by frequency domain. The main time in spatial domain is used in operation of Sobel operator, whose complexity of time is MNT_{add}.(T_{add} means the time one addition takes). The main time in frequency domain is used in multiply, FFT and IFFT, whose complexity of time is $(M+m)(N+n)T_{multiply} + (M+m)(N+n)log((M+m)(N+n))T_{multiply}$.($T_{multiply}$ means the time one multiply takes).

Other processing:

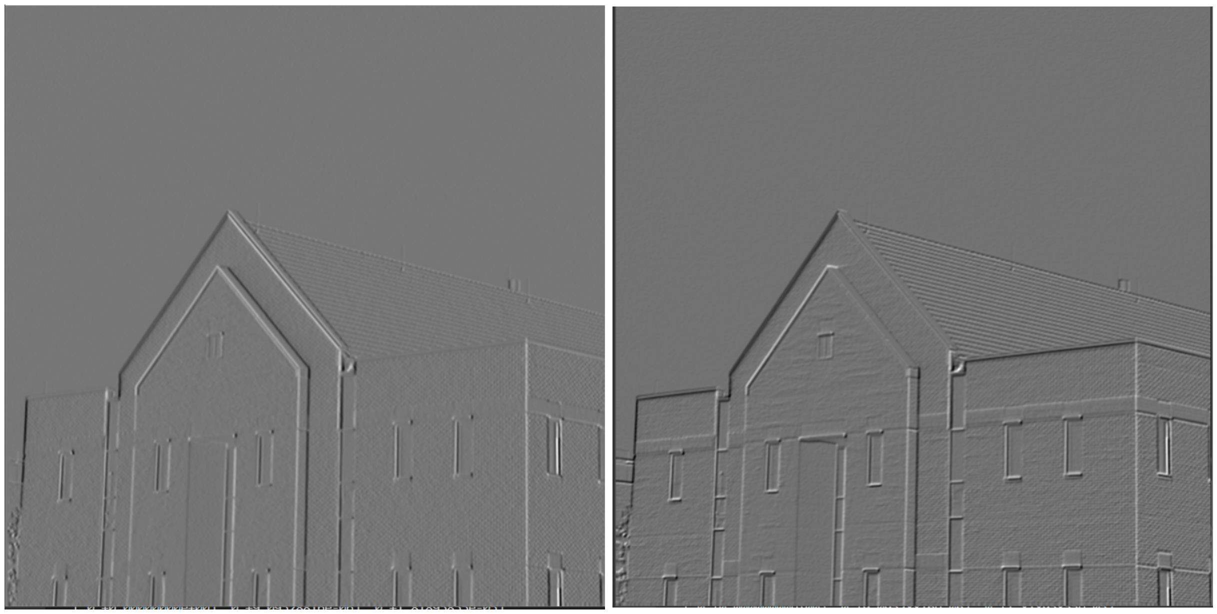


Fig.8 correlation using vertical Sobel operator in spatial domain(left) and correlation using horizontal and vertical Sobel operator in spatial domain(right)

Comparing the two pictures with the result filtering in frequency domain, it is obvious that he first two methods give better results. The result of correlation using horizontal and vertical Sobel operator in spatial domain is the best among the three because it makes the changes in horizontal direction and vertical direction more noticeable.

2.3 Butterworth notch reject filter

principle: Butterworth notch reject filter processes small regions of the frequency rectangle. To keep the resultant image with real intensity values, the zero-phase-shift must be symmetric about the origin. A notch with center at (u_0, v_0) must have a corresponding notch at location $(-u_0, -v_0)$. Notch reject filters are constructed as products of high-pass filter whose center have been translated to the centers of the notches.

Question formulation: According to the principle, the formula of notch reject filters is：

$$H_{NR}=\prod_{k=1}^{Q}H_k(u,v)H_{-k}(u,v) \tag{17}$$

where $H_k(u,v)$ and $H_{-k}(u,v)$ are high-pass filters whose centers are at (u_k,v_k) and $(-u_k,-v_k)$,respectively.

A Butterworth notch reject filter of order n is:

$$H_{NR}=\prod_{k=1}^{K}[\frac{1}{1+[\frac{D_{0k}}{D_k(u,v)}]^{2n}}][\frac{1}{1+[\frac{D_{0k}}{D_{-k}(u,v)}]^{2n}}] \tag{18}$$

Where

$$\begin{aligned}D_k(u,v)=[(u-M/2-u_k)^2+(v-N/2-v_k)^2]^{\frac{1}{2}}\\ D_{-k}(u,v)=[(u-M/2+u_k)^2+(v-N/2+v_k)^2]^{\frac{1}{2}}\end{aligned} \tag{19}$$

So, an important thing is to get the center coordinates of $H_k(u,v)$, which can be realized by `plt.imshow`.

pseudo code:

```
padimage=np.zeros([2*input_image.shape[0],2*input_image.shape[1])
padimage[0:input_image.shape[0],0:input_image.shape[1]]=input_image
padimage=padimage*((-1)**(x+y))
padimage_DFT=FFT(padimage)

click the point in the spectrum of padimage to get the center coordinats

Butterworth_notch_reject_filter=np.ones([2*input_image.shape[0],2*input_image.
shape[1]]
Butterworth_notch_reject_filter=construct(coordinate,n,D0)

real_inverse_DFT=real(IFFT(padimage_DFT*Butterworth_notch_reject_filter))
((-1)**(x+y))

output_image=real_inverse_DFT[0:input_image.shape[0],0:input_image.shape[1]]

output_image=clip(output_image,0,255)
```

python code:

```
def DFT_input_image(input_image):
    H,W=input_image.shape

    #pad the input image
    P=2*H
    Q=2*W
    padimage=np.zeros([P,Q],dtype=np.float32)
    padimage[0:H,0:W]=input_image

    for i in range(0,P):
        for j in range(0,Q):
            padimage[i,j]=padimage[i,j]*((-1)**(i+j))

    padimage_DFT=np.fft.fft2(padimage)



    return padimage_DFT
    #return np.log(np.abs(padimage_DFT))



def Butterworth_notch_filter(input_image,coordinate,n,D0):
    H,W=input_image.shape

    P=2*H
    Q=2*W
    #Butterworth notch reject filter
    Butterworth_notch_reject_filter=np.ones([P,Q],dtype=np.float32)
    for i in range(0,P):
        for j in range(0,Q):
            for k in range(0,len(coordinate[0])):
                #D=np.sqrt((i-P/2-coordinate[0,k])**2+(j-Q/2-
coordinate[1,k])**2)
                #D_inverse=np.sqrt((i-P/2+coordinate[0,k])**2+(j-
Q/2+coordinate[1,k])**2)
```

```
                #D=np.sqrt((i-P/2-(coordinate[0,k]-P/2))**2+(j-Q/2-
(coordinate[1,k]-Q/2))**2)
                #D_inverse=np.sqrt((i-P/2+(coordinate[0,k]-P/2))**2+(j-Q/2+
(coordinate[1,k]-Q/2))**2)

                D=np.sqrt((i-coordinate[0,k])**2+(j-coordinate[1,k])**2)
                D_inverse=np.sqrt((i-(P/2-(coordinate[0,k]-P/2)))**2+(j-(Q/2-
(coordinate[1,k]-Q/2)))**2)
                if D!=0 and D_inverse!=0:

 Butterworth_notch_reject_filter[i,j]=Butterworth_notch_reject_filter[i,j]*
(1/(1+(D0/D)**(2*n)))*(1/(1+(D0/D_inverse)**(2*n)))
                else:
                    Butterworth_notch_reject_filter[i,j]=0

    return Butterworth_notch_reject_filter

def obtain(DFT_input,DFT_filter,input_image):
    H,W=input_image.shape
    output_image=np.zeros([H,W],dtype=np.float32)
    product=DFT_input*DFT_filter
    real_inverse_DFT_lowpass=np.real(np.fft.ifft2(product))

    for i in range (0,real_inverse_DFT_lowpass.shape[0]):
        for j in range (0,real_inverse_DFT_lowpass.shape[1]):
            real_inverse_DFT_lowpass[i,j]=real_inverse_DFT_lowpass[i,j]*
((-1)**(i+j))


    output_image=real_inverse_DFT_lowpass[0:H,0:W]
    output_image=np.clip(output_image,0,255)
    output_image=output_image.astype(np.uint8)

    return output_image
```

result:

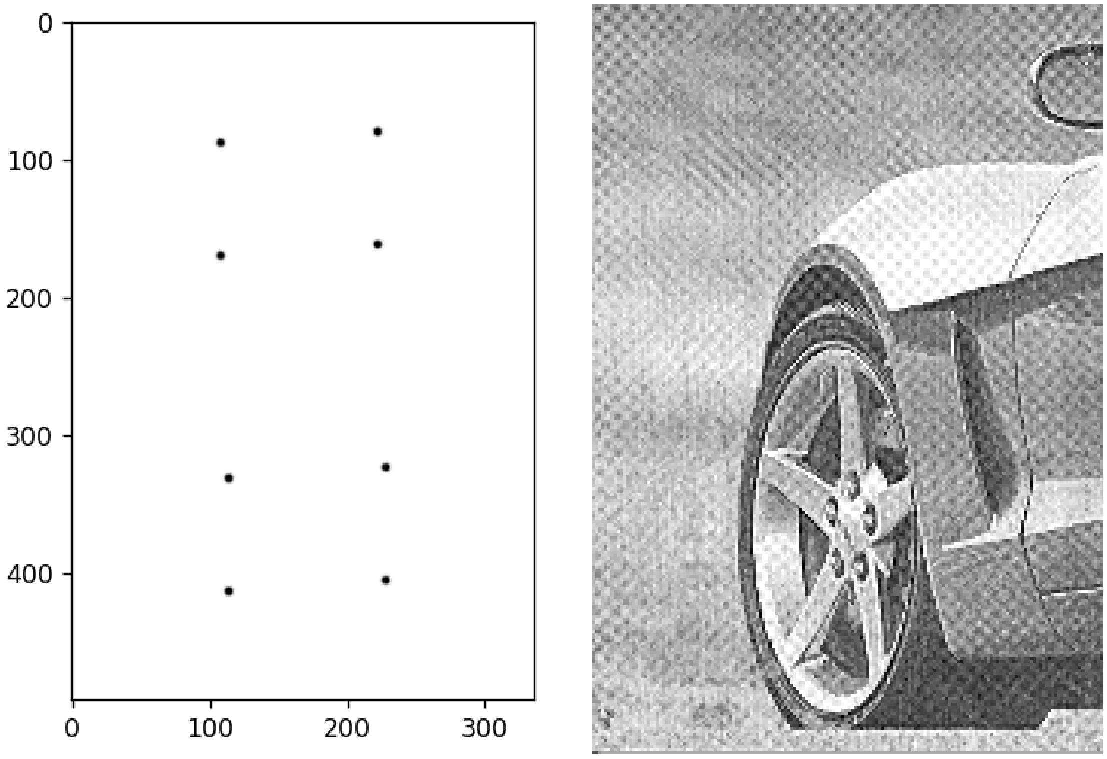


Fig.9 D0=3 n=4

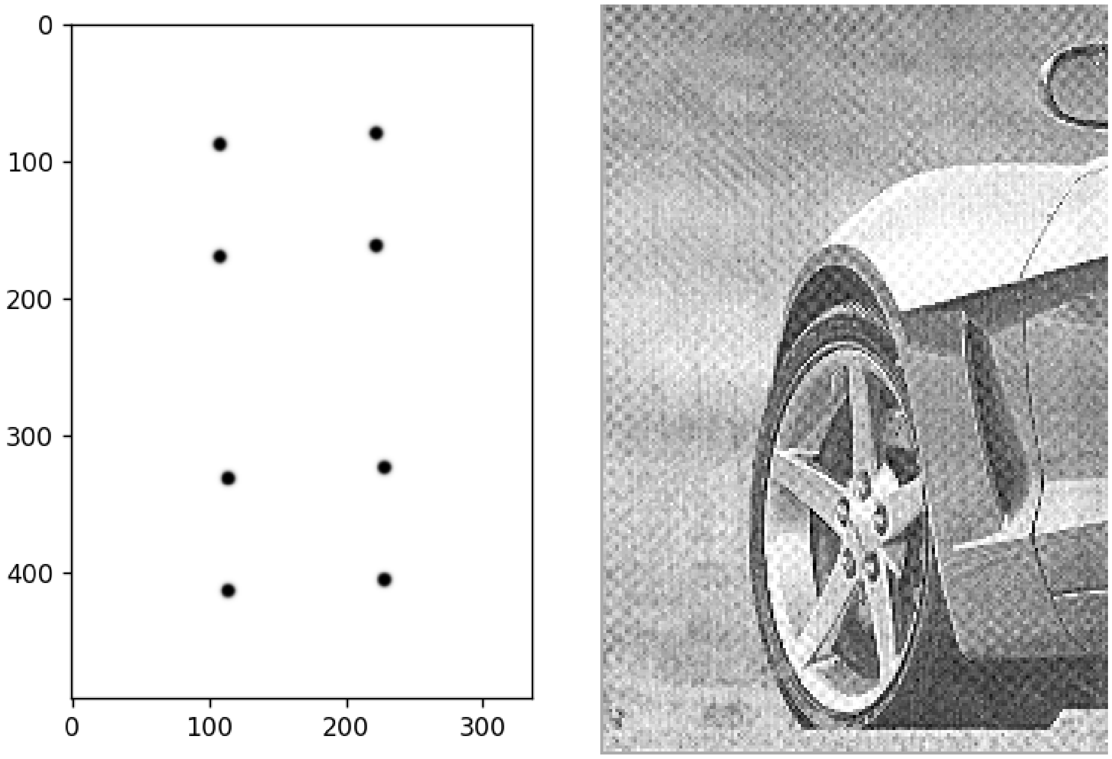


Fig.10 D0=5 n=4

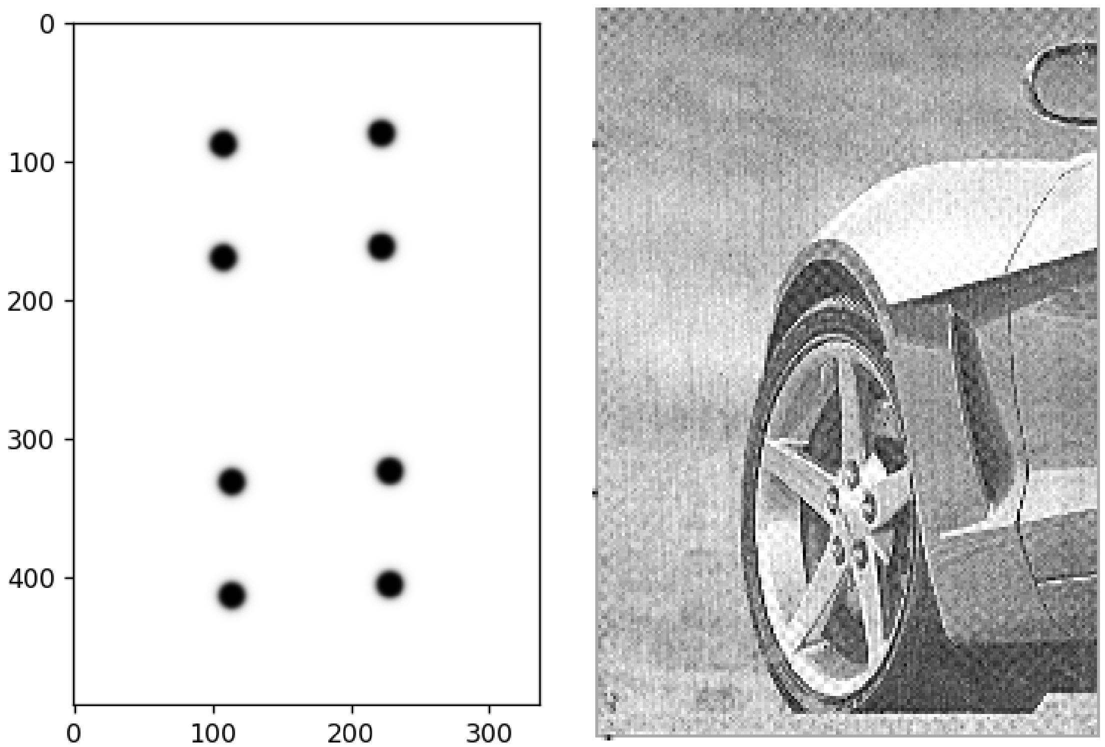


Fig.11 D0=10 n = 4

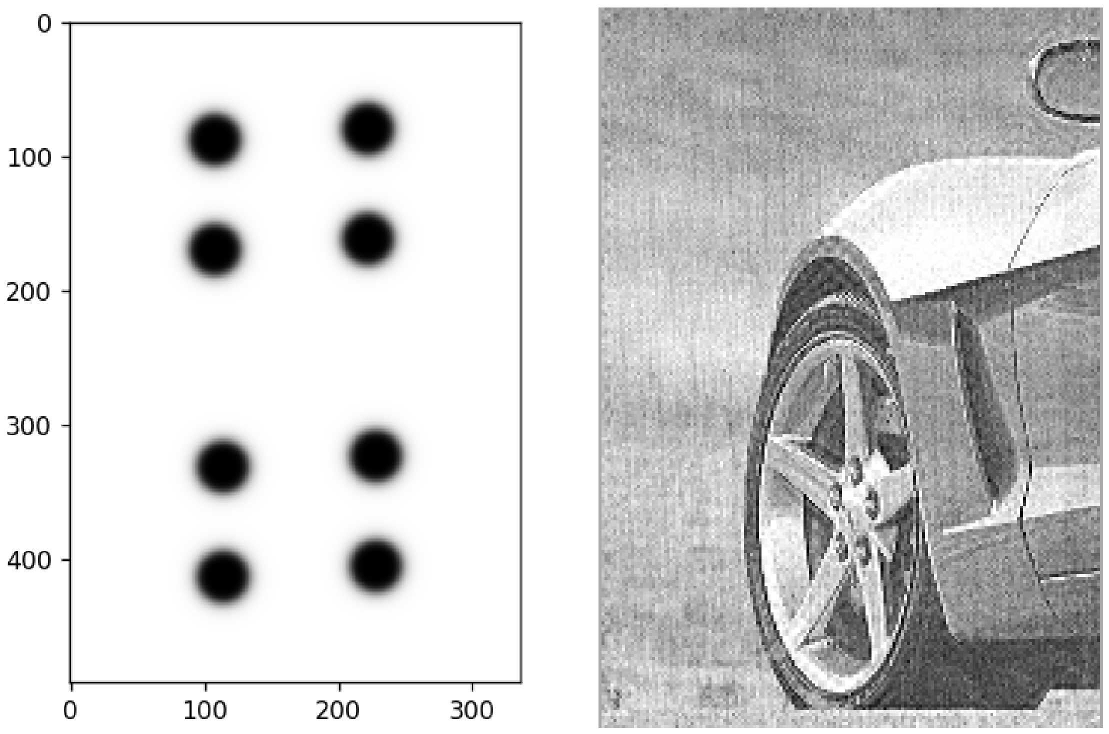


Fig.12 D0=20 n=4

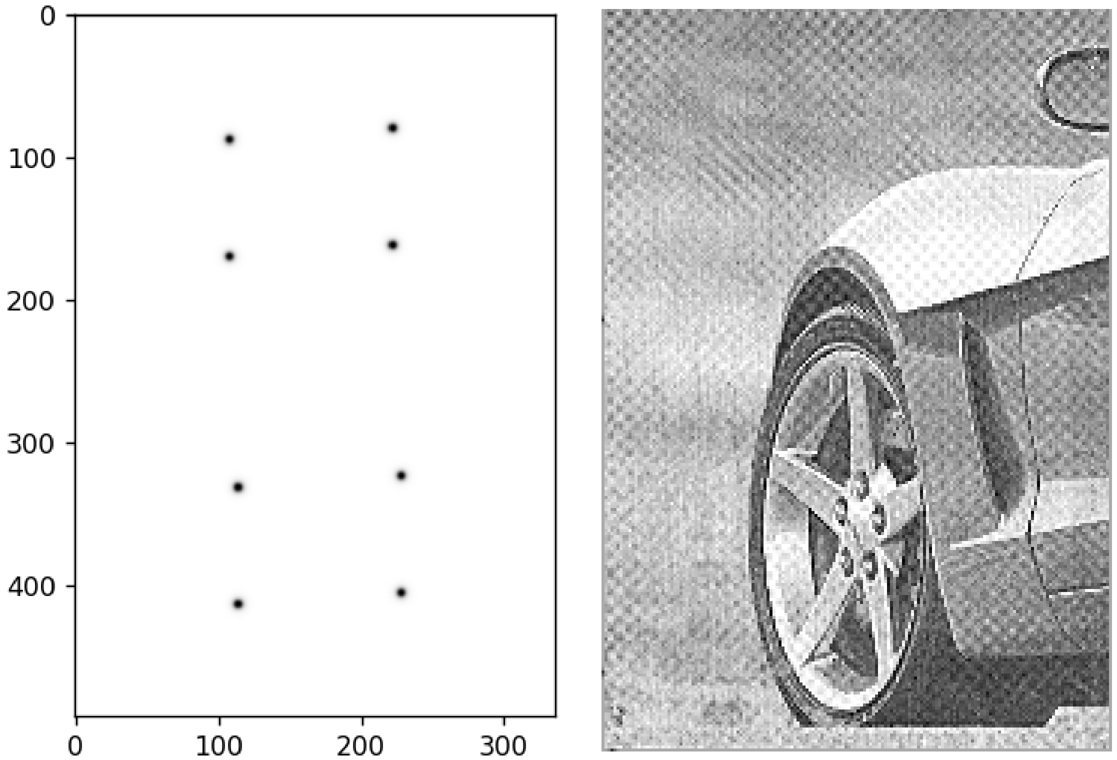


Fig.13 D0=3 n=2

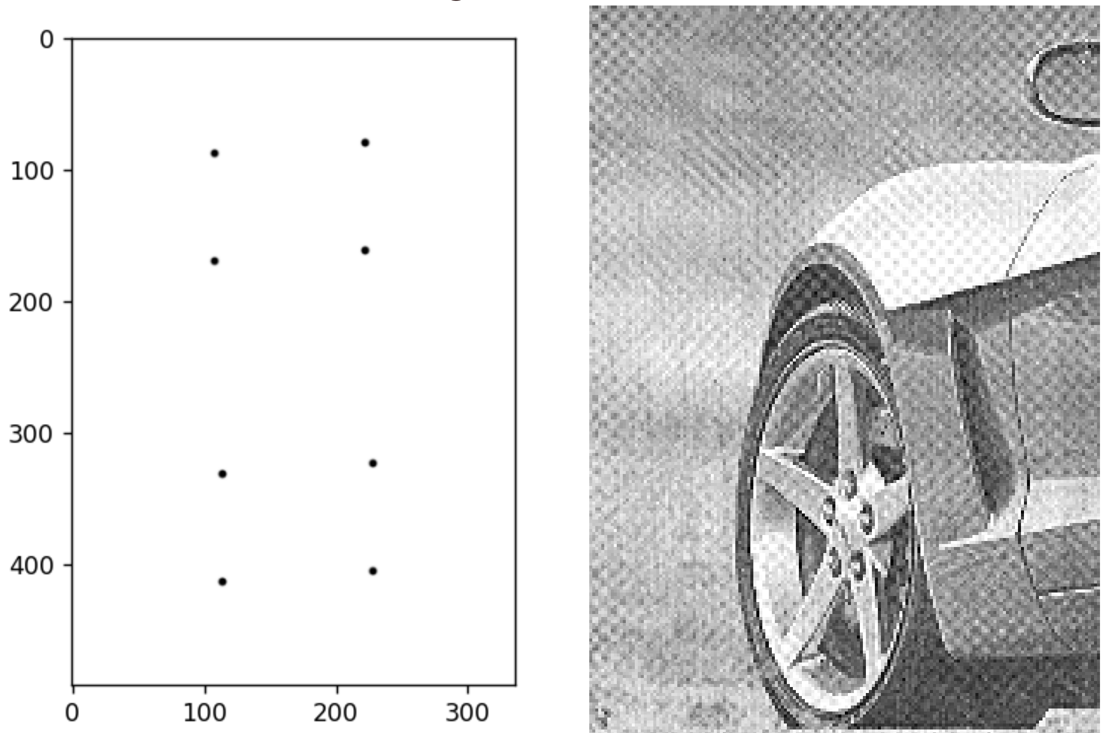
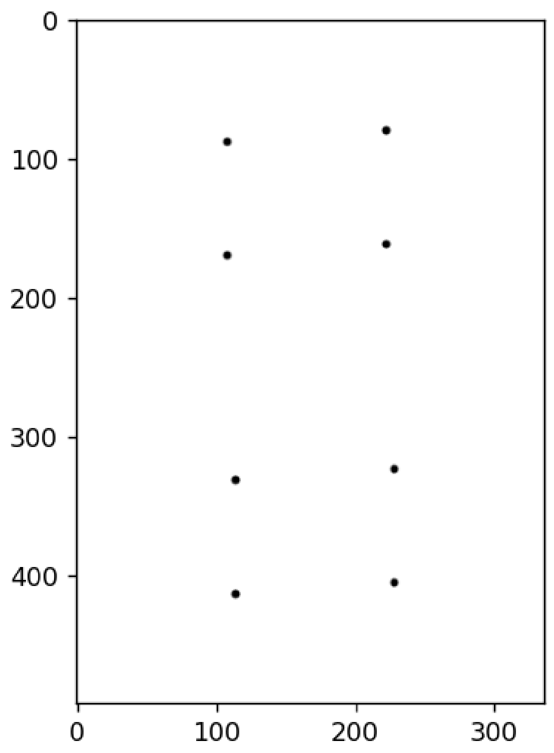


Fig.14 D0=3 n=6



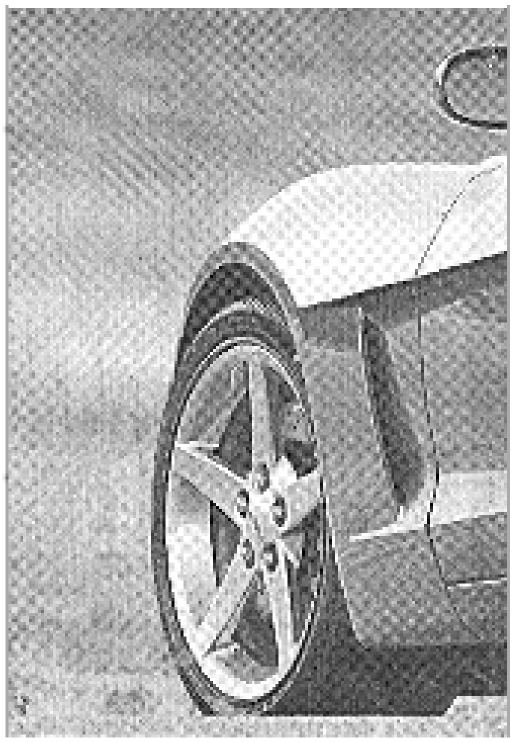
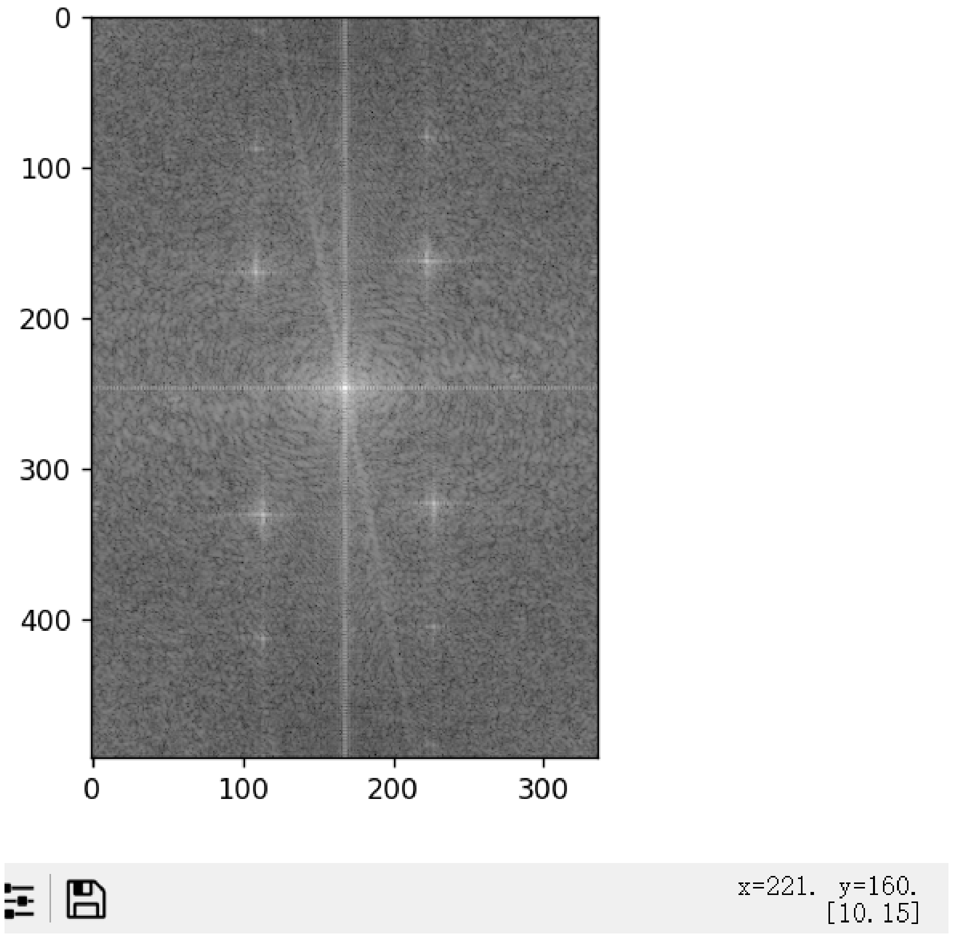


Fig.15 D0=3 n=8

Analysis: From the above pictures, what can be known is that with $D0$ increasing in a certain range, Butterworth notch reject filter works better. However, the effect it attenuates components other than noise is stronger according to `Formula(18)`. With n increasing, the fuzzy component around the notch point is decreasing. According to `Formula(18)`, with n increasing, the closer points is to notch point, the faster the attenuation will be.

how the parameters in the notch filters are selected, and why:

To determine the value of K and the centers of high-pass filter, the function `plt.imshow` is used to the log spectrum. The value of K is equal to half the number of noise points in the log spectrum we need to remove. And click the noise points in the log spectrum to get the coordinate of it, which can be used in `Formula(19)`.



As for the value of n and $D0$, they are determined by the size of the noise points in the log spectrum and the strength of difference between the noise points and the points in the neighborhood because n and $D0$ have a great effect on the performance of removing the noise.

algorithm complexity:

ORDER	2	4	6	8
average time/s	6.691	6.767	6.827	6.938

It is obvious that with n increasing, the average time is increasing. The main time is used in in multiply, FFT and IFFT, whose complexity of time is $nM^2T_{multiply} + M^2logM^2T_{multiply}$.($T_{multiply}$ means the time one multiply takes).

2.4 Explain why H(μ, ν) 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

The principle of filtering in frequency domain is that the spectrum of the image is multiplied by the spectrum of the filter and do IFFT to obtain the processed image. The DFT of an image can be expressed as :

$$F(u,v) = R_F(u,v) + jI_F(u,v) \tag{20}$$

The filtered expression in spatial domain is:

$$g(x,y) = F^{-1}[H(u,v)R(u,v) + jH(u,v)I(u,v)] \tag{21}$$

The the phase of the image stores the profile information of the image. Besides., $F(u,v)$ is periodic. These profile information can be kept when $H(u,v)$ is real and symmetric.

However, there is an exception, which is the $H(u,v)$ of the Sobel mask used in `Section 2.2`. The Sobel mask exhibits odd symmetry, provided that it is embedded in an array of zeros of even sizes. The odd symmetry is preserved with respect to the padded array in forming $h_p(x,y)$, $H_p(u,v)$ of $h_p(x,y)$ is purely imaginary and odd according to the symmetry properties of the 2-D DFT, which yields results that are identical to filtering the image spatially using $h(x,y)$. To keep the results identical, $H_p(u,v)$ has to be purely imaginary. Because of that, we need to set the real part of the result DFT of $h_p(x,y)(-1)^{x+y}$ to 0.

3. Conclusion

From the analysis about the two methods of filtering in frequency domain, The two methods have their own characteristics.

Sobel operator is a discrete differential operator, which is used to calculate the approximate gradient of image gray function. That is, Sobel operator can be used to measure the change of image in vertical and horizontal directions. Due to the discrete characteristics of pixels, Sobel operator provides the approximation of image gradient through pixel difference in horizontal and vertical directions, which is used in the edge detection of an image.

Butterworth notch reject filter processes small regions of the frequency rectangle, which can be used to remove periodic noise. The Butterworth notch reject filter mainly attenuates a certain point and does not have an effect to other components.

What's more, to avoid the data from adjacent periods produce wraparound error, yielding an incorrect convolution result, function padding must be used.