

# Lab 4 Frequency Domain Filtering

Student name & ID: YUAN Tong 11810818

---

Course: *LAB Session I* – Professor: *YU Yajun*

Date: *March 30, 2021*

## Introduction

A thorough understanding of image filtering is impossible without having at least a working knowledge of how the Fourier transform and the frequency domain can be used for image filtering. Briefly, the frequency domain is a space which is defined by **Fourier Transform** – a very wide application in image processing, **frequency domain analysis** is used to indicate how signal energy can be distributed in a range of frequency. By filtering in frequency domain, we can manipulate signal with specific frequency contained in the image.

In this LAB, we will first introduce the procedure by which we used to apply Discrete Fourier Transform to image and different kinds of operator, then we will apply **sobel**, **Gaussian** and **Butterworth** notch filters to different image and analysis their performance.

Generally, this series of methods can be used to extract specific information contained in the image, can sharp or unshape the image, pre-process the image for neural network training, and also can add special effect to the image. As the most widely used technique for image processing, Fourier Transform still benefits many areas.

## Fourier Transform and Filtering in Frequency Domain

### Fourier Transform Basic.

In 1D case, we demonstrate that all convergent continuous signal can be written in the form of

$$f(t) = \sum_{n=-\infty}^{+\infty}$$

with

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{j\frac{2\pi n}{T}t} dt$$

and we define fourier transform as

$$F(\mu) = \Im\{f(t)\} = \int_{-\infty}^{\infty} f(t) e^{j2\pi\mu t} dt$$

One important property of fourier transform is that the Fourier transform of the convolution of two functions in the spatial domain is equal to the product in the frequency domain of the Fourier transforms of the two functions, and can be expressed as

$$f(t) \star h(t) = H(\mu)F(\mu)$$

or

$$f(t)h(t) = H(\mu) \star F(\mu)$$

which will be the basic idea in Frequency domain filtering.

For arbitrary discrete signal, we have

$$F(u) = \sum_{n=0}^{M-1} f_n e^{-j2\pi un/M}$$

and

$$f(x) = \frac{1}{M} \sum_{u=0}^{M-1} F(u) e^{j2\pi ux/M}$$

with property

$$F(u) = F(u + kM)$$

### Fourier Transform For Image.

To apply Fourier transform to image, firstly we need to expand the Fourier Transform to 2D form. The 2D fourier transform can be expressed as

$$F(\mu, \nu) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(t, z) e^{-j2\pi(\mu t + \nu z)} dt dz$$

and

$$f(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F(\mu, \nu) e^{j2\pi(\mu t + \nu z)} d\mu d\nu$$

And in discrete form we have

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

and

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

## Image Filtering in Frequency domain.

**Property.** Several important properties of Fourier Transform will be introduced here so we can design the procedure of the image processing steps.

**Firstly**, is the relationship that

$$f(t) \star h(t) = H(\mu)F(\mu)$$

or

$$f(t)h(t) = H(\mu) \star F(\mu)$$

which we have introduced before, according to this property we can first do the Fourier Transform to both image and the operator, multiple them together, and than do the reverse Fourier Transform to the product, the result is the image filtered in frequency domain.

**Secondly** is about the periodicity, we have

$$f(x)e^{j2\pi(u_0x/M)} \Leftrightarrow F(u - u_0)$$

we assign  $u_0 = \frac{M}{2}$  to the formula and we obtain

$$f(x)(-1)^x \Leftrightarrow F(u - M/2)$$

and for 2D we have

$$f(x, y)(-1)^{x+y} \Leftrightarrow F(u - M/2, v - N/2)$$

the result of such transform are shown in Figure 1. The periodicities of the transform and its inverse are important issues in the implementation of DFT-based algorithms. the transform data in the interval from 0 to  $M - 1$  consists of two back-to-back half periods meeting at point  $M/2$ . Shift the zero-frequency component to the center of the spectrum **For display and filtering purposes**, it is more convenient to **have in this interval a complete period of the transform** in which the data are contiguous. If we do not perform the shift operation, we will get the result shown in Figure 2b

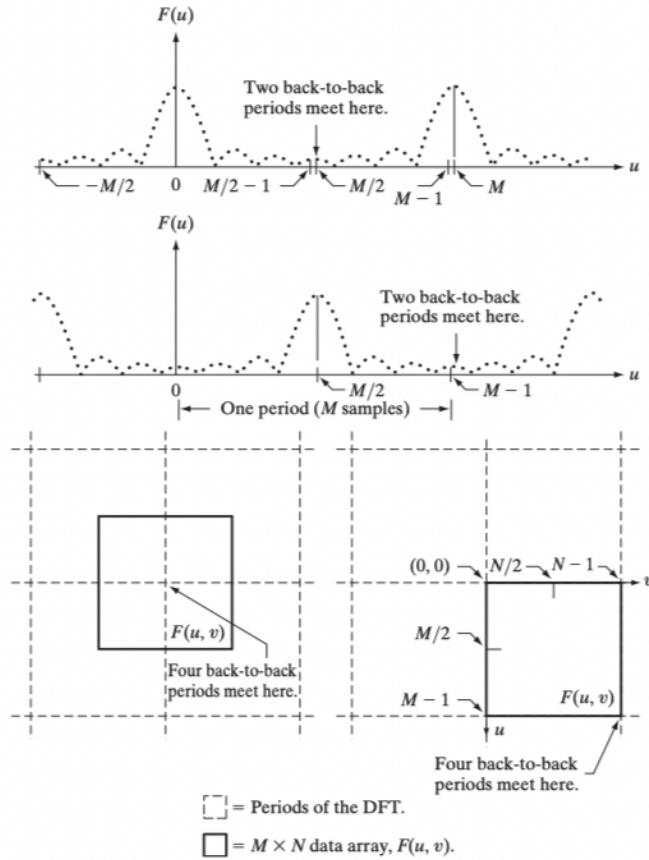


Figure 1: Shifting of the signal

Thirdly, and most importantly, is about the Symmetry Properties, we define discrete function that satisfy  $f(x) = f(N - x)$  even function and those satisfy  $f(x) = -f(N - x)$  odd function.

**Zero padding.** Before we apply the Fourier Transform to the iamge, we will first pad the image as shown in Figure 2c. Because we are dealing here with discrete quantities, computation of the Fourier transforms is carried out with a DFT algorithm. If we elect to compute the spatial convolution using the IDFT of the product of the two transforms, then the periodicity issues must be taken into account. when convolving two periodic functions, the convolution itself is periodic, so we may meet wraparound error as shown in Figure 2d, so to avoid this we must pad the image with the

$$P > A + C - 1$$

and

$$Q > B + D - 1$$

with

$$P > 2M - 1$$

and

$$Q > 2N - 1$$

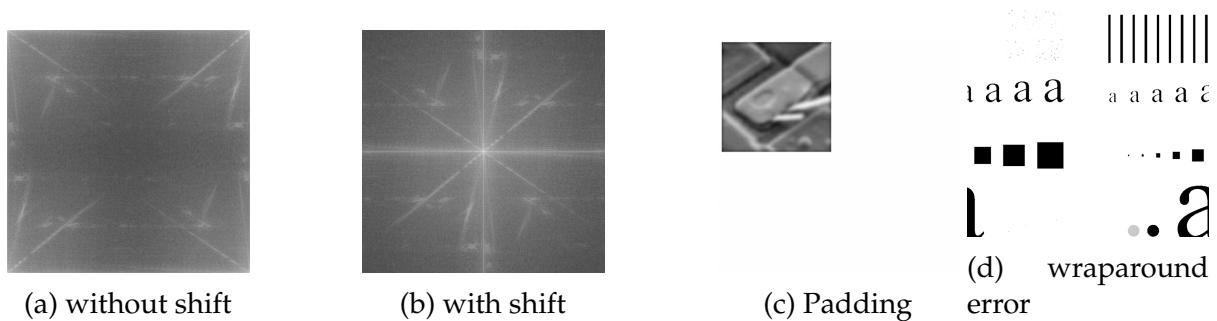


Figure 2: Why we need padding and shift

**Procedure.** Here we summarize the steps to filtering in 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  $f_p(x, y)$  by  $(-1)^{x+y}$  to center its transform note: as mentioned in property 2, centering helps in visualizing the filtering process and in generating the filter functions themselves, but centering is not a fundamental requirement.
4. Compute the DFT,  $F(u, v)$  of the image from 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

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

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

### Task 1: Sobel filter

Implement the Sobel filter to the input images Q5\_1.tif in both spatial domain and frequency domain. Compare the results. Refer to slides 78 to 81 of Lecture 4.



Figure 3: Q5\_1.tif

**Analysis.** The Sobel operator is used in image processing and computer vision, particularly within **edge detection** algorithms where it creates an image emphasising edges. The application of Sobel operator can be written as

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A$$

and

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

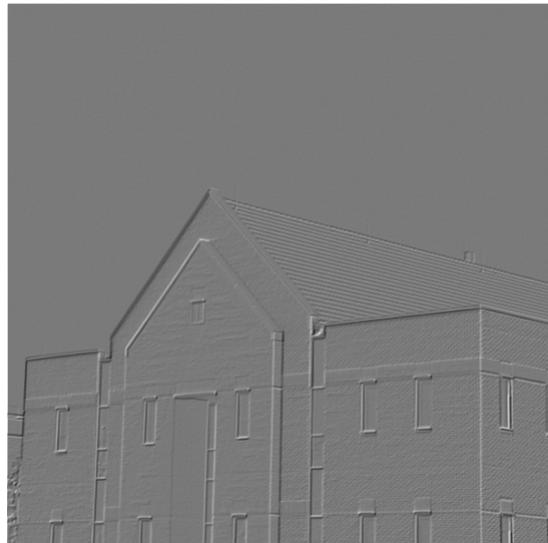
In this task we will apply both vertical and horizontal filter to the given image to detect the edges contained in the image on both spatial domain and frequency domain.

And to apply a spatial filter to frequency domain, we need to do the steps 2 to 4 in the basic steps of image processing in frequency domain, then multiply them together, finally we do IDFT to the product and we can get the result.

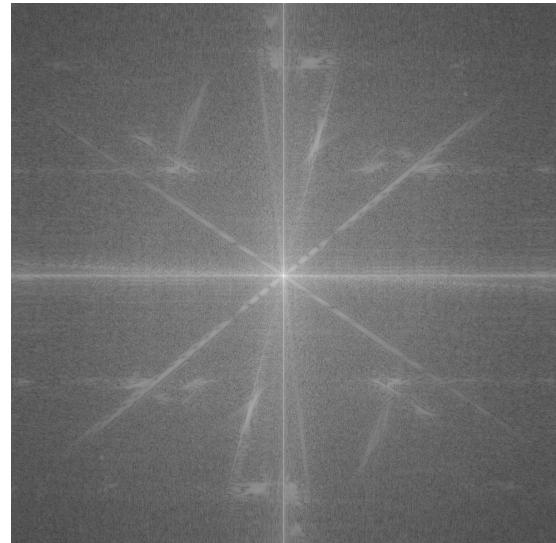
### Result.

In this task we first apply the sobel filter to the image in spatial domain, and we get the result shown in Figure 4a.

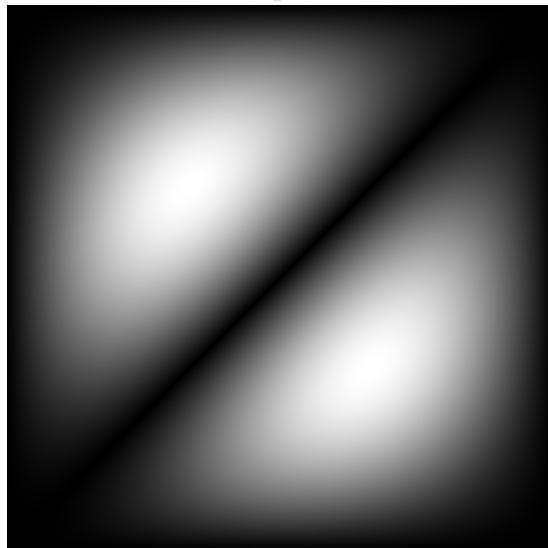
Then we apply the sobel operator on frequency domain, first convert the image by shift and DFT to frequency domain as shown in Figure 4b, second, we apply same procedure to the sobel filter after padding it and we obtain Figure 4c, finally we multiply them together and get Figure 4d, compare Figure 4a and Figure 4d, we found that they are almost the same, which means here filter in spatial domain and filter in frequency domain are equal.



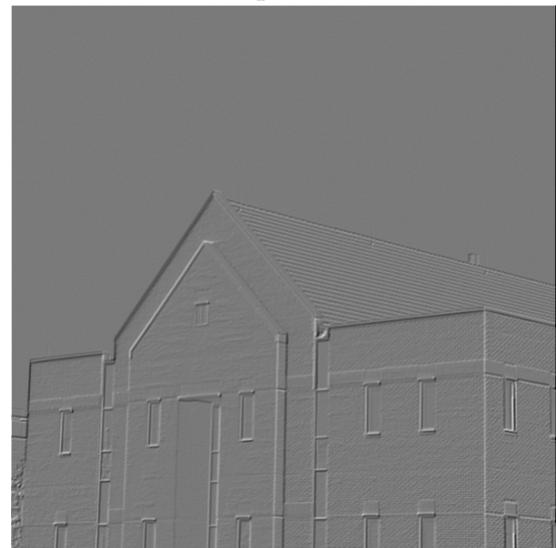
(a) spatial



(b) spectrum



(c) Filter



(d) frquency

Figure 4: Image filtered by sobel filter in spatial domain

**Discussion.** In this task we apply a shift at the step 4, and the reason has been discussed at last session.

And there is another rules that in both procedure, if you filp the sobel operator we can found the output image also "flipped", the "direction" of the gridant in both image will change.

### Task 2: Gaussian low pass and high pass

Implement the Gaussian low pass and high pass to the input image Q5\_2.tif. results for  $D_0$  0 = 30 , 60 , and 160, respectively.

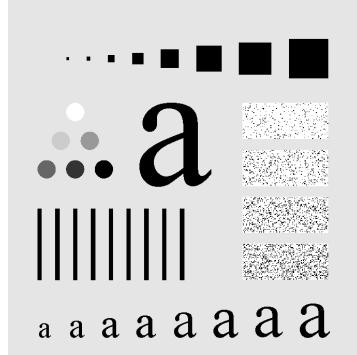


Figure 5: Q5\_2.tif

#### Analysis.

In this task, our goal is to reduce the signal intensity contained in high or low frequency domain by Gaussian filter. The Gaussian filter can be described as

$$H(u, v) = e^{-D^2(u,v)/2\sigma^2}$$

as shown in Figure 6, bigger n is, less signal in high frequency will be filtered.

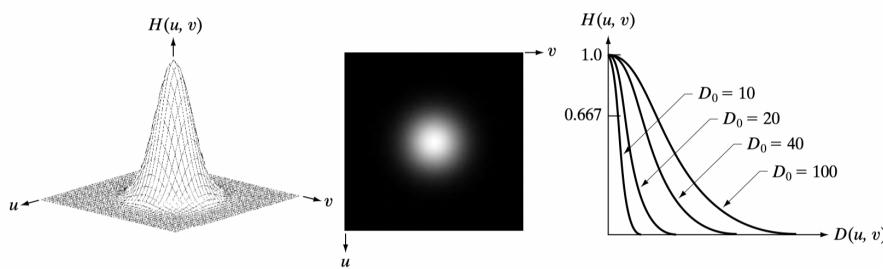


Figure 6: Gaussian lowpass filter

#### Result.

In this task, we apply Gaussian highpass and lowpass filter with  $\sigma$  equal to 30, 60, 160, the Gaussian lowpass filters and the filtered images are shown in Figure 7

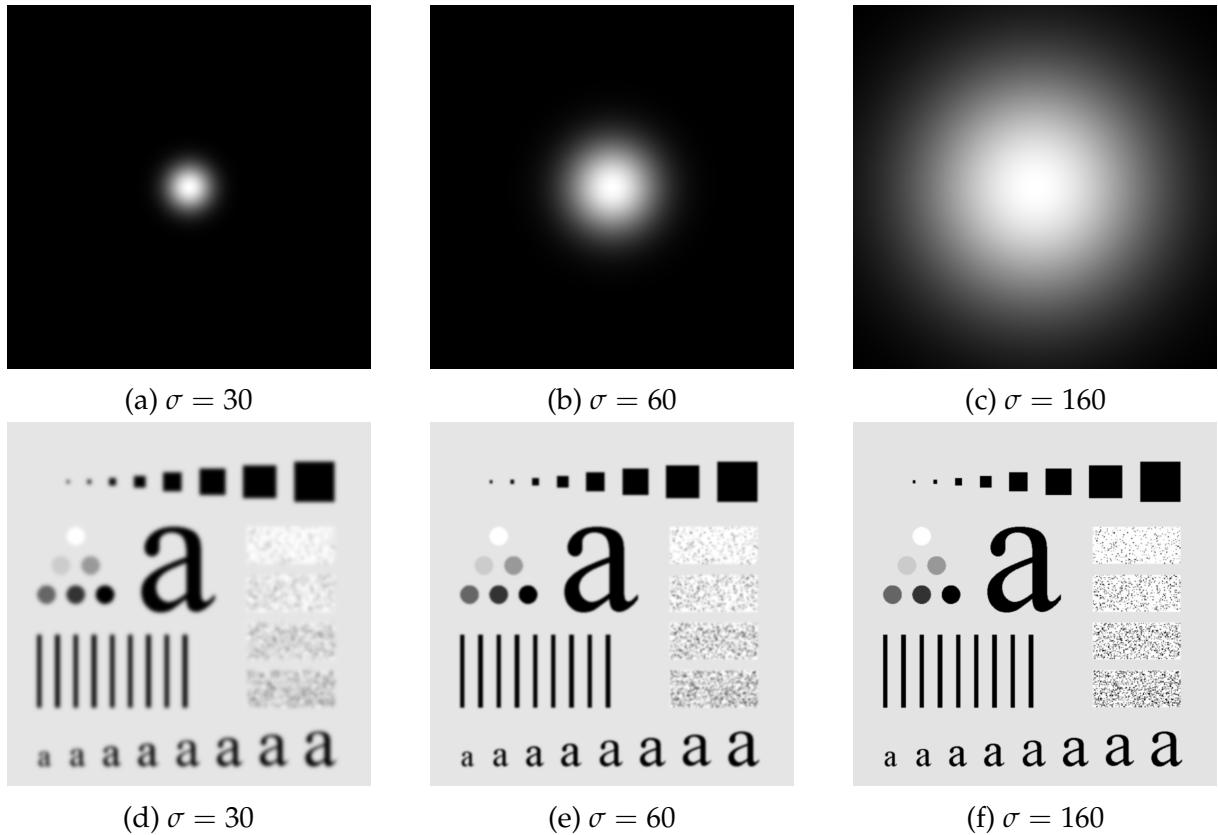


Figure 7: The Gaussian low filter and output images

Here we can found that, as  $\sigma$  increase, the image become clearer, which means that more information in high frequency domain is reserved, and the image is sharper.

Then we convert Gaussian lowpass filter to Gaussian highpass filter via the formula  $\max(H) - H$ , and we apply the filter to the image to the frequency domain.

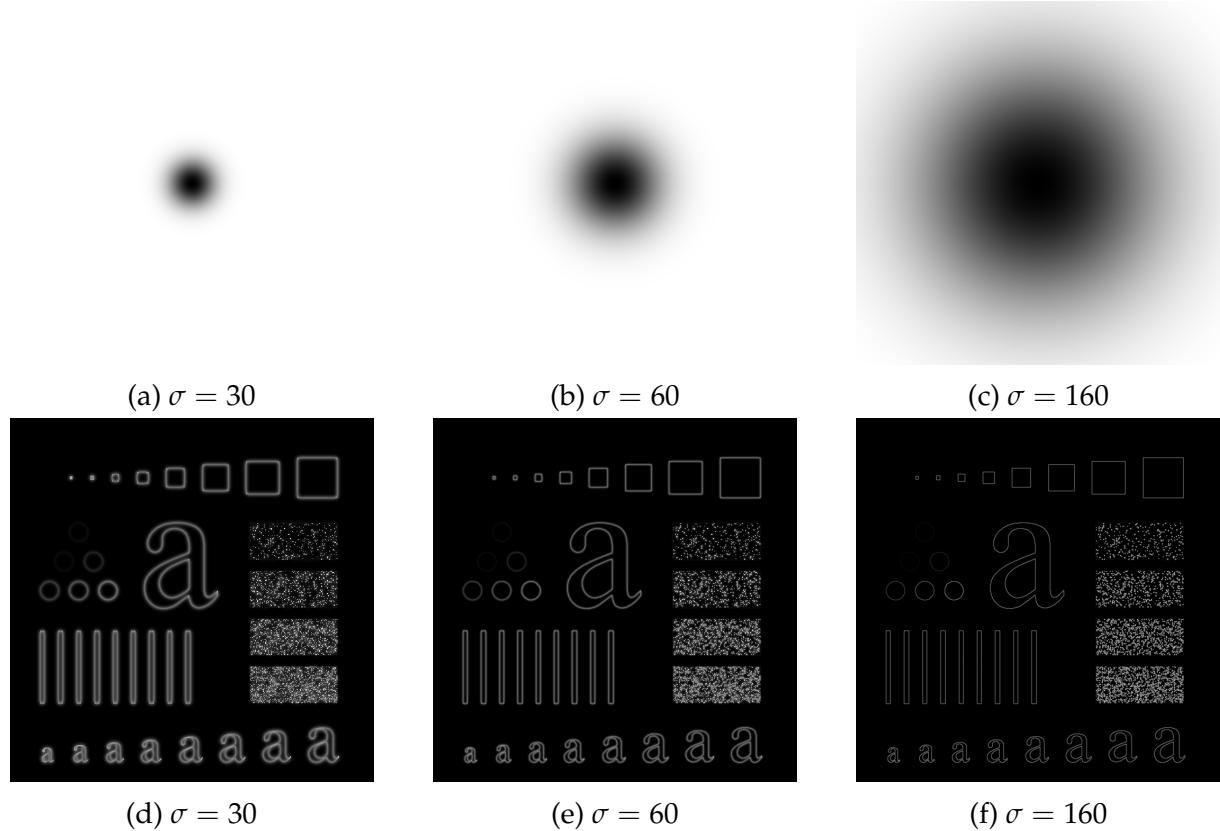


Figure 8: The Gaussian highpass filter and output images

With the increase of  $\sigma$  more features in low frequency will be filtered, and finally only edge can be reserved.

### Discussion.

Interestingly, we found that while the addition of highpass filter and lowpass filter with same  $\sigma$  produce an all-white image, the addition of the filtered image is equal to the original image. When  $\sigma = 1$ , almost no feature remained after filtered by lowpass filter, which means no feature contained in the extreme low frequency, with more information in relatively high frequency remained, almost all the features remained, for highpass filter, only edge remained after most of the information in low frequency is removed. So we can draw a conclusion that, most of the information, except the edge, contained in low frequency domain.

We can also conclude the application of different kind of Gaussian filters. For Gaussian lowpass filter, with lower  $\sigma$  we can remove the details in the image. For Gaussian highpass filter, with higher  $\sigma$  we can remove most of the pixels except the edge.

### Task 3: Butterworth notch filters

Implement the Butterworth notch filters to the input images Q5\_3.tif. Refer to slides 110 to 114 of Lecture 4.



Figure 9: Q5\_3.tif

**Analysis.** In this task, our goal is quite similar to that in Task 2, we want to reduce signal with specific frequency to , the Butterworth notch filters can be described as

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

as shown in Figure 10 bigger n is, more signal in high frequency will be filtered, and compared with Gaussian filter, we found that with bigger n, wider range of signal in low frequency will be preserved.

Different from Gaussian filter, two parameter in butterworth filter can be adjusted,  $\sigma$ , the cutoff frequency and the  $n$ , the order, with bigger n, as shown in Figure 10, the information in high frequency domain will be absolutely filtered, with smaller n, some information in high frequency domain will be reserved and some information in low frequency will be filtered.

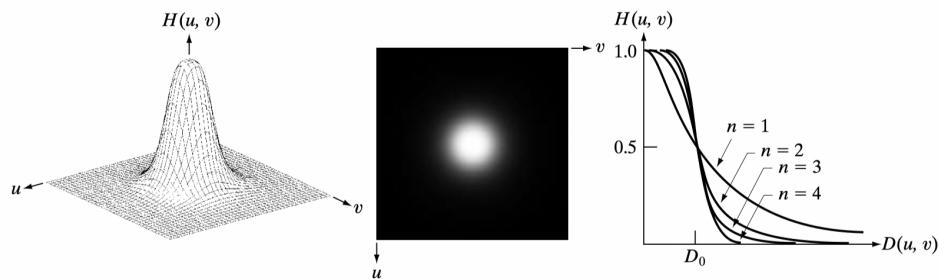


Figure 10: Butterworth lowpass filter

For the given image we first apply the DFT to it, and we found that the noise is contained in 8 points, and our task is to remove them. We generate 8 Butterworth

highpass filter, and add them together to obtain the filter shown in Figure 12, then we apply it to the spectrum of the spectrum and get Figure 11c, then we will compare which parameters are better.

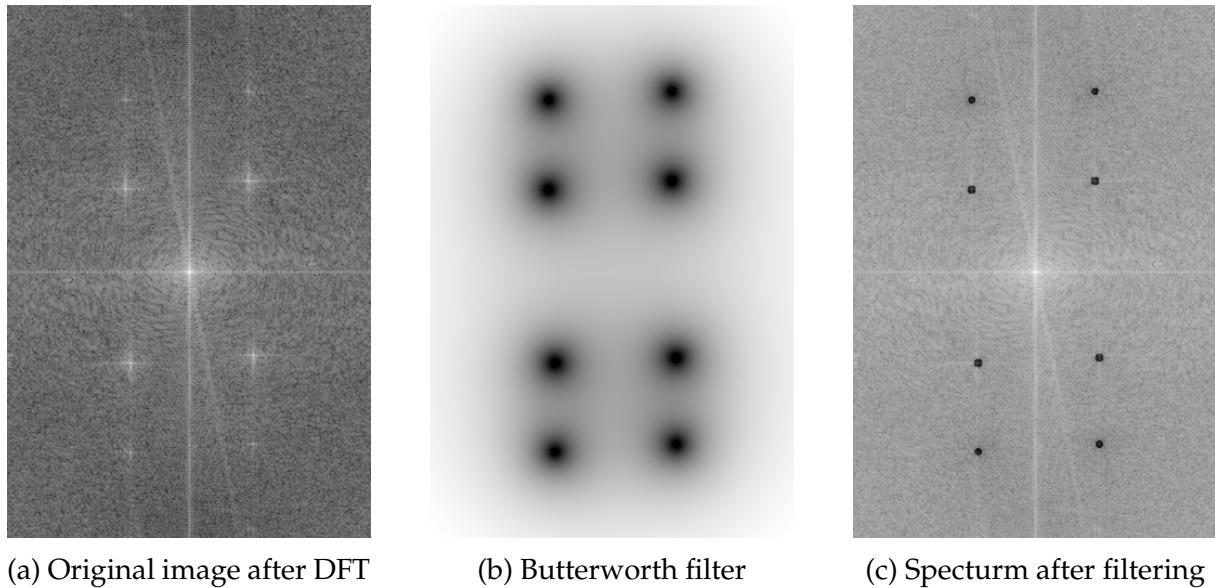


Figure 11: Butterworth filter

### Result.

Here we apply Butterworth filter with different  $n$  and  $\sigma$  to image.

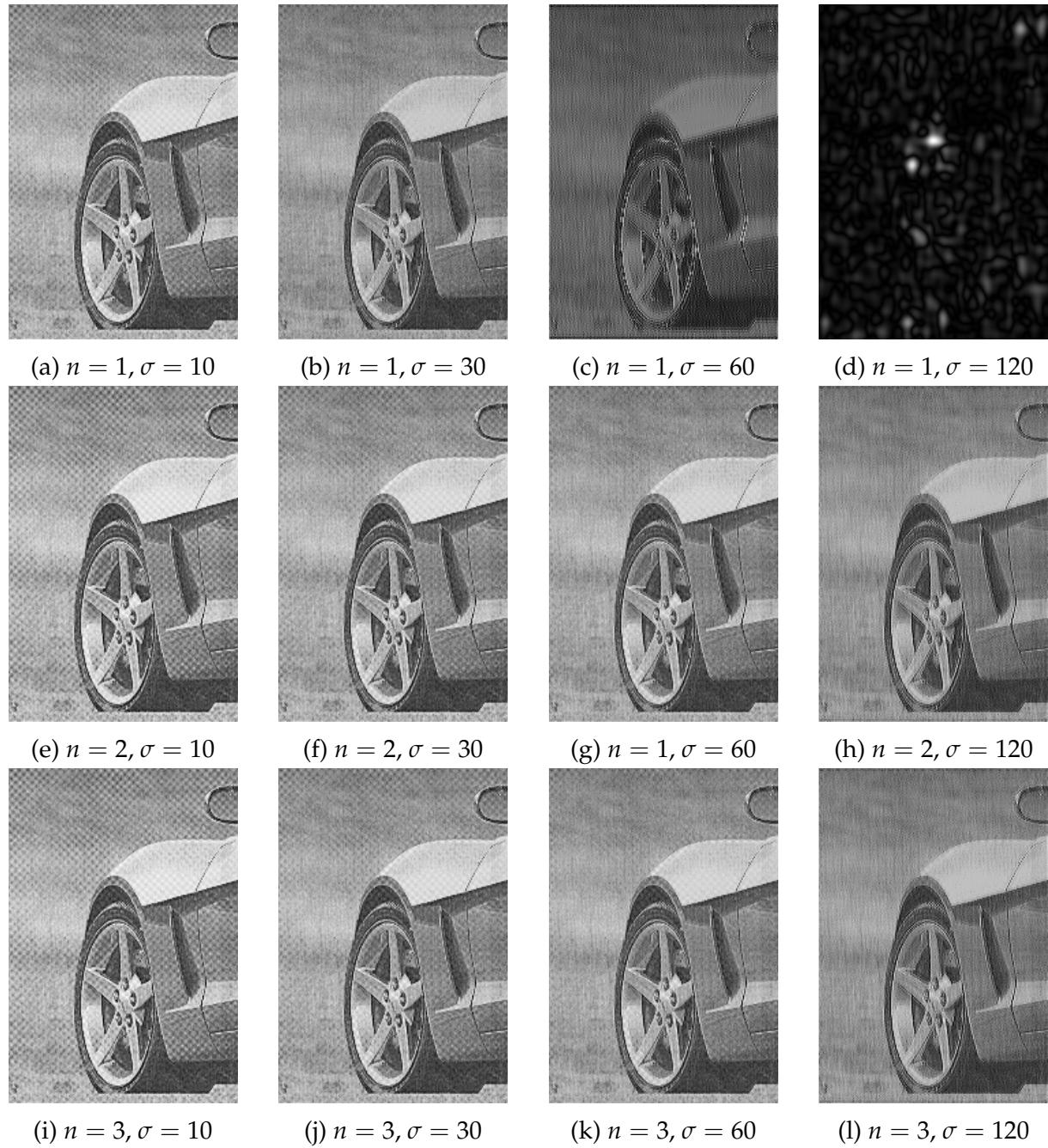


Figure 12: Apply butterworth filter

### Discussion.

We found that, with  $n = 1$ , we increase  $\sigma$ , when  $\sigma = 30$ , most of the blemish disappeared, if we keep increase  $\sigma$ , the image begin to be damaged. For  $n = 2$ , blemish disappeared at  $\sigma = 60$ , then image begin to be damaged, and for  $n=3$ , this number is bigger. We can find from Figure 12 that with  $n=1$  the butterworth filter is similar to Gaussian filter, and for bigger  $n$  it is similar to ideal lowpass filter. So both increase  $n$  and  $\sigma$  can let more signal in low frequency remain, and we also find that when most of the blemish is removed, smaller  $n$  can lead to less damage to image.

So generally, if using butterworth filter to reduce the blemish in the figure, we can use small  $n$  so only a small area will be totally moved, and we need to use relative high  $\sigma$  to make sure the blemish is removed.

## Conclusion

In this lab we applied sobel filter on both spatial and frequency domain, we use Gaussian and Butterworth lowpass and lowpass filter in frequency domain.

Filtering in spatial domain focus on the relationship between pixels within a specific domain, and filtering in frequency domain focus on the speed pixels are chaning in all image and adjust the relationship.

In my opinion, for remove unwanted signal from a image, we can use both spatial and frequency filtering, while filtering in frequency often have better performance, filtering in spatial domain require less computing, and for edge detection, I recommend to use spatial filter due to the error in digital calculating may cause bad disturb in the result.

## Source code

```
1  """
2  Library USE for EE326 2021
3  """
4
5  import numpy as np
6  from skimage import io, data
7  import math
8  from scipy import interpolate
9  import matplotlib.pyplot as plt
10
11
12 # General
13 def format_image(input_image):
14     output_image = input_image
15     output_image -= np.min(output_image)
16     output_image = (output_image/np.max(output_image))*255
17     return output_image
18
19 # LAB 4
20
21
22 def convolution_3x3(input_image, operator_3x3):
23     col, row = input_image.shape
24     output_image = np.zeros([col, row])
25     input_image = np.pad(input_image, 1)
26     for i in range(0, col):
27         for j in range(0, row):
28             for i2 in range(3):
29                 for j2 in range(3):
30                     output_image[i, j] += input_image[i+i2, j+j2] *
31                         operator_3x3[i2, j2]
32
33
34
35 def sobel_filter(input_image):
36
37     operator1 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
38     operator2 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
39
40     output_image1 = convolution_3x3(input_image, operator1)
41     output_image2 = convolution_3x3(input_image, operator2)
42     #
43     # output_image1 = np.clip(output_image1, 0, 255)
44     # output_image2 = np.clip(output_image2, 0, 255)
```

```
45
46     output_image = output_image1 + output_image2 # + input_image
47     # output_image = np.clip(output_image, 0, 255)
48
49     output_image = output_image.astype(np.uint8)
50
51     return output_image
52
53
54 def zero_padding(input_image, P, Q):
55     output_image = np.zeros([P, Q])
56
57     return output_image
58
59 # LAB 5
60
61
62 def extract_result_eastsouth(input_image):
63     x, y = input_image.shape
64     output_image = input_image[int(x/2):x, int(y/2):y]
65
66     return output_image
67
68
69 def extract_result_westnorth(input_image):
70     x, y = input_image.shape
71     output_image = input_image[0:int(x/2), 0:int(y/2)]
72
73     return output_image
74
75
76 def zero_padding_DFT(input_image, P, Q):
77     m, n = input_image.shape
78
79     output_image = np.zeros([P, Q])
80     output_image[0:m, 0:n] = input_image
81
82     return output_image
83
84
85 def zero_padding_DFT(input_image):
86     m, n = input_image.shape
87
88     P = 2*m
89     Q = 2*n
90
91     output_image = np.zeros([P, Q])
```

```
92     output_image[0:m, 0:n] = input_image
93
94     return output_image
95
96
97 def centering(size):
98     m, n = size
99     centering_matrix = np.ones(size)
100    mul1 = 1
101    for i in range(m):
102        mul2 = mul1
103        for j in range(n):
104            centering_matrix[i, j] = centering_matrix[i, j] * mul2
105            mul2 *= -1
106        mul1 *= -1
107    return centering_matrix
108
109
110 def generating_from_spatial_filter(input_filter, P, Q):
111     output_filter = np.zeros(P, Q)
112
113     return output_filter
114
115
116 def gaussian_filter(a, b, sigma):
117     x, y = np.meshgrid(np.linspace(0, a-1, a), np.linspace(0, b-1, b))
118     x = x - a/2
119     y = y - b/2
120     d = x * x + y * y
121     g = np.exp(-(d / (2.0 * sigma ** 2)))
122     # g = g/np.sum(g)
123     return g
124
125
126 def butterworth_filter(b, a, center, n, sigma):
127     cx, cy = center
128     x, y = np.meshgrid(np.linspace(0, a - 1, a), np.linspace(0, b - 1,
129         ↴ b))
130     x = x - cx
131     y = y - cy
132     d = np.sqrt(x * x + y * y)
133     h = 1/((1+(d/sigma))**(2*n))
134     return h
135
136 if __name__ == '__main__':
137     test_input = np.array([[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1,
138         ↴ 2, 3, 4]])
```

```
138     print(transform_centering(test_input))
```

```
1 import numpy as np
2 import numpy.fft
3 from skimage import io, data
4 import math
5 from scipy import interpolate
6 import matplotlib.pyplot as plt
7 from matplotlib import cm
8 import matplotlib.image as mplimg
9 import EE326_SUSTech

10
11
12 def sobel_filter_11810818(input_image):
13     m, n = input_image.shape
14     kernel1 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
15     kernel2 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
16     kernel = kernel1 + kernel2

17
18     # Filtering in the spatial Domain
19     # output_spatial = EE326_SUSTech.convolution_3x3(input_image,
20     #                                                 ← kernel)
21     # plt.imsave("Q5_1_spatial.png",
22     #             output_spatial,
23     #             cmap=cm.gray)

24     # Filtering in the Frequency Domain
25     kernel = np.flip(kernel)
26     input_image = np.pad(input_image, ((0, m), (0, n)))

27
28     input_image = np.multiply(input_image,
29     ← EE326_SUSTech.centering(input_image.shape))
30     input_image = np.fft.fft2(input_image)

31     mplimg.imsave("Q5_1_spectrum.png",
32                 np.log(np.abs(input_image)),
33                 cmap=cm.gray)
34     sz = (input_image.shape[0] - kernel1.shape[0],
35           input_image.shape[1] - kernel1.shape[1])
36     kernel = np.pad(kernel,
37                     (((sz[0] + 1) // 2, sz[0] // 2),
38                      ((sz[1] + 1) // 2, sz[1] // 2)))

39
40     kernel = np.multiply(kernel, EE326_SUSTech.centering(kernel.shape))
41     kernel_fft = np.fft.fft2(kernel)

42
```

```

43     mplimg.imsave("Q5_1_filter.png",
44             (np.abs(kernel_fft)),
45             cmap=cm.gray)
46
47     filtered = np.multiply(input_image, kernel_fft)
48
49     mplimg.imsave("Q5_1_spectrum_filtered.png",
50             (np.abs(filtered)),
51             cmap=cm.gray)
52     filtered = np.fft.ifft2(filtered)
53     filtered = np.multiply(filtered,
54         EE326_SUSTech.centering(filtered.shape))
55
56     output_image = EE326_SUSTech.extract_result_eastsouth(filtered)
57     output_image = np.real(output_image)
58
59     mplimg.imsave("Q5_1_frequency.png",
60             (output_image),
61             cmap=cm.gray)
62
63 if __name__ == '__main__':
64     sobel_filter_11810818(io.imread("Q5_1.tif"))

```

```

1 import numpy as np
2 import numpy.fft
3 from skimage import io, data
4 import math
5 from scipy import interpolate
6 import matplotlib.pyplot as plt
7 from matplotlib import cm
8 import matplotlib.image as mplimg
9 import EE326_SUSTech
10
11
12 def gaussian_pass_11810818(input_image):
13
14     # input_image = EE326_SUSTech.zero_padding_DFT(input_image)
15     x, y = input_image.shape
16
17     input_image = np.fft.fft2(input_image)
18     input_image = np.fft.fftshift(input_image)
19
20     mplimg.imsave("Q5_2_spectrum.png",
21                 (np.abs(input_image)),
22                 cmap=cm.gray)

```

```

23
24     for sigma in [1, 10, 30, 60, 160]:
25         filter_lowpass = EE326_SUSTech.gaussian_filter(x, y, sigma)
26         output_image_frquency = np.multiply(input_image, filter_lowpass)
27         output_image = np.fft.fftshift(output_image_frquency)
28         output_image = (np.abs(np.fft.ifft2(output_image)))
29
30         mplimg.imsave("Q5_2_lowpass_filter_"+str(sigma)+".png",
31                         (np.abs(filter_lowpass)),
32                         cmap=cm.gray)
33         mplimg.imsave("Q5_2_lowpass_"+str(sigma)+".png",
34                         (np.abs(output_image)),
35                         cmap=cm.gray)
36
37         filter_highpass = np.ones((x, y)) - filter_lowpass
38         output_image_frquency = np.multiply(input_image,
39             ~ filter_highpass)
40         output_image = np.fft.fftshift(output_image_frquency)
41         output_image = np.real(np.fft.ifft2(output_image))
42
43         mplimg.imsave("Q5_2_highpass_filter_" + str(sigma) + ".png",
44                         (np.abs(filter_highpass)),
45                         cmap=cm.gray)
46         mplimg.imsave("Q5_2_highpass_" + str(sigma) + ".png",
47                         (np.abs(output_image)),
48                         cmap=cm.gray)
49
50 if __name__ == '__main__':
51     gaussian_pass_11810818(io.imread("Q5_2.tif"))

```

```

1 import numpy as np
2 import numpy.fft
3 from skimage import io, data
4 import math
5 from scipy import interpolate
6 import matplotlib.pyplot as plt
7 from matplotlib import cm
8 import matplotlib.image as mplimg
9 import EE326_SUSTech
10
11
12 def butterworth_notch_filters_11810818(input_image):
13     input_image = EE326_SUSTech.zero_padding_DFT(input_image)
14     x, y = input_image.shape
15

```

```
16     input_image = np.fft.fft2(input_image)
17     input_image = np.fft.fftshift(input_image)
18
19     show_image = np.abs(input_image)
20     mplimg.imsave("Q5_2_spectrum.png",
21                     show_image,
22                     cmap=cm.gray)
23
24     for sigma in [10, 30, 60, 90, 120, 160]:
25         for n in [1, 2, 3]:
26             centers = [
27                 [109, 87],
28                 [109, 170],
29                 [115, 330],
30                 [115, 412],
31                 [227, 405],
32                 [227, 325],
33                 [223, 162],
34                 [223, 79]
35             ]
36
37         filter_lowpass = np.zeros([x, y])
38
39         for center in centers:
40             filter_lowpass += EE326_SUSTech.butterworth_filter(x, y,
41                                         center, n, sigma)
42
43         filter_highpass = np.ones([x, y]) - np.clip(filter_lowpass,
44                                     0.00001, 0.99999)
45         output_image = np.multiply(input_image, filter_highpass)
46
47         mplimg.imsave("Q5_3_spectrum_filtered_" + str(n) + "_" +
48                         str(sigma) + ".png",
49                         (np.abs(output_image)),
50                         cmap=cm.gray)
51
52         output_image = np.fft.fftshift(output_image)
53         output_image = np.abs(np.fft.ifft2(output_image))
54         output_image =
55             EE326_SUSTech.extract_result_westnorth(output_image)
56
57         mplimg.imsave("Q5_3_filter_" + str(n) + "_" + str(sigma) + +
58                         ".png",
59                         (np.abs(filter_highpass)),
60                         cmap=cm.gray)
61
62         mplimg.imsave("Q5_3_" + str(n) + "_" + str(sigma) + ".png",
```

```
58         (np.abs(output_image)),
59         cmap=cm.gray)
60
61
62 if __name__ == '__main__':
63     butterworth_notch_filters_11810818(io.imread("Q5_3.tif"))
```