

# SUSTech\_EE326\_lab6\_Appendix

---

*Topic: Image Restoration*

*Author: 11911521钟新宇*

*Project: lab6 report for Digital Image Processing*

## Outline

### SUSTech\_EE326\_lab6\_Appendix

1. MeanFilters
  - 1.1 arithmetic\_mean
  - 1.2 geometric\_mean
  - 1.3 harmonic\_mean
  - 1.4 contraharmonic\_mean
2. OrderStatisticFilters
  - 2.1 median\_filter
  - 2.2 max\_filter
  - 2.3 min\_filter
  - 2.4 midpoint\_filter
  - 2.5 alpha\_trimmed\_mean
3. AdaptiveFilters
  - 3.1 adaptive\_arithmetic\_mean
  - 3.2 adaptive\_median\_filter
4. Degradation Filters
  - 4.1 full\_inverse
  - 4.2 limit\_inverse
  - 4.3 wiener
5. Degradation Functions
  - 5.1 turbulence
  - 5.2 motion\_blur
6. Top
  - 6.1 lab6\_1
  - 6.2 lab6\_2
  - 6.3 lab6\_3

## 1. MeanFilters

---

## 1.1 arithmetic\_mean

```

1  def arithmetic_mean(input_image, size):
2      img = np.array(input_image, dtype=int)
3      row, col = img.shape
4      step = (size - 1) // 2
5      if step * 2 + 1 >= row or step * 2 + 1 >= col:
6          print("The parameter size is too large.")
7      img_pad = np.pad(input_image, [step, step], 'constant', constant_values=
[0] * 2)
8      img_out = np.array(np.zeros((row, col)))
9      for i in range(step, row):
10         for j in range(step, col):
11             img_out[i - step, j - step] = np.sum(img_pad[i - step:i + step +
1, j - step:j + step + 1]) / (step ** 2)
12         img_out = normalize(img_out)
13     return img_out

```

## 1.2 geometric\_mean

```

1  def geometric_mean(input_image, size):
2      """
3          一般来说，几何平均滤波器的平滑效果 可与算术平均滤波器相媲美，但它会损失较少的图像细
节。
4          注意：如果图像的动态范围很大，我们一般会做log运算，但是对数运算后一般不使用几何平均滤
波器。
5          :param input_image:
6          :param size:
7          :return:
8          """
9      img = np.array(input_image, dtype=float)
10     row, col = img.shape
11     step = (size - 1) // 2
12     if step * 2 + 1 >= row or step * 2 + 1 >= col:
13         print("The parameter size is too large.")
14     img_pad = np.pad(img, [step, step], 'constant', constant_values=[1] * 2)
15     img_out = np.array(np.zeros((row, col)))
16     for i in range(step, row):
17         for j in range(step, col):
18             temp = img_pad[i - step:i + step + 1, j - step:j + step + 1]
19             temp = np.prod(temp)
20             temp = np.power(temp, 1 / (size ** 2))
21             img_out[i - step, j - step] = temp
22     img_out = normalize(img_out)
23     return img_out

```

## 1.3 harmonic\_mean

```

1  def harmonic_mean(input_image, size):
2      """
3      它对盐噪声的效果很好，但对椒噪声则效果不好。它对其他类型的噪声如高斯噪声也有很好的效果。
4      :param input_image:
5      :param size:
6      :return:
7      """
8      img = np.array(input_image, dtype=int)
9      row, col = img.shape
10     step = (size - 1) // 2
11     if step * 2 + 1 >= row or step * 2 + 1 >= col:
12         print("The parameter size is too large.")
13
14     img_pad = np.pad(img, [step, step], 'constant', constant_values=[1] * 2)
15     img_out = np.array(np.zeros((row, col)))
16     for i in range(step, row):
17         for j in range(step, col):
18             temp = np.array(img_pad[i - step:i + step + 1, j - step:j + step
19 + 1], dtype=float)
19             temp = np.reciprocal(temp)
20             temp = (size ** 2) / np.sum(temp)
21             img_out[i - step, j - step] = temp
22     img_out = normalize(img_out)
23     return img_out

```

## 1.4 contraharmonic\_mean

```

1  def contraharmonic_mean(input_image, q, size):
2      """
3      它非常适合于减少椒盐噪声的影响。Q>0处理胡椒噪声，Q<0处理盐噪声。
4      缺点：不能同时处理椒和盐的噪声；
5
6      :param input_image:
7      :param q: Q>0 会导致黑色区域缩小，白色区域放大；Q<0 会导致白色区域缩小，黑色区域放
8      大。
9      :param size:
10     :return:
11     """
12     global q2_array, q_array
13     img = np.array(input_image, dtype=float)
14     row, col = img.shape
15     step = (size - 1) // 2
16     if step * 2 + 1 >= row or step * 2 + 1 >= col:
17         print("The parameter size is too large.")
18
19     img_pad = np.pad(img, [step, step], 'constant', constant_values=[1] * 2)
20     img_out = np.array(np.zeros((row, col)))
21
22     if q > 0:

```

```

22     q_array = np.array(np.maximum(np.zeros((size, size)), q),
dtype=float)
23     q2_array = np.array(np.maximum(np.zeros((size, size)), q + 1),
dtype=float)
24     elif q < 0:
25         q_array = np.array(np.minimum(np.zeros((size, size)), q),
dtype=float)
26         q2_array = np.array(np.minimum(np.zeros((size, size)), q + 1),
dtype=float)
27
28     for i in range(step, row):
29         for j in range(step, col):
30             temp = np.array(img_pad[i - step:i + step + 1, j - step:j + step
+ 1], dtype=float)
31             a = np.sum(np.power(temp, q2_array))
32             b = np.sum(np.power(temp, q_array))
33             img_out[i - step, j - step] = a / b
34
35     img_out = normalize(img_out)
36     return img_out
37

```

## 2. OrderStatisticFilters

### 2.1 median\_filter

```

1  def median_filter(input_image, size):
2      """
3      median filter 对椒盐噪声特别有效，并且不会导致图像边缘变模糊，也不会让图像形状大小改
变。
4      median filter 对均匀噪声无效。
5      :param input_image: 使用opencv读取的输入图像数组
6      :param size: 邻域大小，正方形
7      :return: 输出图像数组
8      """
9      img = np.array(input_image, dtype=float)
10     row, col = img.shape
11     step = (size - 1) // 2
12     if step * 2 + 1 >= row or step * 2 + 1 >= col:
13         print("The parameter size is too large.")
14     img_pad = np.pad(input_image, [step, step], 'constant', constant_values=
[1] * 2)
15     img_out = np.array(np.zeros((row, col)))
16     for i in range(step, row):
17         for j in range(step, col):
18             img_out[i - step, j - step] = np.median(img_pad[i - step:i +
step + 1, j - step:j + step + 1])
19     img_out = normalize(img_out)
20     return img_out

```

## 2.2 max\_filter

```

1  def max_filter(input_image, size):
2      """
3      max filter 适用于处理椒（pepper）噪声，但是它会导致图像中黑色区域变小，白色区域变大
4      :param input_image:
5      :param size:
6      :return:
7      """
8      img = np.array(input_image, dtype=float)
9      row, col = img.shape
10     step = (size - 1) // 2
11     if step * 2 + 1 >= row or step * 2 + 1 >= col:
12         print("The parameter size is too large.")
13     img_pad = np.pad(input_image, [step, step], 'constant', constant_values=
14 [1] * 2)
15     img_out = np.array(np.zeros((row, col)))
16     for i in range(step, row):
17         for j in range(step, col):
18             img_out[i - step, j - step] = np.max(img_pad[i - step:i + step +
19 1, j - step:j + step + 1])
20     img_out = normalize(img_out)
21     return img_out

```

## 2.3 min\_filter

```

1  def min_filter(input_image, size):
2      """
3      min filter 适用于处理盐（salt）噪声，但是它会导致图像中白色区域变小，黑色区域变大
4      :param input_image:
5      :param size:
6      :return:
7      """
8      img = np.array(input_image, dtype=float)
9      row, col = img.shape
10     step = (size - 1) // 2
11     if step * 2 + 1 >= row or step * 2 + 1 >= col:
12         print("The parameter size is too large.")
13     img_pad = np.pad(input_image, [step, step], 'constant', constant_values=
14 [0] * 2)
15     img_out = np.array(np.zeros((row, col)))
16     for i in range(step, row):
17         for j in range(step, col):
18             img_out[i - step, j - step] = np.min(img_pad[i - step:i + step +
19 1, j - step:j + step + 1])
20     return img_out

```

## 2.4 midpoint\_filter

```

1  def midpoint_filter(input_image, size):
2      img = np.array(input_image, dtype=float)
3      row, col = img.shape
4      step = (size - 1) // 2
5      if step * 2 + 1 >= row or step * 2 + 1 >= col:
6          print("The parameter size is too large.")
7      img_pad = np.pad(input_image, [step, step], 'constant', constant_values=
8      [1] * 2)
9      img_out = np.array(np.zeros((row, col)))
10     for i in range(step, row):
11         for j in range(step, col):
12             temp_max = np.max(img_pad[i - step:i + step + 1, j - step:j +
13             step + 1])
14             temp_min = np.min(img_pad[i - step:i + step + 1, j - step:j +
15             step + 1])
16             img_out[i - step, j - step] = (temp_max + temp_min) / 2
17     img_out = normalize(img_out)
18     return img_out

```

## 2.5 alpha\_trimmed\_mean

```

1  def alpha_trimmed_mean(input_image, d, size):
2      """
3       $\alpha$ -裁剪均值滤波器
4      修正阿尔法均值滤波器在邻域中，删除 d 个最低灰度值和 d 个最高灰度值，计算剩余像素的算
5      术平均值作为输出结果
6      :param input_image:
7      :param d:
8      :param size:
9      :return:
10     """
11     img = np.array(input_image, dtype=float)
12     row, col = img.shape
13     step = (size - 1) // 2
14     if step * 2 + 1 >= row or step * 2 + 1 >= col:
15         print("The parameter size is too large.")
16     if d >= size ** 2:
17         print("The parameter d is too large.")
18     img_pad = np.pad(input_image, [step, step], 'constant', constant_values=
19     [1] * 2)
20     img_out = np.array(np.zeros((row, col)))
21     for i in range(step, row):
22         for j in range(step, col):
23             temp = np.sort(img_pad[i - step:i + step + 1, j - step:j + step
24             + 1].flat)
25             temp = np.sum(temp[d: size ** 2 - d])
26             img_out[i - step, j - step] = temp / (size ** 2 - d * 2)
27     img_out = normalize(img_out)
28     return img_out

```

## 3. AdaptiveFilters

### 3.1 adaptive\_arithmetic\_mean

```

1  def adaptive_arithmetic_mean(input_image, noise_var, size):
2      """
3      adaptive mean filter 相当于原图像和算术平均滤波的加权平均，权重由方差决定。
4      注意：由于输入图像 == 原图像 + 噪声，因此邻域方差 >= 全局方差。
5      1.全局方差 == 0，输出原图像。
6      2.邻域方差 == 全局方差，输出算术平均。
7      3.邻域方差 >> 全局方差，说明邻域中包含图像的有效信息，输出图像应当接近原图像（全局方
      差较小，接近原图像；全局方差较大，接近算术平均）。
8      :param noise_var:
9      :param input_image:
10     :param size:
11     :return:
12     """
13     img = np.array(input_image, dtype=int)
14     row, col = img.shape
15     img_out = np.array(np.zeros((row, col)))
16
17     step = (size - 1) // 2
18     if step * 2 + 1 >= row or step * 2 + 1 >= col:
19         print("The parameter size is to large.")
20
21     if noise_var == 0:
22         img_out = img
23     else:
24         img_pad = np.pad(input_image, [step, step], 'constant',
25         constant_values=[0] * 2)
26         for i in range(step, row):
27             for j in range(step, col):
28                 temp = img_pad[i - step:i + step, j - step:j + step]
29                 temp_var = np.var(temp)
30                 if noise_var == temp_var:
31                     img_out[i - step, j - step] = np.mean(temp)
32                 elif temp_var == 0:
33                     img_out[i - step, j - step] = img[i - step, j - step]
34                 else:
35                     rat = noise_var / temp_var
36                     val = img[i - step, j - step]
37                     img_out[i - step, j - step] = (val - rat * (val -
38 np.mean(temp)))
39
40     img_out = normalize(img_out)
41     return img_out

```

### 3.2 adaptive\_median\_filter

```

1  def adaptive_median_filter(input_image, smax, smin):
2      """
3      adaptive median filter 适用于椒盐噪声，可以尽可能确保输出值不是脉冲

```

```

4      1.  $a1 > 0$  and  $a2 < 0$ :
5          通过比较邻域内中值和最大值、最小值的关系判断中值是不是脉冲；
6          如果条件满足，说明不是脉冲，goto State B；
7          如果是脉冲，增加窗口大小；
8          如果窗口增加到最大，中值还是一个脉冲，那么直接输出中值
9      2.  $b1 > 0$  and  $b2 < 0$ :
10         通过比较原图像素点和邻域最大值、最小值的关系判断正在处理的点是不是脉冲
11         如果条件满足，说明不是脉冲，输出原像素点的灰度值
12         如果是脉冲，输出邻域中值（不是脉冲），相当于中值滤波
13     :param input_image:
14     :param smax:窗口最大值
15     :param smin:窗口初始值
16     :return:
17     """
18     img = np.array(input_image, dtype=int)
19     row, col = img.shape
20     img_out = np.array(np.zeros((row, col)))
21
22     for i in range(row):
23         for j in range(col):
24
25             s = smin
26             temp, zmed, zmax, zmin, zxy, a1, a2, b1, b2 =
_adaptive_median_mask(s, img, i, j)
27             while temp is not None:
28
29                 # if  $A1 > 0$  and  $A2 < 0$ , go to stage B
30                 if  $a1 > 0$  and  $a2 < 0$ :
31                     # temp, zmed, zmax, zmin, zxy, __, __, b1, b2 =
_adaptive_median_mask(s - 2, img, i, j)
32                     # if  $A1 > 0$  and  $A2 < 0$ , output zxy
33                     if  $b1 > 0$  and  $b2 < 0$ :
34                         img_out[i, j] = zxy
35                         break
36                     # else output zmed
37                 else:
38                     img_out[i, j] = zmed
39                     break
40                 # else increase the window size
41             else:
42                 s += 2
43                 # if window size  $s > smax$ , output zmed
44                 if  $s > smax$ :
45                     img_out[i, j] = zmed
46                     break
47     img_out = normalize(img_out)
48     return img_out

```

## 4. Degradation Filters



## 4.1 full\_inverse

```

1  def full_inverse(input_image, h):
2      """
3      逆滤波：假设没有噪声，只考虑退化函数
4      :param input_image:
5      :param h:
6      :return:
7      """
8      img = np.array(input_image, dtype=float)
9      img_fft = fftshift(fft2(img))
10     img_out_fft = img_fft / h
11     img_out = np.real(ifft2(ifftshift(img_out_fft)))
12     img_out = normalize(img_out)
13     return img_out

```

## 4.2 limit\_inverse

```

1  def limit_inverse(input_image, h, radius):
2      img = np.array(input_image, dtype=float)
3      img_fft = fftshift(fft2(img))
4      row, col = img_fft.shape
5      img_out_fft = np.array(np.zeros(img_fft.shape), dtype=complex)
6      for i in range(1, row + 1):
7          for j in range(1, col + 1):
8              if ((i - row / 2) ** 2 + (j - col / 2) ** 2) < radius ** 2:
9                  img_out_fft[i - 1, j - 1] = img_fft[i - 1, j - 1] / h[i - 1,
10     j - 1]
11     img_out = np.real(ifft2(ifftshift(img_out_fft)))
12     img_out = normalize(img_out)
13     return img_out

```

## 4.3 wiener

```

1  def wiener(input_image, h, k2):
2      """
3      维纳滤波：最小均方误差
4      :param input_image:
5      :param h:
6      :param k2:
7      :return:
8      """
9      img = np.array(input_image, dtype=float)
10     img_fft = fftshift(fft2(img))
11     h_conj = np.conjugate(h)
12     h2 = np.multiply(h_conj, h)
13     img_out_fft = img_fft * h2 / (h * (h2 + k2))
14     img_out = np.real(ifft2(ifftshift(img_out_fft)))
15     img_out = normalize(img_out)
16     return img_out

```

## 5. Degradation Functions

### 5.1 turbulence

```

1  def turbulence(input_image, k):
2      """
3      大气湍流的退化函数
4      :param input_image:
5      :param k:
6      :return:
7      """
8      img = np.array(input_image, dtype=float)
9      # img = to_center(img)
10     img_fft = fftshift(fft2(img))
11     row, col = img_fft.shape
12     u, v = np.meshgrid(np.linspace(0, row - 1, row), np.linspace(0, col - 1,
13     col))
14     u = u - row / 2
15     v = v - col / 2
16     d = np.power(u, 2) + np.power(v, 2)
17     h = np.exp(-(k * (np.power(d, 5 / 6))))
18     return h

```

### 5.2 motion\_blur

```

1  def motion_blur(input_image, a, b, T):
2      """
3      相机运动模糊的退化函数
4      :param input_image:
5      :param a:
6      :param b:
7      :param T:
8      :return:
9      """
10     img = np.array(input_image, dtype=float)
11     # img = to_center(img)
12     img_fft = fftshift(fft2(img))
13     row, col = img_fft.shape
14     u, v = np.meshgrid(np.linspace(1, row, row), np.linspace(1, col, col))
15     d = pi * (u * a + v * b)
16     e = np.exp(-1j * d)
17     t = np.full([row, col], T)
18     h = t * sin(d) * e / d
19     return h

```

## 6. Top

## 6.1 lab6\_1

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  # @FileName   :lab6_1.py
4  # @Time      :2022-04-23 18:33
5  # @Author    :钟新宇
6  import cv2
7  import numpy as np
8  from matplotlib import pyplot as plt
9
10 from EE326_library.Base import plot
11 from EE326_library.AdaptiveFilters import adaptive_arithmetic_mean,
    adaptive_median_filter
12 from EE326_library.MeanFilters import arithmetic_mean, geometric_mean,
    harmonic_mean, contraharmonic_mean
13 from EE326_library.OrderStatisticFilters import median_filter, max_filter,
    min_filter, midpoint_filter, \
14     alpha_trimmed_mean
15
16
17 def lab6_1(img, size, path, q, noise_var, d, smax):
18     img = np.maximum(img, 1)
19     img_arithmetic_mean = arithmetic_mean(img, size=size)
20     img_geometric_mean = geometric_mean(img, size=size)
21     img_harmonic_mean = harmonic_mean(img, size=size)
22     img_contraharmonic_mean_1 = contraharmonic_mean(img, size=size, q=q)
23     img_contraharmonic_mean_2 = contraharmonic_mean(img, size=size, q=-q)
24     img_median_filter = median_filter(img, size=size)
25     img_max_filter = max_filter(img, size=size)
26     img_min_filter = min_filter(img, size=size)
27     img_midpoint_filter = midpoint_filter(img, size=size)
28     img_alpha_trimmed_mean = alpha_trimmed_mean(img, d=d, size=size)
29     img_adaptive_arithmetic_mean = adaptive_arithmetic_mean(img, size=size,
    noise_var=noise_var)
30     img_adaptive_median_filter = adaptive_median_filter(img, smax=smax,
    smin=1)
31
32     plot(img=img, title="img", path="./img_result/" + path + "/img.png")
33     plot(img=img_arithmetic_mean, title="img_arithmetic_mean",
    path="./img_result/" + path + "/img_arithmetic_mean.png")
34     plot(img=img_geometric_mean, title="img_geometric_mean",
    path="./img_result/" + path + "/img_geometric_mean.png")
35     plot(img=img_harmonic_mean, title="img_harmonic_mean",
    path="./img_result/" + path + "/img_harmonic_mean.png")
36     plot(img=img_contraharmonic_mean_1, title="img_contraharmonic_mean_1" +
    "\n" + "q=%f" % q,
37         path="./img_result/" + path + "/img_contraharmonic_mean_1.png")
38     plot(img=img_contraharmonic_mean_2, title="img_contraharmonic_mean_1" +
    "\n" + "q=%f" % q,
39         path="./img_result/" + path + "/img_contraharmonic_mean_2.png")
40     plot(img=img_median_filter, title="img_median_filter",
    path="./img_result/" + path + "/img_median_filter.png")
41     plot(img=img_max_filter, title="img_max_filter", path="./img_result/" +
    path + "/img_max_filter.png")

```

```

42     plot(img=img_min_filter, title="img_min_filter", path="./img_result/" +
path + "/img_min_filter.png")
43     plot(img=img_midpoint_filter, title="img_midpoint_filter",
path="./img_result/" + path + "/img_midpoint_filter.png")
44     plot(img=img_alpha_trimmed_mean, title="img_alpha_trimmed_mean" + "\n" +
"d=%d" % d,
45         path="./img_result/" + path + "/img_alpha_trimmed_mean.png")
46     plot(img=img_adaptive_arithmetic_mean,
title="img_adaptive_arithmetic_mean" + "\n" + "noise var=%f" % noise_var,
47         path="./img_result/" + path + "/img_adaptive_arithmetic_mean.png")
48     plot(img=img_adaptive_median_filter,
title="img_adaptive_median_filter""\n" + "smax=%d" % smax,
49         path="./img_result/" + path + "/img_adaptive_median_filter.png")
50     plot(img=img_midpoint_filter, title="img_midpoint_filter",
path="./img_result/" + path + "/img_midpoint_filter.png")
51
52
53 def lab6_1_1():
54     """
55     pepper noise; FIGURE 5.8; Page = 325
56     :return:
57     """
58     img = np.asarray(cv2.imread("./img_source/Q6_1_1.tiff",
cv2.IMREAD_GRAYSCALE), dtype=int)
59     lab6_1(img, size=3, path="lab6_1_1", q=1.5, noise_var=0.1, d=2, smax=7)
60
61
62 def lab6_1_2():
63     """
64     salt noise; FIGURE 5.8; Page = 325
65     :return:
66     """
67     img = np.asarray(cv2.imread("./img_source/Q6_1_2.tiff",
cv2.IMREAD_GRAYSCALE), dtype=int)
68     lab6_1(img, size=3, path="lab6_1_2", q=1.5, noise_var=0.1, d=2, smax=7)
69
70
71 def lab6_1_3():
72     """
73     pepper and salt noise; FIGURE 5.12; Page = 329
74     :return:
75     """
76     img = np.asarray(cv2.imread("./img_source/Q6_1_3.tiff",
cv2.IMREAD_GRAYSCALE), dtype=int)
77     lab6_1(img, size=5, path="lab6_1_3", q=1.5, noise_var=0.25, d=2, smax=7)
78
79
80 def lab6_1_4():
81     """
82     uniform noise and pepper noise and salt noise;
83     FIGURE 5.14; Page = 334
84     :return:
85     """
86     img = np.asarray(cv2.imread("./img_source/Q6_1_4.tiff",
cv2.IMREAD_GRAYSCALE), dtype=int)
87     lab6_1(img, size=7, path="lab6_1_4", q=1.5, noise_var=0.25, d=2, smax=7)
88
89

```

```

90 if __name__ == '__main__':
91     try:
92         lab6_1_1()
93         lab6_1_2()
94         lab6_1_3()
95         lab6_1_4()
96     except KeyboardInterrupt:
97         pass
98

```

## 6.2 lab6\_2

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  # @FileName :lab6_2.py
4  # @Time :2022-04-23 20:40
5  # @Author :钟新宇
6  import cv2
7  import numpy as np
8
9  from EE326_library.Degradations import full_inverse, limit_inverse, wiener,
turbulence
10 from EE326_library.Base import normalize, plot
11
12
13 def lab6_full_inverse(img, path, k):
14     h_1 = turbulence(img, k=k[0])
15     h_2 = turbulence(img, k=k[1])
16     h_3 = turbulence(img, k=k[2])
17     h_4 = turbulence(img, k=k[3])
18     img_full_inverse_1 = full_inverse(img, h=h_1)
19     img_full_inverse_2 = full_inverse(img, h=h_2)
20     img_full_inverse_3 = full_inverse(img, h=h_3)
21     img_full_inverse_4 = full_inverse(img, h=h_4)
22
23     plot(img=img_full_inverse_1, title="k=%f" % k[0], path="./img_result/" +
path + "/img_full_inverse_1.png")
24     plot(img=img_full_inverse_2, title="k=%f" % k[1], path="./img_result/" +
path + "/img_full_inverse_2.png")
25     plot(img=img_full_inverse_3, title="k=%f" % k[2], path="./img_result/" +
path + "/img_full_inverse_3.png")
26     plot(img=img_full_inverse_4, title="k=%f" % k[3], path="./img_result/" +
path + "/img_full_inverse_4.png")
27
28
29 def lab6_limit_inverse(img, path, k, radius):
30     h = turbulence(img, k=k)
31     img_limit_inverse_1 = limit_inverse(img, h=h, radius=radius[0])
32     img_limit_inverse_2 = limit_inverse(img, h=h, radius=radius[1])
33     img_limit_inverse_3 = limit_inverse(img, h=h, radius=radius[2])
34     img_limit_inverse_4 = limit_inverse(img, h=h, radius=radius[3])
35
36     plot(img=img_limit_inverse_1, title="radius=%f" % radius[0],

```

```

37     path="./img_result/" + path + "/img_limit_inverse_1.png")
38     plot(img=img_limit_inverse_2, title="radius=%f" % radius[1],
39         path="./img_result/" + path + "/img_limit_inverse_2.png")
40     plot(img=img_limit_inverse_3, title="radius=%f" % radius[2],
41         path="./img_result/" + path + "/img_limit_inverse_3.png")
42     plot(img=img_limit_inverse_4, title="radius=%f" % radius[3],
43         path="./img_result/" + path + "/img_limit_inverse_4.png")
44
45
46 def lab6_wiener(img, path, k, k2):
47     h = turbulence(img, k=k)
48     img_wiener_1 = wiener(img, h=k, k2=k2[0])
49     img_wiener_2 = wiener(img, h=h, k2=k2[1])
50     img_wiener_3 = wiener(img, h=h, k2=k2[2])
51     img_wiener_4 = wiener(img, h=h, k2=k2[3])
52
53     plot(img=img_wiener_1, title="k2=%f" % k2[0], path="./img_result/" +
54         path + "/img_wiener_1.png")
55     plot(img=img_wiener_2, title="k2=%f" % k2[1], path="./img_result/" +
56         path + "/img_wiener_2.png")
57     plot(img=img_wiener_3, title="k2=%f" % k2[2], path="./img_result/" +
58         path + "/img_wiener_3.png")
59     plot(img=img_wiener_4, title="k2=%f" % k2[3], path="./img_result/" +
60         path + "/img_wiener_4.png")
61
62
63 def lab6_add_gaussian_noise(img, path, sigma):
64     img = np.array(img, dtype=float)
65     row, col = img.shape
66     noise = np.random.normal(0, sigma, (row, col))
67     img_out = img + noise
68     img_out = normalize(img_out)
69
70     plot(img=img, title="img", path="./img_result/" + path + "/img.png")
71     plot(img=img_out, title="img_with_gaussian", path="./img_result/" + path
72         + "/img_with_gaussian.png")
73     return img_out
74
75
76 if __name__ == '__main__':
77     try:
78         img = cv2.imread("./img_source/Q6_2.tif", cv2.IMREAD_GRAYSCALE)
79         path = "lab6_2"
80         img = lab6_add_gaussian_noise(img=img, path=path, sigma=0.0065)
81         lab6_full_inverse(img=img, path=path, k=np.array([2.5e-3, 1e-3,
82             2.5e-4, 1e-4], dtype=float))
83         lab6_limit_inverse(img=img, path=path, k=2.5e-4,
84             radius=np.array([40, 80, 120, 160], dtype=float))
85         lab6_wiener(img=img, path=path, k=2.5e-4, k2=np.array([1e-20, 1e-15,
86             1e-10, 1e-5], dtype=float))
87         pass
88     except KeyboardInterrupt:
89         pass
90

```

## 6.3 lab6\_3

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  # @FileName :lab6_3.py
4  # @Time :2022-04-23 18:33
5  # @Author :钟新宇
6  import cv2
7  import matplotlib.pyplot as plt
8  import numpy as np
9  from PIL import Image
10
11 from EE326_library.Base import plot
12 from EE326_library.Degradations import motion_blur, full_inverse,
13 limit_inverse, wiener
14 from lab6.lab6_1 import lab6_1
15
16 def lab6_motion_blur(img, path, a, b, T, mode, radius=70, k2=100):
17     h = motion_blur(img, a=a, b=b, T=T)
18     if mode == "full":
19         img_motion_blur_full = full_inverse(img, h=h)
20         plot(img=img_motion_blur_full, title="img_motion_blur_full",
21              path="./img_result/" + path + "/img_motion_blur_full.png")
22     elif mode == "limit":
23         img_motion_blur_limit = limit_inverse(img, h=h, radius=radius)
24         plot(img=img_motion_blur_limit, title="img_motion_blur_limit",
25              path="./img_result/" + path + "/img_motion_blur_limit.png")
26     elif mode == "wiener":
27         img_motion_blur_wiener = wiener(img, h=h, k2=k2)
28         plot(img=img_motion_blur_wiener, title="img_motion_blur_wiener",
29              path="./img_result/" + path + "/img_motion_blur_wiener.png")
30
31
32 def lab6_3_1():
33     """
34     no noise
35     :return:
36     """
37     img = np.asarray(cv2.imread("./img_source/Q6_3_1.tiff",
38 cv2.IMREAD_GRAYSCALE), dtype=int)
39     # img = plt.imread("./img_source/Q6_3_1.tiff")
40     img = np.array(img, dtype=int)
41     # plt.figure()
42     # plt.imshow(img, cmap='gray')
43     # plt.show()
44     lab6_motion_blur(img=img, path="lab6_3_1", a=0.1, b=0.1, T=1,
45 mode="full")
46     lab6_motion_blur(img=img, path="lab6_3_1", a=0.1, b=0.1, T=1,
47 mode="limit", radius=40)
48     lab6_motion_blur(img=img, path="lab6_3_1", a=0.1, b=0.1, T=1,
49 mode="wiener", k2=100)
50
51
52 def lab6_3_2():
53     """

```

```
50     uniform noise;
51     :return:
52     """
53     img = np.asarray(cv2.imread("./img_source/Q6_3_2.tiff",
54 cv2.IMREAD_GRAYSCALE), dtype=int)
55     lab6_1(img, size=3, path="lab6_3_2", q=1.5, noise_var=0.1, d=2, smax=7)
56
57 def lab6_3_3():
58     """
59     pepper and salt noise;
60     :return:
61     """
62     img = np.asarray(cv2.imread("./img_source/Q6_3_3.tiff",
63 cv2.IMREAD_GRAYSCALE), dtype=int)
64     lab6_1(img, size=5, path="lab6_3_3", q=1.5, noise_var=0.25, d=2, smax=7)
65
66 if __name__ == '__main__':
67     try:
68         # lab6_3_1()
69         lab6_3_2()
70         # lab6_3_3()
71
72     except KeyboardInterrupt:
73         pass
74
```