SUSTech_EE326_lab6_Appendix

Topic: Image Restoration

Author: 11911521钟新宇

Project: lab6 report for Digital Image Processing

Outline

1. MeanFilters

1.1 arithmetic_mean

```python
def arithmetic_mean(input_image, size):
    img = np.array(input_image, dtype=int)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")
    img_pad = np.pad(input_image, [step, step], 'constant', constant_values=[0] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            img_out[i - step, j - step] = np.sum(img_pad[i - step:i + step + 1, j - step:j + step + 1]) / (step ** 2)
    img_out = normalize(img_out)
    return img_out
```

1.2 geometric_mean

```python
def geometric_mean(input_image, size):
    """
    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。
    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。
    :param input_image:
    :param size:
    :return:
    """
    img = np.array(input_image, dtype=float)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")
    img_pad = np.pad(img, [step, step], 'constant', constant_values=[1] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            temp = img_pad[i - step:i + step + 1, j - step:j + step + 1]
            temp = np.prod(temp)
            temp = np.power(temp, 1 / (size ** 2))
            img_out[i - step, j - step] = temp
    img_out = normalize(img_out)
    return img_out
```

1.3 harmonic_mean

```python
def harmonic_mean(input_image, size):
    """
    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。
    :param input_image:
    :param size:
    :return:
    """
    img = np.array(input_image, dtype=int)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")

    img_pad = np.pad(img, [step, step], 'constant', constant_values=[1] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            temp = np.array(img_pad[i - step:i + step + 1, j - step:j + step + 1], dtype=float)
            temp = np.reciprocal(temp)
            temp = (size ** 2) / np.sum(temp)
            img_out[i - step, j - step] = temp
    img_out = normalize(img_out)
    return img_out
```

1.4 contraharmonic_mean

```python
def contraharmonic_mean(input_image, q, size):
    """
    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。
    缺点：不能同时处理椒和盐的噪声；

    :param input_image:
    :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，黑色区域放大。
    :param size:
    :return:
    """
    global q2_array, q_array
    img = np.array(input_image, dtype=float)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")

    img_pad = np.pad(img, [step, step], 'constant', constant_values=[1] * 2)
    img_out = np.array(np.zeros((row, col)))

    if q > 0:
        q_array = np.array(np.maximum(np.zeros((size, size)), q), dtype=float)
        q2_array = np.array(np.maximum(np.zeros((size, size)), q + 1), dtype=float)
    elif q < 0:
        q_array = np.array(np.minimum(np.zeros((size, size)), q), dtype=float)
        q2_array = np.array(np.minimum(np.zeros((size, size)), q + 1), dtype=float)

    for i in range(step, row):
        for j in range(step, col):
            temp = np.array(img_pad[i - step:i + step + 1, j - step:j + step + 1], dtype=float)
            a = np.sum(np.power(temp, q2_array))
            b = np.sum(np.power(temp, q_array))
            img_out[i - step, j - step] = a / b

    img_out = normalize(img_out)
    return img_out

```

2. OrderStatisticFilters

2.1 median_filter

```python
def median_filter(input_image, size):
    """
    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。
    median filter 对均匀噪声无效。
    :param input_image: 使用opencv读取的输入图像数组
    :param size: 邻域大小，正方形
    :return: 输出图像数组
    """
    img = np.array(input_image, dtype=float)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")
    img_pad = np.pad(input_image, [step, step], 'constant', constant_values=[1] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            img_out[i - step, j - step] = np.median(img_pad[i - step:i + step + 1, j - step:j + step + 1])
    img_out = normalize(img_out)
    return img_out
```

2.2 max_filter

```python
def max_filter(input_image, size):
    """
    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
    :param input_image:
    :param size:
    :return:
    """
    img = np.array(input_image, dtype=float)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")
    img_pad = np.pad(input_image, [step, step], 'constant', constant_values=[1] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            img_out[i - step, j - step] = np.max(img_pad[i - step:i + step + 1, j - step:j + step + 1])
    img_out = normalize(img_out)
    return img_out
```

2.3 min_filter

```python
def min_filter(input_image, size):
    """
    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
    :param input_image:
    :param size:
    :return:
    """
    img = np.array(input_image, dtype=float)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")
    img_pad = np.pad(input_image, [step, step], 'constant', constant_values=[0] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            img_out[i - step, j - step] = np.min(img_pad[i - step:i + step + 1, j - step:j + step + 1])
    return img_out
```

2.4 midpoint_filter

```python
def midpoint_filter(input_image, size):
    img = np.array(input_image, dtype=float)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")
    img_pad = np.pad(input_image, [step, step], 'constant', constant_values=[1] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            temp_max = np.max(img_pad[i - step:i + step + 1, j - step:j + step + 1])
            temp_min = np.min(img_pad[i - step:i + step + 1, j - step:j + step + 1])
            img_out[i - step, j - step] = (temp_max + temp_min) / 2
    img_out = normalize(img_out)
    return img_out
```

2.5 alpha_trimmed_mean

```python
def alpha_trimmed_mean(input_image, d, size):
    """
    α—裁剪均值滤波器
    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，计算剩余像素的算术平均值作为输出结果
    :param input_image:
    :param d:
    :param size:
    :return:
    """
    img = np.array(input_image, dtype=float)
    row, col = img.shape
    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")
    if d >= size ** 2:
        print("The parameter d is to large.")
    img_pad = np.pad(input_image, [step, step], 'constant', constant_values=[1] * 2)
    img_out = np.array(np.zeros((row, col)))
    for i in range(step, row):
        for j in range(step, col):
            temp = np.sort(img_pad[i - step:i + step + 1, j - step:j + step + 1].flat)
            temp = np.sum(temp[d: size ** 2 - d])
            img_out[i - step, j - step] = temp / (size ** 2 - d * 2)
    img_out = normalize(img_out)
    return img_out
```

3. AdaptiveFilters

3.1 adaptive_arithmetic_mean

```python
def adaptive_arithmetic_mean(input_image, noise_var, size):
    """
    adaptive mean filter 相当于原图像和算数平均滤波的加权平均，权重由方差决定。
    注意：由于输入图像 == 原图像 + 噪声，因此邻域方差 >= 全局方差。
    1.全局方差 == 0，输出原图像。
    2.邻域方差 == 全局方差，输出算术平均。
    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）。
    :param noise_var:
    :param input_image:
    :param size:
    :return:
    """
    img = np.array(input_image, dtype=int)
    row, col = img.shape
    img_out = np.array(np.zeros((row, col)))

    step = (size - 1) // 2
    if step * 2 + 1 >= row or step * 2 + 1 >= col:
        print("The parameter size is to large.")

    if noise_var == 0:
        img_out = img
    else:
        img_pad = np.pad(input_image, [step, step], 'constant', constant_values=[0] * 2)
        for i in range(step, row):
            for j in range(step, col):
                temp = img_pad[i - step:i + step, j - step:j + step]
                temp_var = np.var(temp)
                if noise_var == temp_var:
                    img_out[i - step, j - step] = np.mean(temp)
                elif temp_var == 0:
                    img_out[i - step, j - step] = img[i - step, j - step]
                else:
                    rat = noise_var / temp_var
                    val = img[i - step, j - step]
                    img_out[i - step, j - step] = (val - rat * (val - np.mean(temp)))

    img_out = normalize(img_out)
    return img_out
```

3.2 adaptive_median_filter

```python
def adaptive_median_filter(input_image, smax, smin):
    """
        adaptive median filter 适用于椒盐噪声，可以尽可能确保输出值不是脉冲
        1. a1 > 0 and a2 < 0:
            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;
            如果条件满足，说明不是脉冲，goto State B;
            如果是脉冲，增加窗口大小;
            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
        2.b1 > 0 and b2 < 0:
            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
            如果条件满足，说明不是脉冲，输出原像素点的灰度值
            如果是脉冲，输出邻域中值（不是脉冲），相当于中值滤波
    :param input_image:
    :param smax:窗口最大值
    :param smin:窗口初始值
    :return:
    """
    img = np.array(input_image, dtype=int)
    row, col = img.shape
    img_out = np.array(np.zeros((row, col)))

    for i in range(row):
        for j in range(col):

            s = smin
            temp, zmed, zmax, zmin, zxy, a1, a2, b1, b2 = _adaptive_median_mask(s, img, i, j)
            while temp is not None:

                # if A1>0 and A2<0, go to stage B
                if a1 > 0 and a2 < 0:
                    # temp, zmed, zmax, zmin, zxy, _, _, b1, b2 = adaptive_median_mask(s - 2, img, i, j)
                    # if A1>0 and A2<0, output zxy
                    if b1 > 0 and b2 < 0:
                        img_out[i, j] = zxy
                        break
                    # else output zmed
                    else:
                        img_out[i, j] = zmed
                        break
                # else increase the window size
                else:
                    s += 2
                    # if window size s > smax, output zmed
                    if s > smax:
                        img_out[i, j] = zmed
                        break
    img_out = normalize(img_out)
    return img_out
```

4. Degradation Filters

4.1 full_inverse

```python
def full_inverse(input_image, h):
    """
    逆滤波：假设没有噪声，只考虑退化函数
    :param input_image:
    :param h:
    :return:
    """
    img = np.array(input_image, dtype=float)
    img_fft = fftshift(fft2(img))
    img_out_fft = img_fft / h
    img_out = np.real(ifft2(ifftshift(img_out_fft)))
    img_out = normalize(img_out)
    return img_out
```

4.2 limit_inverse

```python
def limit_inverse(input_image, h, radius):
    img = np.array(input_image, dtype=float)
    img_fft = fftshift(fft2(img))
    row, col = img_fft.shape
    img_out_fft = np.array(np.zeros(img_fft.shape), dtype=complex)
    for i in range(1, row + 1):
        for j in range(1, col + 1):
            if ((i - row / 2) ** 2 + (j - col / 2) ** 2) < radius ** 2:
                img_out_fft[i - 1, j - 1] = img_fft[i - 1, j - 1] / h[i - 1, j - 1]
    img_out = np.real(ifft2(ifftshift(img_out_fft)))
    img_out = normalize(img_out)
    return img_out
```

4.3 wiener

```python
def wiener(input_image, h, k2):
    """
    维纳滤波：最小均方误差
    :param input_image:
    :param h:
    :param k2:
    :return:
    """
    img = np.array(input_image, dtype=float)
    img_fft = fftshift(fft2(img))
    h_conj = np.conjugate(h)
    h2 = np.multiply(h_conj, h)
    img_out_fft = img_fft * h2 / (h * (h2 + k2))
    img_out = np.real(ifft2(ifftshift(img_out_fft)))
    img_out = normalize(img_out)
    return img_out
```

5. Degradation Functions

5.1 turbulence

```python
def turbulence(input_image, k):
    """
    大气湍流的退化函数
    :param input_image:
    :param k:
    :return:
    """
    img = np.array(input_image, dtype=float)
    # img = to_center(img)
    img_fft = fftshift(fft2(img))
    row, col = img_fft.shape
    u, v = np.meshgrid(np.linspace(0, row - 1, row), np.linspace(0, col - 1, col))
    u = u - row / 2
    v = v - col / 2
    d = np.power(u, 2) + np.power(v, 2)
    h = np.exp(-(k * (np.power(d, 5 / 6))))
    return h
```

5.2 motion_blur

```python
def motion_blur(input_image, a, b, T):
    """
    相机运动模糊的退化函数
    :param input_image:
    :param a:
    :param b:
    :param T:
    :return:
    """
    img = np.array(input_image, dtype=float)
    # img = to_center(img)
    img_fft = fftshift(fft2(img))
    row, col = img_fft.shape
    u, v = np.meshgrid(np.linspace(1, row, row), np.linspace(1, col, col))
    d = pi * (u * a + v * b)
    e = np.exp(-1j * d)
    t = np.full([row, col], T)
    h = t * sin(d) * e / d
    return h
```

6. Top

6.1 lab6_1

```python
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :lab6_1.py
# @Time      :2022-04-23 18:33
# @Author    :钟新宇
import cv2
import numpy as np
from matplotlib import pyplot as plt

from EE326_library.Base import plot
from EE326_library.AdaptiveFilters import adaptive_arithmetic_mean, adaptive_median_filter
from EE326_library.MeanFilters import arithmetic_mean, geometric_mean, harmonic_mean, contraharmonic_mean
from EE326_library.OrderStatisticFilters import median_filter, max_filter, min_filter, midpoint_filter, \
    alpha_trimmed_mean


def lab6_1(img, size, path, q, noise_var, d, smax):
    img = np.maximum(img, 1)
    img_arithmetic_mean = arithmetic_mean(img, size=size)
    img_geometric_mean = geometric_mean(img, size=size)
    img_harmonic_mean = harmonic_mean(img, size=size)
    img_contraharmonic_mean_1 = contraharmonic_mean(img, size=size, q=q)
    img_contraharmonic_mean_2 = contraharmonic_mean(img, size=size, q=-q)
    img_median_filter = median_filter(img, size=size)
    img_max_filter = max_filter(img, size=size)
    img_min_filter = min_filter(img, size=size)
    img_midpoint_filter = midpoint_filter(img, size=size)
    img_alpha_trimmed_mean = alpha_trimmed_mean(img, d=d, size=size)
    img_adaptive_arithmetic_mean = adaptive_arithmetic_mean(img, size=size, noise_var=noise_var)
    img_adaptive_median_filter = adaptive_median_filter(img, smax=smax, smin=1)

    plot(img=img, title="img", path="./img_result/" + path + "/img.png")
    plot(img=img_arithmetic_mean, title="img_arithmetic_mean", path="./img_result/" + path + "/img_arithmetic_mean.png")
    plot(img=img_geometric_mean, title="img_geometric_mean", path="./img_result/" + path + "/img_geometric_mean.png")
    plot(img=img_harmonic_mean, title="img_harmonic_mean", path="./img_result/" + path + "/img_harmonic_mean.png")
    plot(img=img_contraharmonic_mean_1, title="img_contraharmonic_mean_1" + "\n" + "q=%f" % q,
         path="./img_result/" + path + "/img_contraharmonic_mean_1.png")
    plot(img=img_contraharmonic_mean_2, title="img_contraharmonic_mean_1" + "\n" + "q=%f" % q,
         path="./img_result/" + path + "/img_contraharmonic_mean_2.png")
    plot(img=img_median_filter, title="img_median_filter", path="./img_result/" + path + "/img_median_filter.png")
    plot(img=img_max_filter, title="img_max_filter", path="./img_result/" + path + "/img_max_filter.png")
    plot(img=img_min_filter, title="img_min_filter", path="./img_result/" + path + "/img_min_filter.png")
    plot(img=img_midpoint_filter, title="img_midpoint_filter", path="./img_result/" + path + "/img_midpoint_filter.png")
    plot(img=img_alpha_trimmed_mean, title="img_alpha_trimmed_mean" + "\n" + "d=%d" % d,
         path="./img_result/" + path + "/img_alpha_trimmed_mean.png")
    plot(img=img_adaptive_arithmetic_mean, title="img_adaptive_arithmetic_mean" + "\n" + "noise var=%f" % noise_var,
         path="./img_result/" + path + "/img_adaptive_arithmetic_mean.png")
    plot(img=img_adaptive_median_filter, title="img_adaptive_median_filter""\n" + "smax=%d" % smax,
         path="./img_result/" + path + "/img_adaptive_median_filter.png")
    plot(img=img_midpoint_filter, title="img_midpoint_filter", path="./img_result/" + path + "/img_midpoint_filter.png")


def lab6_1_1():
    """
    pepper noise; FIGURE 5.8; Page = 325
    :return:
    """
    img = np.asarray(cv2.imread("./img_source/Q6_1_1.tiff", cv2.IMREAD_GRAYSCALE), dtype=int)
    lab6_1(img, size=3, path="lab6_1_1", q=1.5, noise_var=0.1, d=2, smax=7)


def lab6_1_2():
    """
    salt noise; FIGURE 5.8; Page = 325
    :return:
    """
    img = np.asarray(cv2.imread("./img_source/Q6_1_2.tiff", cv2.IMREAD_GRAYSCALE), dtype=int)
    lab6_1(img, size=3, path="lab6_1_2", q=1.5, noise_var=0.1, d=2, smax=7)


def lab6_1_3():
    """
    pepper and salt noise; FIGURE 5.12; Page = 329
    :return:
    """
    img = np.asarray(cv2.imread("./img_source/Q6_1_3.tiff", cv2.IMREAD_GRAYSCALE), dtype=int)
    lab6_1(img, size=5, path="lab6_1_3", q=1.5, noise_var=0.25, d=2, smax=7)


def lab6_1_4():
    """
    uniform noise and pepper noise and salt noise;
    FIGURE 5.14; Page = 334
    :return:
    """
    img = np.asarray(cv2.imread("./img_source/Q6_1_4.tiff", cv2.IMREAD_GRAYSCALE), dtype=int)
    lab6_1(img, size=7, path="lab6_1_4", q=1.5, noise_var=0.25, d=2, smax=7)


```

```python
if __name__ == '__main__':
    try:
        lab6_1_1()
        lab6_1_2()
        lab6_1_3()
        lab6_1_4()
    except KeyboardInterrupt:
        pass

```

6.2 lab6_2

```python
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :lab6_2.py
# @Time      :2022-04-23 20:40
# @Author    :钟新宇
import cv2
import numpy as np

from EE326_library.Degradations import full_inverse, limit_inverse, wiener, turbulence
from EE326_library.Base import normalize, plot


def lab6_full_inverse(img, path, k):
    h_1 = turbulence(img, k=k[0])
    h_2 = turbulence(img, k=k[1])
    h_3 = turbulence(img, k=k[2])
    h_4 = turbulence(img, k=k[3])
    img_full_inverse_1 = full_inverse(img, h=h_1)
    img_full_inverse_2 = full_inverse(img, h=h_2)
    img_full_inverse_3 = full_inverse(img, h=h_3)
    img_full_inverse_4 = full_inverse(img, h=h_4)

    plot(img=img_full_inverse_1, title="k=%f" % k[0], path="./img_result/" + path + "/img_full_inverse_1.png")
    plot(img=img_full_inverse_2, title="k=%f" % k[1], path="./img_result/" + path + "/img_full_inverse_2.png")
    plot(img=img_full_inverse_3, title="k=%f" % k[2], path="./img_result/" + path + "/img_full_inverse_3.png")
    plot(img=img_full_inverse_4, title="k=%f" % k[3], path="./img_result/" + path + "/img_full_inverse_4.png")


def lab6_limit_inverse(img, path, k, radius):
    h = turbulence(img, k=k)
    img_limit_inverse_1 = limit_inverse(img, h=h, radius=radius[0])
    img_limit_inverse_2 = limit_inverse(img, h=h, radius=radius[1])
    img_limit_inverse_3 = limit_inverse(img, h=h, radius=radius[2])
    img_limit_inverse_4 = limit_inverse(img, h=h, radius=radius[3])

    plot(img=img_limit_inverse_1, title="radius=%f" % radius[0],
         path="./img_result/" + path + "/img_limit_inverse_1.png")
    plot(img=img_limit_inverse_2, title="radius=%f" % radius[1],
         path="./img_result/" + path + "/img_limit_inverse_2.png")
    plot(img=img_limit_inverse_3, title="radius=%f" % radius[2],
         path="./img_result/" + path + "/img_limit_inverse_3.png")
    plot(img=img_limit_inverse_4, title="radius=%f" % radius[3],
         path="./img_result/" + path + "/img_limit_inverse_4.png")


def lab6_wiener(img, path, k, k2):
    h = turbulence(img, k=k)
    img_wiener_1 = wiener(img, h=k, k2=k2[0])
    img_wiener_2 = wiener(img, h=h, k2=k2[1])
    img_wiener_3 = wiener(img, h=h, k2=k2[2])
    img_wiener_4 = wiener(img, h=h, k2=k2[3])

    plot(img=img_wiener_1, title="k2=%f" % k2[0], path="./img_result/" + path + "/img_wiener_1.png")
    plot(img=img_wiener_2, title="k2=%f" % k2[1], path="./img_result/" + path + "/img_wiener_2.png")
    plot(img=img_wiener_3, title="k2=%f" % k2[2], path="./img_result/" + path + "/img_wiener_3.png")
    plot(img=img_wiener_4, title="k2=%f" % k2[3], path="./img_result/" + path + "/img_wiener_4.png")


def lab6_add_gaussian_noise(img, path, sigma):
    img = np.array(img, dtype=float)
    row, col = img.shape
    noise = np.random.normal(0, sigma, (row, col))
    img_out = img + noise
    img_out = normalize(img_out)

    plot(img=img, title="img", path="./img_result/" + path + "/img.png")
    plot(img=img_out, title="img_with_gaussian", path="./img_result/" + path + "/img_with_gaussian.png")
    return img_out


if __name__ == '__main__':
    try:
        img = cv2.imread("./img_source/Q6_2.tif", cv2.IMREAD_GRAYSCALE)
        path = "lab6_2"
        img = lab6_add_gaussian_noise(img=img, path=path, sigma=0.0065)
        lab6_full_inverse(img=img, path=path, k=np.array([2.5e-3, 1e-3, 2.5e-4, 1e-4], dtype=float))
        lab6_limit_inverse(img=img, path=path, k=2.5e-4, radius=np.array([40, 80, 120, 160], dtype=float))
        lab6_wiener(img=img, path=path, k=2.5e-4, k2=np.array([1e-20, 1e-15, 1e-10, 1e-5], dtype=float))
        pass
    except KeyboardInterrupt:
        pass

```

6.3 lab6_3

```python
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# @FileName  :lab6_3.py
# @Time      :2022-04-23 18:33
# @Author    :钟新宇
import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

from EE326_library.Base import plot
from EE326_library.Degradations import motion_blur, full_inverse, limit_inverse, wiener
from lab6.lab6_1 import lab6_1


def lab6_motion_blur(img, path, a, b, T, mode, radius=70, k2=100):
    h = motion_blur(img, a=a, b=b, T=T)
    if mode == "full":
        img_motion_blur_full = full_inverse(img, h=h)
        plot(img=img_motion_blur_full, title="img_motion_blur_full",
             path="./img_result/" + path + "/img_motion_blur_full.png")
    elif mode == "limit":
        img_motion_blur_limit = limit_inverse(img, h=h, radius=radius)
        plot(img=img_motion_blur_limit, title="img_motion_blur_limit",
             path="./img_result/" + path + "/img_motion_blur_limit.png")
    elif mode == "wiener":
        img_motion_blur_wiener = wiener(img, h=h, k2=k2)
        plot(img=img_motion_blur_wiener, title="img_motion_blur_wiener",
             path="./img_result/" + path + "/img_motion_blur_wiener.png")


def lab6_3_1():
    """
    no noise
    :return:
    """
    img = np.asarray(cv2.imread("./img_source/Q6_3_1.tiff", cv2.IMREAD_GRAYSCALE), dtype=int)
    # img = plt.imread("./img_source/Q6_3_1.tiff")
    img = np.array(img, dtype=int)
    # plt.figure()
    # plt.imshow(img, cmap='gray')
    # plt.show()
    lab6_motion_blur(img=img, path="lab6_3_1", a=0.1, b=0.1, T=1, mode="full")
    lab6_motion_blur(img=img, path="lab6_3_1", a=0.1, b=0.1, T=1, mode="limit", radius=40)
    lab6_motion_blur(img=img, path="lab6_3_1", a=0.1, b=0.1, T=1, mode="wiener", k2=100)


def lab6_3_2():
    """
    uniform noise;
    :return:
    """
    img = np.asarray(cv2.imread("./img_source/Q6_3_2.tiff", cv2.IMREAD_GRAYSCALE), dtype=int)
    lab6_1(img, size=3, path="lab6_3_2", q=1.5, noise_var=0.1, d=2, smax=7)


def lab6_3_3():
    """
    pepper and salt noise;
    :return:
    """
    img = np.asarray(cv2.imread("./img_source/Q6_3_3.tiff", cv2.IMREAD_GRAYSCALE), dtype=int)
    lab6_1(img, size=5, path="lab6_3_3", q=1.5, noise_var=0.25, d=2, smax=7)


if __name__ == '__main__':
    try:
        # lab6_3_1()
        lab6_3_2()
        # lab6_3_3()

    except KeyboardInterrupt:
        pass

```