

Lab 6 Frequency Domain Filtering

Student name & ID: YUAN Tong 11810818

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

Date: *April 29, 2021*

Introduction

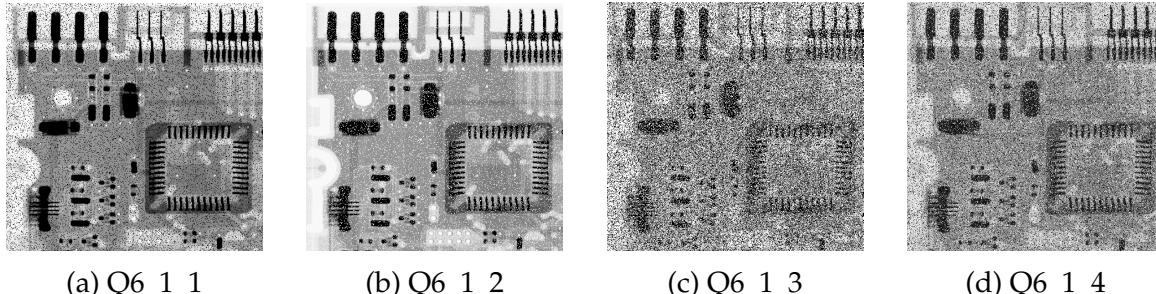
The objective of restoration is to improve a given image in some predefined sense. Compare with image enhancement which is largely a subjective process, image restoration is for the most part an objective process. When we apply image restoring to a degraded image, we will first construct a model about how the image was degraded and then we apply the degrade filter to the restoring filter. After that we could get the reconstructed image, however, we can't build the degrade model precisely and there may be some overlap in frequency domain after the image was degrade, so we can not always restore that image like what it used to be.

In this lab, there are three tasks will be performed.

1. Apply adaptive filter to the image with high noise intensity
2. Apply full inverse filtering, radially limited inverse filtering and Wiener filtering to a image degraded by atmosphere turbulence. Discuss how the parameters, if any, are determined, and the different effects by using the different algorithms.
3. Restore a image degraded by motion blur and noise.

Task I: Adaptive Filter for Image Denoising

Remove the noise from the input images Q6_1_1.tif, Q6_1_2.tif, Q6_1_3.tif and Q6_1_4.tif. Explain your observation and the method used to each of the images, and why such methods are used.



(a) Q6_1_1

(b) Q6_1_2

(c) Q6_1_3

(d) Q6_1_4

Figure 1: Task I, Figures with noise

Analysis. For traditional noise filter in spatial domain, the size of the filter is limited, here comes the problem: if the size of the filter is too small, for areas with high noise intensity, the filter can't reduce all the noise, for example, when apply a median filter to a 3x3 square with more than 5 pixels are noise with 255, the median will always be the 255, for average filter, the number of the result will be greatly affected by the noise.

To solve this problem, two different methods are designed.

Smart filter. The first method is called "smart filter". The filter will analyze the noise situation of the square being filtered, if the noise intensity in the square is high, some pixels in extremely high or low intensity will be removed in future operation. The procedure of the algorithm is shown below.

Algorithm 1: Smart denoise filter

input : The image with noise

output: The denoised image

forall Pixels in the given image **do**

if Pixels with intensity between (5, 250) is contained in the image **then**

Find the normal pixel most close to the middle

else

Calculate the average value

Assign the value to the related pixels in the output image

Adaptive filter. In this method, if the given square has high noise intensity, we will expand the size of the filter until the noise intensity in the domain fit our requirement. The procedure is shown below.

Algorithm 2: Smart denoise filter

input : The image with noise, the initial size, the max size

output: The denoised image

S_{xy} : The domain we apply the filter

z_{min} : The minimum intensity value in S_{xy}

z_{max} : The maximum intensity value in S_{xy}

z_{med} : median intensity value in S_{xy}

z_{xy} : intensity value at coordinates (x, y)

S_{max} : maximum allowed size of S_{xy}

forall Pixels in the given image **do**

repeat

$A_1 = z_{med} - z_{min}$

$A_2 = z_{med} - z_{max}$

if $A_1 > 0$ and $A_2 < 0$ **then**

$B_1 = z_{xy} - z_{min}$

$B_2 = z_{xy} - z_{max}$

if $B_1 > 0$ and $B_2 < 0$ **then**

└ output z_{xy}

else

└ output z_{med}

else

Increase the size of S_{xy} **if** size < S_{max} **then**

└ repeat the loop

else

└ output z_{med}

until output is generated

Result. After applying both filter to the given given image, we obtain the following results

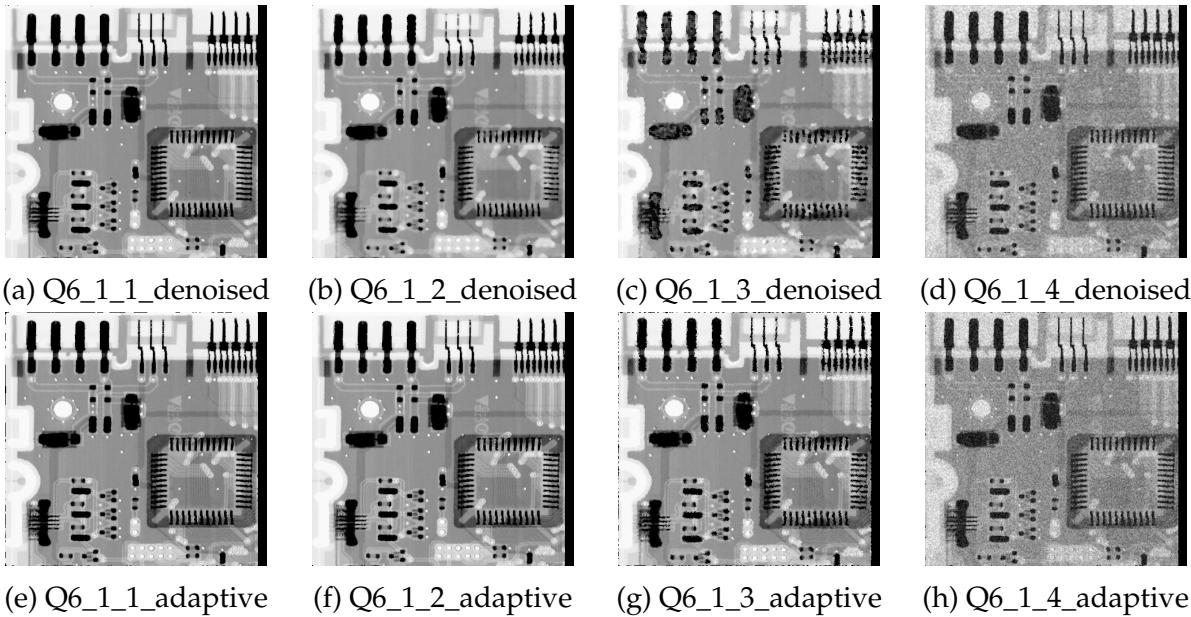


Figure 2: The first line is denoised by smart denoise filter, the second line is denoised by adaptive filter.

Discussion. From the result we found that, as the intensity of the noise increase, some noise in relatively low frequency domain will remain in the image, that is because the noise have cover most of the information in the image, so some information cannot be restored.

Compare two different method, we found that the adaptive filter cause less damage to the details. Previous study had proved that filter with larger size have better ability to reduce the noise but may cause change of some features, the adaptive filter has overcome this problem, provides a method with better noise reducing ability and less image damage.

Task II: Image Restoration

Image Q6_2.tif was degraded from an original image due to the atmosphere turbulence given on slide 65 with $k = 0.0025$. Restore the original image from the input Q6_2.tif by using full inverse filtering, radially limited inverse filtering and Wiener filtering. Discuss how the parameters, if any, are determined, and the different effects by using the different algorithms.

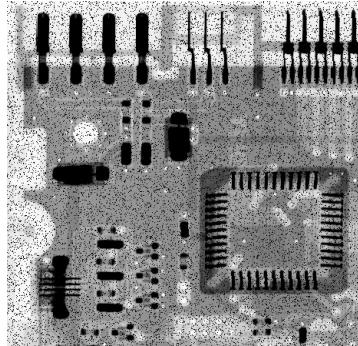


Figure 3: Q6_1_1

Analysis. In this task we will apply three different kinds of inverse filters including full inverse filtering, radially limited inverse filtering and Wiener filtering. Then we will compare the performance of each filter.

full inverse filtering. The image degrade process can be modeled as image filtering process with degrade filter

$$H(u, v)$$

and get result $G(u, v)$, so most directly, the process can be inversed by constructing the inverse filter

$$\frac{1}{H(u, v)}$$

Then the inverse filter will be applied to the degraed image. Here we have

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)}$$

In this task we will apply the filters to a image degraded by atmosphere turbulence blur

$$H(u, v) = e^{-k(u-M/2)^2 + (v-N/2)^2}$$

with $k = 0.0025$.

Radially limited inverse filtering. The edge of most of the lowpass is close to 0 so when we apply the inverse filter, due to the error in digital computing and image storing, the noise in high frequency will be greatly enhanced, so to avoid the situation, we will add a cutoff filter, here a Butterworth lowpass function of order 10. This provided a sharp (but smooth) transition at the desired radius.

Wiener filtering. Here we discuss an approach that incorporates both the degradation function and statistical characteristics of noise into the restoration process. So that the image degrading during image storing and computing process can be taken into consideration.

The filter can be obtained by the expression

$$\hat{F}(u, v) = \left[\frac{1}{H(u, v)} \frac{|H(u, v)|^2}{|H(u, v)|^2 + S_\eta(u, v) / S_f(u, v)} \right] G(u, v)$$

However for most of the case we can not get the exact value of $S_\eta(u, v) / S_f(u, v)$ which represent the signal noise ratio (SNR) of the image, so we will represent it by K so the expression become

$$\hat{F}(u, v) = \left[\frac{1}{H(u, v)} \frac{|H(u, v)|^2}{|H(u, v)|^2 + K} \right] G(u, v)$$

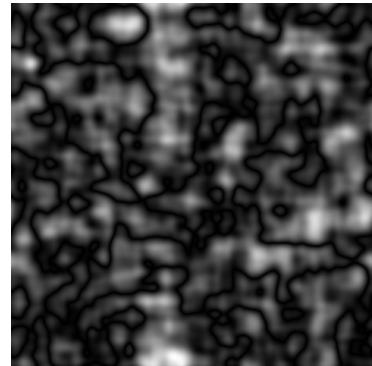


Figure 4: Full inverse filtering

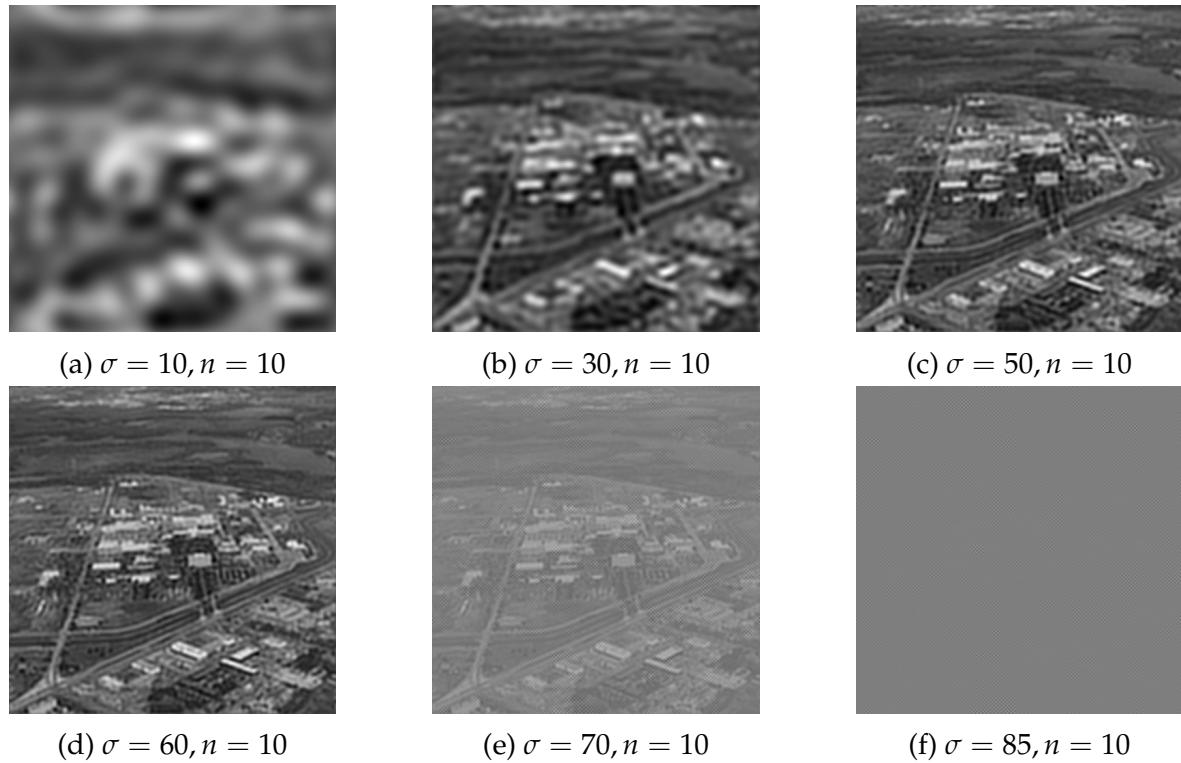


Figure 5: Filtered by radially limited inverse filter

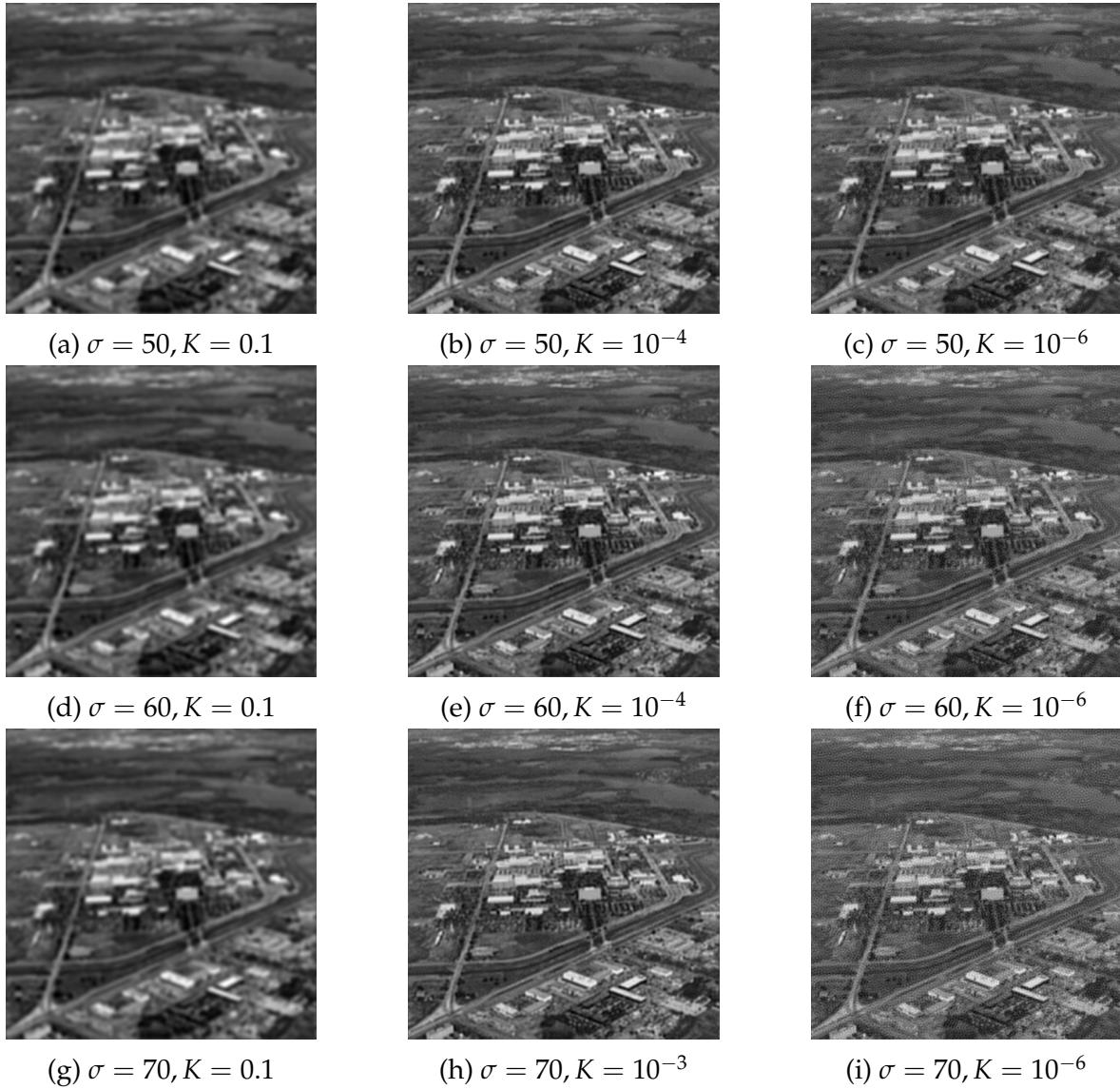


Figure 6: Filtered by wiener filter

Result.

Discussion. The result shows that full inverse filter is not capable for most of the case due to the error that may occur that we discussed in the Analysis section.

Fig 5 shows that $\sigma = 60$ can be a proper cut off frequency for the filter, when σ keeps increase, the high frequency noise will appear again, when σ keeps decrease, the details in the image will also be filtered.

Fig 6 shows that $\sigma = 60, K = 10^{-4}$ can be a proper pair to restore the image by wiener filter, comparing between each image in the image set, when K is decreasing, more details in the image will be restored while the noise increase, that because the detail of the image and noise are similar somehow, so when less detail is filtered, less noise will be filtered. The adjustment of σ has been discussed in last paragraph.

Compare between two filters, we found that obviously wiener filter shown better performance, one possible reason can be that during the store of the image, some information, such as phase, lost, so the inverse filter can't restore the information perfectly. But for wiener filter, such kind of information loss can be treated as noise and

can be reduced.

Task III: Motion Deblurring

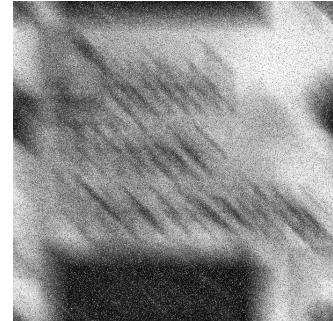
Restore the original images from the inputs Q6_3_1.tif, Q6_3_2.tif and Q6_3_3. Explain your observation and the method used.



(a) Q6_3_1



(b) Q6_3_2



(c) Q6_3_3

Figure 7: Task III, Figures with motion blur

Analysis. In this task we will reconstruct the image degraded by motion blur and noise. The motion blur can be expressed as filter

$$\begin{aligned}
 H(u, v) &= \int_0^T e^{-j2\pi[ux_0(t)+vy_0(t)]} dt \\
 * &\quad = \int_0^T e^{-j2\pi[ua+vb]t/T} dt \\
 &= \frac{T}{\pi(ua + vb)} \sin[\pi(ua + vb)] e^{-j\pi(ua + vb)}
 \end{aligned} \tag{1}$$

And we will apply radially limited inverse filter and wieber filter to the degreaded image.

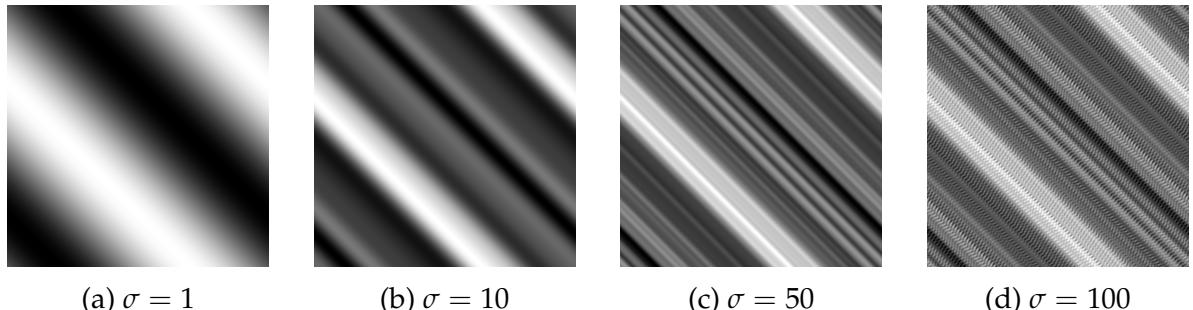


Figure 8: Filtered by radially limited inverse filter

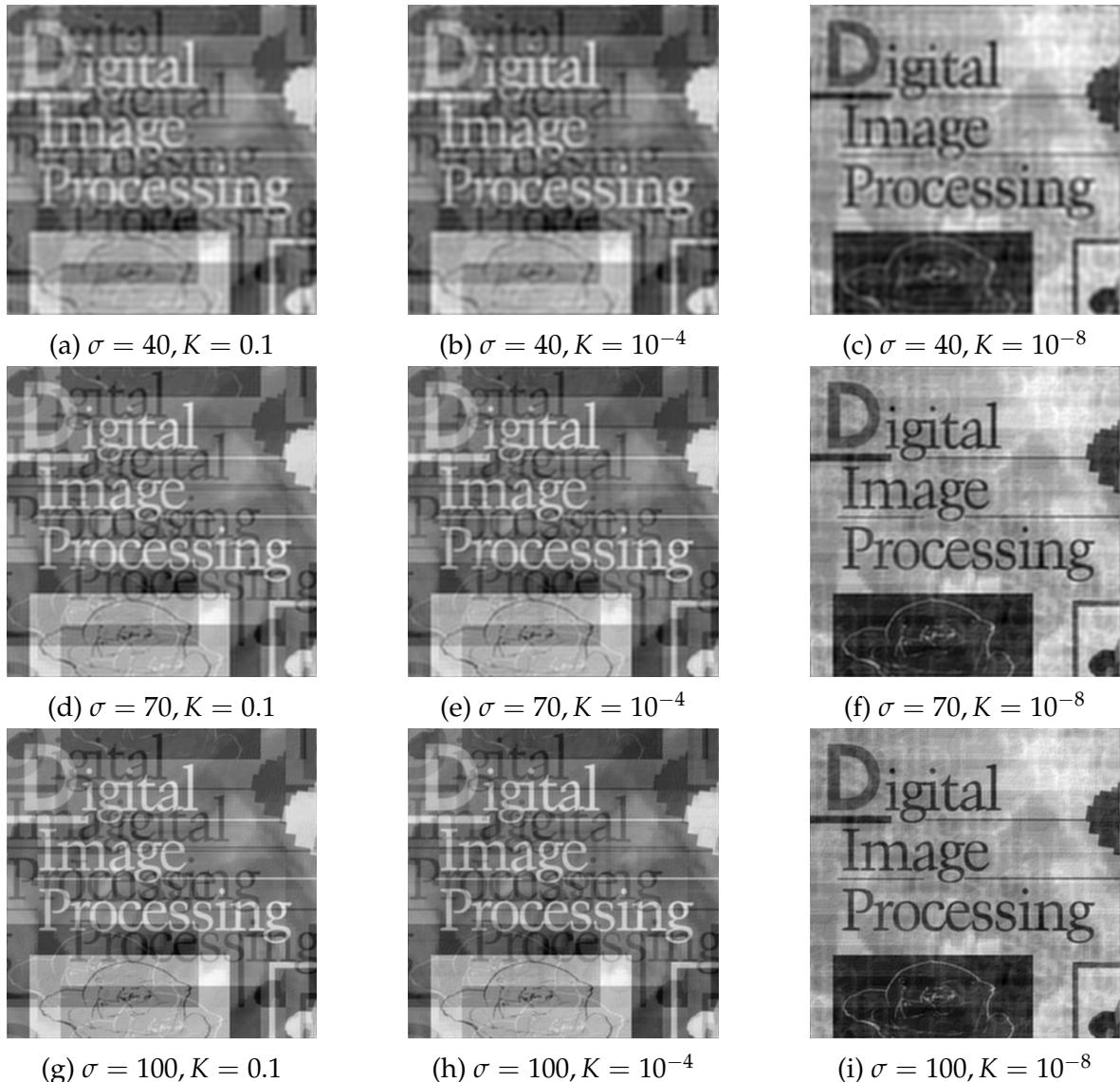


Figure 9: Filtered by wiener filter

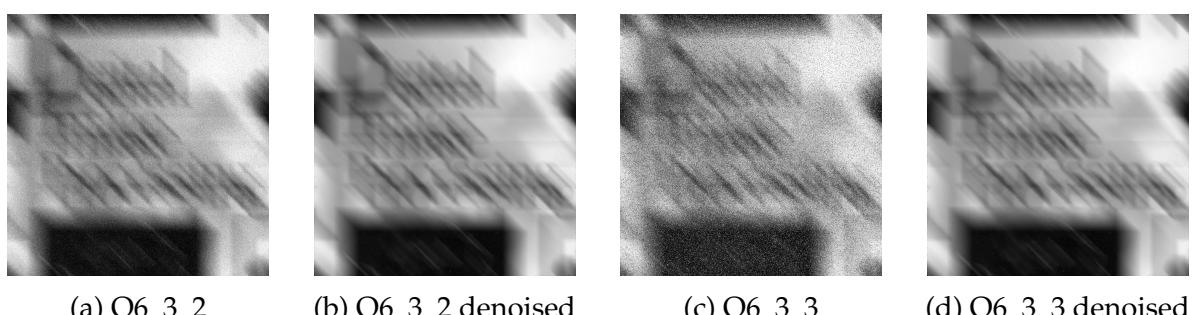


Figure 10: original image and denoised image

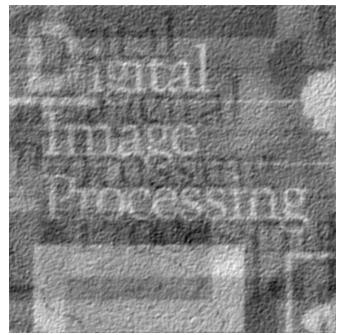
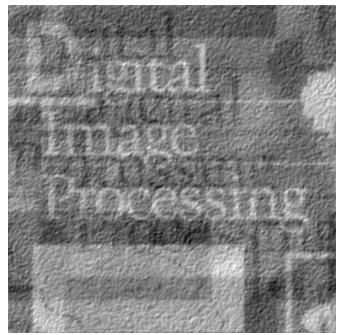
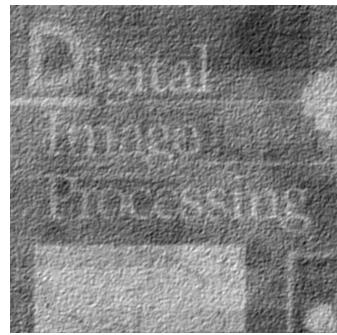
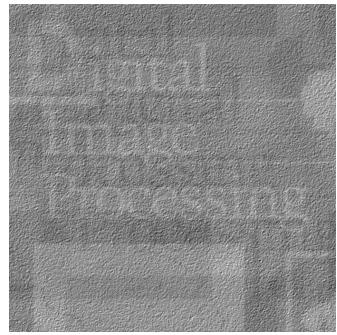
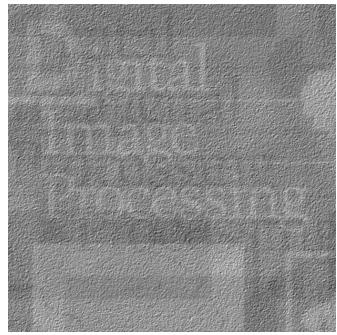
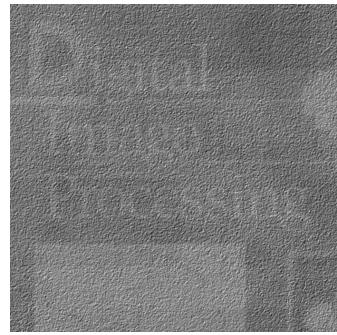
(a) $\sigma = 10, K = 0.25$ (b) $\sigma = 10, K = 0.0025$ (c) $\sigma = 10, K = 2.5 \times 10^{-5}$ (d) $\sigma = 50, K = 0.25$ (e) $\sigma = 50, K = 0.0025$ (f) $\sigma = 50, K = 2.5 \times 10^{-5}$ (g) $\sigma = 100, K = 0.25$ (h) $\sigma = 100, K = 0.0025$ (i) $\sigma = 100, K = 2.5 \times 10^{-5}$

Figure 11: motion deblur of the figure

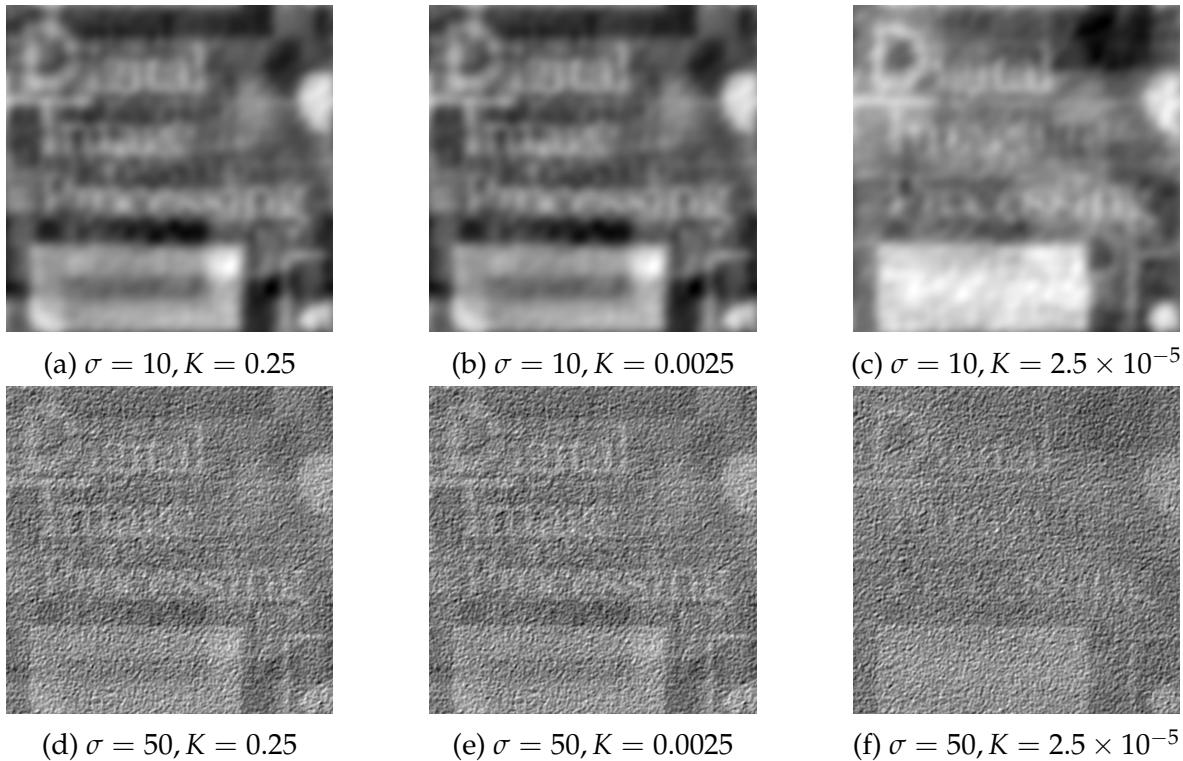


Figure 12: motion deblur of the figure with more noise

Result.

Discussion. Fig 8 indicate that we can't use radially limited inverse filter to restore the image. Here a unit impluse is used to estimate the behavior of the filters

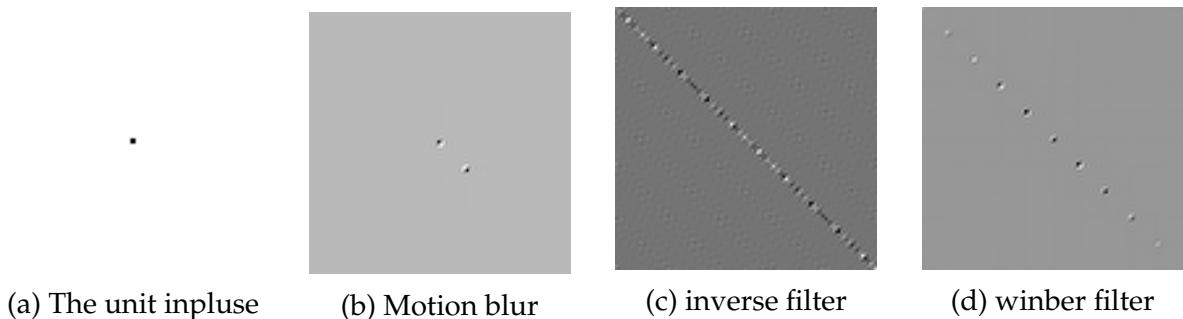


Figure 13: The behavier of the system to unit impluse

After the system is applied to the unit impluse response, we can see a motion blur appeared, then two inverse filters are applied to the degraded image. By inverse filter, the motion blur become more severe, for winber filter, the impluse appear periodically and their intensity decrease when their distance to the center of the image increase.

However, in previous process, we will save the image and read it again before we apply the inverse filter to it, to save the image, we need to apply `abs()` function to convert it from complex number to pure real number, so during this process some information is lost. Which create a special kind of noise, then, in further process, the noise is enhanced by the inverse filter. That's why we get useless result in Fig 8.

Fig 9 shows the result after restored by wiener filter, here $\sigma = 100, K = 10^{-4}$ is a proper pair to restore the image. Compare between image, if the details of the image is not clear enough, we just increase σ , if the shadow is to outstanding, we increase the K and if the noise appeared, we increase K.

In addition, the reason of the shadow have been analysed when we apply the system to the unit impulse.

Then we try to apply the filter to image with noise, first we denoise the image by adaptive filter we introduced in section 1, after filtering 3-5 times, we get the filtered image with most of the noise removed, then the wiener filter is applied to the image. In the result we found that $\sigma = 50K = 0.25$, is a proper pair to restore the image, however, there is still a layer of noise in the result, that is because, the denoise process also create lot of noise with different motion to the image, so the noise is also enhanced during the filtering. And for image with higher noise, as shown in Fig 12, the affect of such kind of noise become more severe.

Conclusion

In this lab, we studied adaptive image denoise and image reconstruction method, we found that the adaptive filter is a better method for image denoising due to its auto adjusting feature.

When we use image restoring technic, we have one or multiple parameters to adjust. For example, σ in both radially limited inverse filter and wieber filter is used to reduce the high frequency noise, and K in wiener filter can adjust the reduced noise, when we adjust the parameters, one demision of image quality will increase while another decrease, so we need to find a balance between each demision of image quality.

Appendix: Source Code

```

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 from numba import njit, prange
10 import EE326_SUSTech as ee
11 import time

12
13 @njit(parallel=True)
14 def adaptive_11810818(input_image, n_size, smax):
15     output_image = np.zeros(input_image.shape, dtype=np.uint8)
16     m, n = input_image.shape

17
18     for i in prange(m):
19         for j in prange(n):
20             n_size_2 = n_size

21
22         while True:
23             step = (int)((n_size_2 - 1) / 2)
24             pixels = np.zeros(n_size_2 * n_size_2)

25
26             for i2 in range(n_size_2):
27                 for j2 in range(n_size_2):
28                     if i - step + i2 >= 0 \
29                         and i - step + i2 < input_image.shape[0]
29                         ↵ \
30                         and j - step + j2 >= 0 \
31                         and j - step + j2 <
31                         ↵ input_image.shape[0]:
32                         pixels[j2 * n_size_2 + i2] = input_image[i -
32                         ↵ step + i2, j - step + j2]

33
34             pixels_sorted = np.sort(pixels)
35             med = (int)((n_size_2 * n_size_2-1)/2)
36             a1 = pixels_sorted[med] - pixels_sorted[0]
37             a2 = pixels_sorted[med] - pixels_sorted[n_size_2 *
37             ↵ n_size_2-1]
38             if(a1>0 and a2<0):
39                 b1 = input_image[i, j] - pixels_sorted[0]
40                 b2 = input_image[i, j] - pixels_sorted[n_size_2 *
40                 ↵ n_size_2-1]

```

```

41         if(b1>0 and b2<0):
42             output_image[i, j] = pixels[med]
43         else:
44             output_image[i, j] = pixels_sorted[med]
45         break
46     else:
47         if(n_size_2 < smax):
48             n_size_2 += 2
49         else:
50             output_image[i, j] = pixels_sorted[med]
51             break
52
53     return output_image
54
55 if __name__ == '__main__':
56     for i in [1
57                 , 2
58                 , 3
59                 , 4
60                 ]:
61         start_time = time.time()
62         input_image = "Q6_1_" + str(i)
63         output_name = "plots/" + str(input_image) + "_adaptive.png"
64         input_image = io.imread(input_image + ".tiff")
65         output_image = adaptive_11810818(input_image, 3, 20)
66         print(time.time() - start_time)
67         mplimg.imsave(output_name,
68                         output_image,
69                         cmap=cm.gray)
70
71     print("Finish processing " + str(i))

```

```

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 as ee
10 import time
11
12
13 def atmosphere_turbulence(shape, k):
14     col, row = shape

```

```
15     u, v = np.meshgrid(np.linspace(0, col-1, col), np.linspace(0, row-1,
16                           ↳   row))
17     u = u - col / 2
18     v = v - row / 2
19     d = u * u + v * v
20     h = np.exp(-(k * (d ** (5/6))))
21
22
23
24 def full_inverse_filtering_11810818(input_image):
25     input_image = io.imread(input_image + ".tif")
26     m, n = input_image.shape
27     input_image = np.pad(input_image, ((0, m), (0, n)))
28
29     filter = atmosphere_turbulence(input_image.shape, 0.0025)
30     inverse_filter = 1/filter
31
32     input_image = np.fft.fft2(input_image)
33     input_image = np.fft.fftshift(input_image)
34     input_image = input_image*inverse_filter
35     input_image = np.fft.fftshift(input_image)
36     output_image = np.abs(np.fft.ifft2(input_image))
37
38     return output_image
39
40 def radially_limited_inverse_filtering_11810818(input_image, sigma):
41     input_name = input_image
42     input_image = io.imread(input_image + ".tif")
43     m, n = input_image.shape
44     # input_image = np.pad(input_image, ((0, m), (0, n)))
45     # m, n = input_image.shape
46
47     filter = atmosphere_turbulence(input_image.shape, 0.0025)
48     inverse_filter = np.reciprocal(filter)
49     # g = ee.gaussian_filter(m, n, sigma)
50     g = ee.butterworth_filter(m, n, [m/2, n/2], 10, sigma)
51
52     input_image = np.fft.fft2(input_image)
53     input_image = np.fft.fftshift(input_image)
54     input_image = input_image * inverse_filter * g
55     input_image = np.fft.ifftshift(input_image)
56     output_image = np.real(np.fft.ifft2(input_image))
57
58     return output_image
59
60 def wiener_filter_11810818(input_image, sigma, k):
```

```

61     input_image = io.imread(input_image + ".tif")
62     m, n = input_image.shape
63     # input_image = np.pad(input_image, ((0, m), (0, n)))
64     # m, n = input_image.shape
65
66     g = ee.gaussian_filter(m, n, sigma)
67     filter = atmosphere_turbulence(input_image.shape, 0.0025)
68
69     f = ((1/filter)*(filter**2/(filter**2 + k*np.ones([m, n])))) * g
70
71     input_image = np.fft.fft2(input_image)
72     input_image = np.fft.fftshift(input_image)
73     output_image = input_image*f
74     output_image = np.fft.fftshift(output_image)
75     output_image = np.fft.ifft2(output_image)
76     output_image = np.abs(output_image)
77
78     return output_image
79
80
81 if __name__ == '__main__':
82
83     input_image = "Q6_2"
84     # output_name = "plots/" + str(input_image) + "_full_inverse.png"
85     # output_image = full_inverse_filtering_11810818(input_image)
86     # mplimg.imsave(output_name,
87     #                 output_image,
88     #                 cmap=cm.gray)
89     # print("Finish processing full inverse filtering")
90
91     for i in [10, 30, 35, 40, 45, 50, 55, 60, 65, 70, 85]:
92         start = time.time()
93         output_name = "plots/" + str(input_image) + "_radially_limited"
94         ↵ + str(i) + ".png"
95         output_image =
96             ↵ radially_limited_inverse_filtering_11810818(input_image, i)
97         print(time.time() - start)
98         mplimg.imsave(output_name,
99                         output_image,
100                        cmap=cm.gray)
101
102     print("Finish processing radially limited filtering")
103
104     for sigma in [50, 60, 70]:
105         for K in [0.0000000001, 0.00000001, 0.000001, 0.0001, 0.01,
106             ↵ 0.1]:
107             start = time.time()

```

```

105     output_name = "plots/" + str(input_image) + "_wiener_" +
106         ↵ str(sigma) + "_" + str(K) + ".png"
107     output_image = wiener_filter_11810818(input_image, sigma, K)
108     print(time.time() - start)
109     mplimg.imsave(output_name,
110                     output_image,
111                     cmap=cm.gray)
112
113     print("Finish processing wiener filtering")

```

```

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 as ee
10 import adaptive_filter_11810818 as ada
11
12
13 def undergo(a, b, u, v, T):
14     u, v = np.meshgrid(np.linspace(1, u, u), np.linspace(1, v, v))
15     d = a * u + b * v
16     #  $h = (T / (np.pi * d)) * np.sin(np.pi * d) * np.exp(-1j * (np.pi * d))$ 
17     h = (T / (np.pi * d)) * np.sin(d * np.pi) * np.exp(-1 * 1j * np.pi * d)
18
19     return h
20
21
22
23 def radially_limited_inverse_filtering_11810818(input_image, sigma):
24     input_name = input_image
25     input_image = io.imread(input_image + ".tiff")
26
27     # input_image = ada.adaptive_11810818(input_image, 3, 30)
28
29     m, n = input_image.shape
30     # input_image = np.pad(input_image, ((0, m), (0, n)))
31     # m, n = input_image.shape
32
33     filter = undergo(0.1, 0.1, m, n, 1)
34     inverse_filter = np.reciprocal(filter)

```

```
35 g = ee.gaussian_filter(m, n, sigma)
36
37 input_image = np.fft.fft2(input_image)
38 input_image = np.fft.fftshift(input_image)
39
40 input_image = input_image * inverse_filter * g
41 # input_image = input_image * inverse_filter
42
43 input_image = np.fft.ifftshift(input_image)
44 output_image = np.real(np.fft.ifft2(input_image))
45
46 return output_image
47
48
49 def wiener_filter_11810818(input_image, sigma, k):
50     input_image = io.imread(input_image + ".tiff")
51     # input_image = ada.adaptive_11810818(input_image, 3, 30)
52     m, n = input_image.shape
53     # input_image = np.pad(input_image, ((0, m), (0, n)))
54     # m, n = input_image.shape
55
56     g = ee.gaussian_filter(m, n, sigma)
57     filter = undergo(0.1, 0.1, m, n, 1)
58
59     # f = ((1/filter)*(filter*np.conj(filter))/(filter*np.conj(filter) +
60     # → k))) * g
61     # f = ((filter * np.conj(filter)) / (filter * np.conj(filter) + k)))
62
63     buf = filter * np.conj(filter)
64     f = buf / (filter * (buf + k)) * g
65
66     input_image = np.fft.fft2(input_image)
67     input_image = np.fft.fftshift(input_image)
68
69     output_image = input_image * f
70
71     output_image = np.fft.ifftshift(output_image)
72     output_image = np.fft.ifft2(output_image)
73     output_image = np.real(output_image)
74
75
76     return output_image
77
78
79 def test_image():
80     test = np.ones([100, 100])*255
81     test[49:51, 49:51] = [[0, 0], [0, 0]]
```

```
81     mplimg.imsave("test_input.png",
82                     test,
83                     cmap=cm.gray)
84
85     filter = undergo(0.1, 0.1, 100, 100, 1)
86     test = np.fft.fftshift(np.fft.fft2(test))
87     test = test * filter
88     test = np.fft.ifft2(np.fft.fftshift(test))
89     test = np.abs(test)
90
91     mplimg.imsave("test_input_burl.png",
92                     np.real(test),
93                     cmap=cm.gray)
94
95     buf = filter * np.conj(filter)
96     f = buf / (filter * (buf + 0.000000025))
97
98     test = np.fft.fftshift(np.fft.fft2(test))
99     test = test * f
100    test = np.fft.ifft2(np.fft.ifftshift(test))
101
102   mplimg.imsave("test_input_restore_winber.png",
103                   np.real(test),
104                   cmap=cm.gray)
105
106
107
108 def denoise1():
109     for input_name in ["Q6_3_2", "Q6_3_3"]:
110         output_image =
111             ee.adaptive_filter(io.imread(str(input_name)+".tiff"), 3,
112             30)
113         print("Finish denoise 1 " + str(input_name))
114         output_image = ee.adaptive_filter(output_image, 3, 30)
115         print("Finish denoise 2 " + str(input_name))
116         output_image = ee.adaptive_filter(output_image, 3, 30)
117         print("Finish denoise 3 " + str(input_name))
118         output_image = ee.adaptive_filter(output_image, 3, 30)
119         mplimg.imsave(str(input_name) + ".png",
120                         output_image,
121                         cmap=cm.gray)
122         print("Finish denoise " + str(input_name))
123
124
125 def denoise2():
126     for input_name in ["Q6_3_3"]:
127         output_image =
128             ee.adaptive_filter(io.imread(str(input_name)+".tiff"), 3,
129             30)
```

```
126     print("Finish denoise 1 " + str(input_name))
127     output_image = ee.adaptive_filter(output_image, 3, 30)
128     print("Finish denoise 2 " + str(input_name))
129     output_image = ee.adaptive_filter(output_image, 3, 30)
130     print("Finish denoise 3 " + str(input_name))
131     output_image = ee.adaptive_filter(output_image, 3, 30)
132     print("Finish denoise 4 " + str(input_name))
133     output_image = ee.adaptive_filter(output_image, 3, 30)
134     print("Finish denoise 5 " + str(input_name))
135     output_image = ee.adaptive_filter(output_image, 3, 30)
136     mplimg.imsave(str(input_name) + ".png",
137                     output_image,
138                     cmap=cm.gray)
139     print("Finish denoise " + str(input_name))
140
141
142 def run():
143     for input_image in ["Q6_3_3"]:
144         for sigma in [1, 10, 50, 100]:
145             output_name = "plots/" + str(input_image) +
146             " _radially_limited_" + str(sigma) + ".png"
147             output_image =
148                 radially_limited_inverse_filtering_11810818(input_image,
149                 sigma)
150             mplimg.imsave(output_name,
151                             output_image,
152                             cmap=cm.gray)
153             print("Finish processing radially limited filtering")
154
155         for sigma in [1, 10, 20, 30, 40, 50]:
156             for K in [0.25, 0.0025, 0.000025]:
157                 output_name = "plots/" + str(input_image) + "_wiener_" +
158                 str(sigma) + "_" + str(K) + ".png"
159                 output_image = wiener_filter_11810818(input_image,
160                 sigma, K)
161                 mplimg.imsave(output_name,
162                             output_image,
163                             cmap=cm.gray)
164             print("Finish processing wiener filtering")
165
166
167 if __name__ == '__main__':
168     test_image()
169     # run()
170     # denoise2()
```

```
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  from numba import njit, prange
11
12 # General
13
14 def format_image(input_image):
15     output_image = input_image
16     output_image -= np.min(output_image)
17     output_image = (output_image / np.max(output_image)) * 255
18     return output_image
19
20 # LAB 4
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
60 def denoise_filter(input_image, n_size, mode):
61     output_image = np.zeros(input_image.shape, dtype=np.uint8)
62
63     m, n = input_image.shape
64
65     for i in range(m):
66         for j in range(n):
67             step = (int)((n_size - 1) / 2)
68             pixels = np.zeros(n_size * n_size)
69
70             for i2 in range(n_size):
71                 for j2 in range(n_size):
72                     if i - step + i2 >= 0 \
73                         and i - step + i2 < input_image.shape[0] \
74                         and j - step + j2 >= 0 \
75                         and j - step + j2 < input_image.shape[0]:
76                         pixels[j2 * n_size + i2] = input_image[i - step
77                                         + i2, j - step + j2]
78
79             pixels = np.sort(pixels)
80
81             if(mode == "max"):
82                 output_image[i, j] = pixels[pixels.shape[0]-1]
83             elif(mode == "medium"):
84                 output_image[i, j] = pixels[(int)((n_size * n_size + 1)
85                                         / 2)]
86             elif(mode == "min"):
87                 output_image[i, j] = pixels[0]
88             elif(mode == "average"):
89                 output_image[i, j] = np.average(pixels)
90             elif(mode == "smart"):
91                 have_normal_pixel = 0
92                 for pixel in pixels:
93                     if(pixel < 250 and pixel > 5):
```

```
92             have_normal_pixel = 1
93         if(have_normal_pixel):
94             selected = (int)(pixels.shape[0]/2)
95             while(pixels[selected] < 5):
96                 selected = selected + 1
97             while(pixels[selected] > 250):
98                 selected = selected - 1
99             output_image[i, j] = pixels[selected]
100        else:
101            output_image[i, j] = np.average(pixels)
102
103
104
105    return output_image
106
107 # LAB 5
108
109
110 def extract_result_eastsouth(input_image):
111     x, y = input_image.shape
112     output_image = input_image[int(x/2):x, int(y/2):y]
113
114     return output_image
115
116
117 def extract_result_westnorth(input_image):
118     x, y = input_image.shape
119     output_image = input_image[0:int(x/2), 0:int(y/2)]
120
121     return output_image
122
123
124 def zero_padding_DFT(input_image, P, Q):
125     m, n = input_image.shape
126
127     output_image = np.zeros([P, Q])
128     output_image[0:m, 0:n] = input_image
129
130     return output_image
131
132
133 def zero_padding_DFT(input_image):
134     m,n = input_image.shape
135
136     P = 2*m
137     Q = 2*n
138
```

```
139     output_image = np.zeros([P, Q])
140     output_image[0:m, 0:n] = input_image
141
142     return output_image
143
144
145 def centering(size):
146     m, n = size
147     centering_matrix = np.ones(size)
148     mul1 = 1
149     for i in range(m):
150         mul2 = mul1
151         for j in range(n):
152             centering_matrix[i, j] = centering_matrix[i, j] * mul2
153             mul2 *= -1
154         mul1 *= -1
155     return centering_matrix
156
157 @njit
158 def generating_from_spatial_filter(input_filter, P, Q):
159     output_filter = np.zeros(P, Q)
160
161     return output_filter
162
163
164 def gaussian_filter(a, b, sigma):
165     x, y = np.meshgrid(np.linspace(0, a-1, a), np.linspace(0, b-1, b))
166     x = x - a/2
167     y = y - b/2
168     d = x * x + y * y
169     g = np.exp(-(d / (2.0 * sigma ** 2)))
170     # g = g/np.sum(g)
171     return g
172
173
174 def butterworth_filter(b, a, center, n, sigma):
175     cx, cy = center
176     x, y = np.meshgrid(np.linspace(0, a - 1, a), np.linspace(0, b - 1,
177         ↳ b))
178     x = x - cx
179     y = y - cy
180     d = np.sqrt(x * x + y * y) + 0.1
181     h = 1/(1+(sigma/d)**(-2*n))
182     return h
183
184 # LAB 6
```

```

185 @njit(parallel=True)
186 def adaptive_filter(input_image, n_size, smax):
187     output_image = np.zeros(input_image.shape, dtype=np.uint8)
188     m, n = input_image.shape
189
190     for i in prange(m):
191         for j in prange(n):
192             n_size_2 = n_size
193
194             while True:
195                 step = (int)((n_size_2 - 1) / 2)
196                 pixels = np.zeros(n_size_2 * n_size_2)
197
198                 for i2 in range(n_size_2):
199                     for j2 in range(n_size_2):
200                         if i - step + i2 >= 0 \
201                             and i - step + i2 < input_image.shape[0]
202                             ↵ \
203                             and j - step + j2 >= 0 \
204                             and j - step + j2 <
205                             ↵ input_image.shape[0]:
206                             pixels[j2 * n_size_2 + i2] = input_image[i -
207                             ↵ step + i2, j - step + j2]
208
209                 pixels_sorted = np.sort(pixels)
210                 med = (int)((n_size_2 * n_size_2-1)/2)
211                 a1 = pixels_sorted[med] - pixels_sorted[0]
212                 a2 = pixels_sorted[med] - pixels_sorted[n_size_2 *
213                             ↵ n_size_2-1]
214                 if(a1>0 and a2<0):
215                     b1 = input_image[i, j] - pixels_sorted[0]
216                     b2 = input_image[i, j] - pixels_sorted[n_size_2 *
217                             ↵ n_size_2-1]
218                     if(b1>0 and b2<0):
219                         output_image[i, j] = pixels[med]
220                     else:
221                         output_image[i, j] = pixels_sorted[med]
222                     break
223                 else:
224                     if(n_size_2 < smax):
225                         n_size_2 += 2
226                     else:
227                         output_image[i, j] = pixels_sorted[med]
228                         break
229
230             return output_image

```