综合实验<三>

学号	姓名	分工
21307035	邓栩瀛	完成思考题
21307056	钟欣余	完成实验三中(1)(2)代码的编写
21307037	金思琪	完成实验三中(3)代码的编写
21307062	郑越	完成实验三中(4)代码的编写

1、问题描述

实验目的

1. 傅里叶变换的二维推广
2. 图像信号的频域表示与分析
3. 图像处理常见任务与滤波器设计

实验要求

(1) 对数字图像进行不同参数下的理想滤波（低通、高通、带通），并分析得到的结果； (2) 设计适当的非理想滤波器（如巴特沃斯滤波器）对相同的图像进行滤波，并与理想滤波结果进行对比分析； (3) 假设图像拍摄时存在运动模糊，之后还叠加了一定程度的高斯随机噪声，试通过编程对这一图像质量退化过程进行仿真； (4) 如果已知上述图像退化过程的参数，请编程实现对已退化图像的复原。

实验思考

如果退化过程的参数未知，如何进行复原？请编程实现，并与参数已知时的结果进行比较。

2、问题分析

实验原理

本次实验版本Python=3.6.5，numpy=1.14.5，scipy=1.1.0，sympy=1.1.1， matplotlib= 3.1.1。 对于本实验Python库约定：

```
# 导入numpy和scipy库
import numpy as np
import cv2
from PIL import ImageEnhance, Image
from numpy.fft import fft2, ifft2
from scipy.signal import convolve2d
import matplotlib.pyplot as plt
```

考查内容

- 数字图像滤波（理想滤波和非理想滤波）的原理和应用。
- 图像质量退化过程的仿真和复原。

相关原理和解决思路：

(1) 理想滤波：

- 低通滤波器：通过保留低频信息而抑制高频信息来平滑图像。可以使用频域中的矩形窗口函数（理想滤波器）。
- 高通滤波器：通过保留高频信息而抑制低频信息来增强图像的细节和边缘。可以使用频域中的矩形窗口函数（理想滤波器）。
- 带通滤波器：通过抑制或保留一定频率范围内的信息来滤波图像。可以使用频域中的矩形窗口函数（理想滤波器）。

(2) 非理想滤波器（以巴特沃斯滤波器为例）：

- 巴特沃斯滤波器是一种常用的非理想滤波器，它具有自定义的截止频率和阶数。
- 首先，根据滤波器类型（低通、高通、带通）和要求的截止频率，选择适当的巴特沃斯滤波器类型和参数。
- 然后，将巴特沃斯滤波器应用于输入图像，可以使用时域或频域方法进行滤波。
- 对比分析巴特沃斯滤波器的滤波效果与理想滤波器的结果。

(3) 图像质量退化过程的仿真：

- 模拟图像拍摄时的运动模糊可以使用模糊核函数进行卷积操作。模糊核函数的选择取决于运动模糊的方向和程度。
- 高斯随机噪声可以通过向图像的每个像素添加一个随机值来模拟。
- 将运动模糊和高斯随机噪声添加到原始图像中，以模拟图像退化过程。

(4) 图像复原：

- 已知图像退化过程的参数，可以尝试使用逆滤波或维纳滤波等图像复原技术。
- 逆滤波是通过将退化过程的逆操作应用于退化图像来尝试恢复原始图像。但逆滤波容易放大噪声。
- 维纳滤波是一种折衷方法，通过平衡逆滤波和噪声增益来进行图像复原。

3、实验代码

1

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def show_images(original, low, high, band):
    plt.figure(figsize=(20, 6))
    plt.subplot(141), plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
    plt.subplot(142), plt.imshow(cv2.cvtColor(low, cv2.COLOR_BGR2RGB)),
plt.title('Lowpass Image')
    plt.subplot(143), plt.imshow(cv2.cvtColor(high, cv2.COLOR_BGR2RGB)),
plt.title('Highpass Image')
    plt.subplot(144), plt.imshow(cv2.cvtColor(band, cv2.COLOR_BGR2RGB)),
```

```
plt.title('Bandpass Image')
    plt.show()
def ideal_filter(img, filter_type, radius):
    # 转换为灰度图像
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_shape = gray.shape

    # 傅里叶变换
    f_gray = np.fft.fft2(gray)
    f_gray_shift = np.fft.fftshift(f_gray)

    # 创建空的滤波器
    filter_mask = np.zeros(img_shape)

    # 计算滤波器中心
    center_x, center_y = img_shape[1] // 2, img_shape[0] // 2

    for x in range(img_shape[1]):
        for y in range(img_shape[0]):
            distance = np.sqrt((x - center_x) ** 2 + (y - center_y) ** 2)

            if filter_type == 'lowpass':
                if distance <= radius:
                    filter_mask[y, x] = 1
            elif filter_type == 'highpass':
                if distance >= radius:
                    filter_mask[y, x] = 1
            elif filter_type == 'bandpass':
                if distance >= radius[0] and distance <= radius[1]:
                    filter_mask[y, x] = 1

    # 应用滤波器
    filtered_shift = f_gray_shift * filter_mask
    filtered = np.fft.ifftshift(filtered_shift)
    filtered_img = np.abs(np.fft.ifft2(filtered))

    # 转换为8位整数
    filtered_img = np.uint8(filtered_img)

    return filtered_img


img = cv2.imread('lena.png')

# 低通滤波
lowpass = ideal_filter(img, 'lowpass', 30)

# 高通滤波
highpass = ideal_filter(img, 'highpass', 30)

# 带通滤波
bandpass = ideal_filter(img, 'bandpass', [30, 80])

show_images(img, lowpass,highpass,bandpass)
```

2

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def show_images(original, low, high, band):
    plt.figure(figsize=(20, 6))
    plt.subplot(141), plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
    plt.subplot(142), plt.imshow(cv2.cvtColor(low, cv2.COLOR_BGR2RGB)),
plt.title('Lowpass Image')
    plt.subplot(143), plt.imshow(cv2.cvtColor(high, cv2.COLOR_BGR2RGB)),
plt.title('Highpass Image')
    plt.subplot(144), plt.imshow(cv2.cvtColor(band, cv2.COLOR_BGR2RGB)),
plt.title('Bandpass Image')
    plt.show()


def butterworth_filter(img, filter_type, radius, order):
    # 转换为灰度图像
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_shape = gray.shape

    # 傅里叶变换
    f_gray = np.fft.fft2(gray)
    f_gray_shift = np.fft.fftshift(f_gray)

    # 创建空的滤波器
    filter_mask = np.zeros(img_shape)

    # 计算滤波器中心
    center_x, center_y = img_shape[1] // 2, img_shape[0] // 2

    for x in range(img_shape[1]):
        for y in range(img_shape[0]):
            distance = np.sqrt((x - center_x) ** 2 + (y - center_y) ** 2)

            if filter_type == 'lowpass':
                filter_mask[y, x] = 1 / (1 + (distance / radius) ** (2 * order))
            elif filter_type == 'highpass':
                filter_mask[y, x] = 1 / (1 + (radius / distance) ** (2 * order))
            elif filter_type == 'bandpass':
                low_radius, high_radius = radius
                low_mask = 1 / (1 + (distance / low_radius) ** (2 * order))
                high_mask = 1 / (1 + (high_radius / distance) ** (2 * order))
                filter_mask[y, x] = low_mask * high_mask

    # 应用滤波器
```

```
    filtered_shift = f_gray_shift * filter_mask
    filtered = np.fft.ifftshift(filtered_shift)
    filtered_img = np.abs(np.fft.ifft2(filtered))

    # 转换为8位整数
    filtered_img = np.uint8(filtered_img)

    return filtered_img


img = cv2.imread('lena.png')
# 巴特沃斯低通滤波
lowpass_butter = butterworth_filter(img, 'lowpass', 30, 2)

# 巴特沃斯高通滤波
highpass_butter = butterworth_filter(img, 'highpass', 30, 2)

# 巴特沃斯带通滤波
bandpass_butter = butterworth_filter(img, 'bandpass', [30, 80], 2)

show_images(img, lowpass_butter,highpass_butter,bandpass_butter)
```

3

```
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt


def show_images(original, filtered):
    plt.figure(figsize=(12, 6))
    plt.subplot(121), plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
    plt.subplot(122), plt.imshow(cv2.cvtColor(filtered, cv2.COLOR_BGR2RGB)),
plt.title('Noisy Image')
    plt.show()


def motion_blur(img, size, angle):
    # 创建运动模糊滤波器
    blur_filter = np.ones((size, size))
    blur_filter[int((size - 1) / 2), :] = 0
    blur_filter = cv2.getRotationMatrix2D((size / 2 - 1, size / 2 - 1), angle, 1)
    M = np.eye(size)
    M[:2, :3] = blur_filter
    blur_filter = cv2.warpAffine(M, blur_filter, (size, size))
    blur_filter = blur_filter / np.sum(blur_filter)
    # 应用滤波器
    return cv2.filter2D(img, -1, blur_filter)
```

```
def add_gaussian_noise(img, mean=0, sigma=10):
    # 添加高斯噪声
    noise = np.zeros_like(img)
    cv2.randn(noise, mean, sigma)
    noisy_img = cv2.add(img, noise, dtype=cv2.CV_8UC3)
    return noisy_img


# 读入图像
img = cv2.imread('lena.png')

# 进行运动模糊和高斯噪声退化
motion_blurred = motion_blur(img, 30, 45)
noisy_motion_blurred = add_gaussian_noise(motion_blurred)

# 显示结果
show_images(img, noisy_motion_blurred)
```

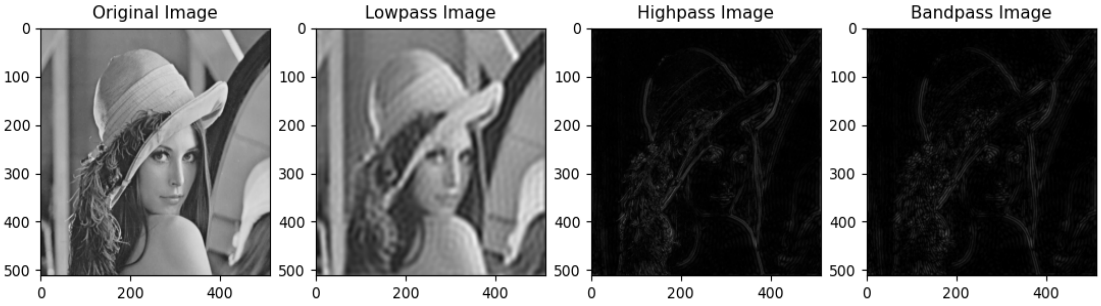
4

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def show_images(original, noisy, inverse, wiener):
    plt.figure(figsize=(20, 6))
    plt.subplot(141), plt.imshow(original, cmap='gray'), plt.title('Original
Image')
    plt.subplot(142), plt.imshow(noisy, cmap='gray'), plt.title('Noisy Image')
    plt.subplot(143), plt.imshow(inverse, cmap='gray'),
plt.title('Inverse_Restored Image')
    plt.subplot(144), plt.imshow(wiener, cmap='gray'), plt.title('Wiener_Restored
Image')
    plt.show()


# 读取图像并转换为灰度图
img = cv2.imread('lena.png')
img = img.astype(np.uint8)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# 生成运动模糊核
kernel_size = 15  # 模糊核的大小
angle = 45  # 模糊核的角度
kernel_motion_blur = np.zeros((kernel_size, kernel_size))
kernel_motion_blur[int((kernel_size - 1) / 2), :] = np.ones(kernel_size)
kernel_motion_blur = cv2.warpAffine(kernel_motion_blur,
                                    cv2.getRotationMatrix2D((kernel_size / 2 -
0.5, kernel_size / 2 - 0.5), angle, 1.0),
```

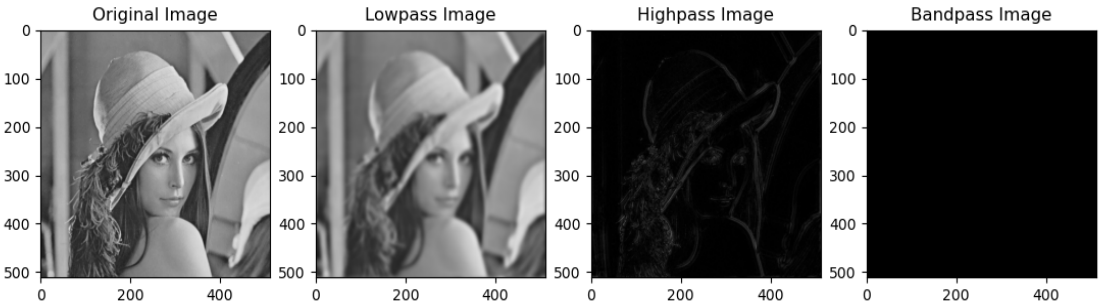
```
                                        (kernel_size, kernel_size))
kernel_motion_blur = kernel_motion_blur / kernel_size

# 对图像进行卷积操作，得到运动模糊后的图像
img_blur = cv2.filter2D(img, -1, kernel_motion_blur)

# 生成高斯随机噪声
noise_mean = 0  # 噪声的均值
noise_var = 0.01  # 噪声的方差
noise = np.random.normal(noise_mean, noise_var ** 0.5, img.shape)

# 加到运动模糊后的图像上，得到最终退化后的图像
img_noisy = img_blur + noise


# 对退化后的图像进行傅里叶变换，并计算其幅度谱和相位谱
img_noisy_fft = np.fft.fft2(img_noisy)
img_noisy_fft_shift = np.fft.fftshift(img_noisy_fft)
img_noisy_magnitude = np.abs(img_noisy_fft_shift)
img_noisy_phase = np.angle(img_noisy_fft_shift)
img_noisy = np.clip(img_noisy, 0, 255).astype(np.uint8)
# 对运动模糊核进行傅里叶变换，并计算其幅度谱和相位谱
kernel_motion_blur_fft = np.fft.fft2(kernel_motion_blur, s=img.shape)
kernel_motion_blur_fft_shift = np.fft.fftshift(kernel_motion_blur_fft)
kernel_motion_blur_magnitude = np.abs(kernel_motion_blur_fft_shift)
kernel_motion_blur_phase = np.angle(kernel_motion_blur_fft_shift)

# 用逆滤波对退化后的图像进行复原，并计算其幅度谱和相位谱
img_inverse_fft_shift = img_noisy_fft_shift / kernel_motion_blur_fft_shift
img_inverse_magnitude = np.abs(img_inverse_fft_shift)
img_inverse_phase = np.angle(img_inverse_fft_shift)

# 用维纳滤波对退化后的图像进行复原，并计算其幅度谱和相位谱
K = 0.02  # 正则参数
img_wiener_fft_shift = kernel_motion_blur_fft_shift.conj() * img_noisy_fft_shift /
(
            np.abs(kernel_motion_blur_fft_shift) ** 2 + K)
img_wiener_magnitude = np.abs(img_wiener_fft_shift)
img_wiener_phase = np.angle(img_wiener_fft_shift)

# 对复原后的图像进行逆傅里叶变换，并转换为灰度图
img_inverse_fft_ishift = np.fft.ifftshift(img_inverse_fft_shift)
img_inverse_restored = np.fft.ifft2(img_inverse_fft_ishift)
img_inverse_restored = np.real(img_inverse_restored)
img_inverse_restored = np.clip(img_inverse_restored, 0, 255).astype(np.uint8)

img_wiener_fft_ishift = np.fft.ifftshift(img_wiener_fft_shift)
img_wiener_restored = np.fft.ifft2(img_wiener_fft_ishift)
img_wiener_restored = np.real(img_wiener_restored)
img_wiener_restored = np.clip(img_wiener_restored, 0, 255).astype(np.uint8)

# 显示原始图像、退化后的图像和复原后的图像
show_images(img,img_noisy,img_inverse_restored,img_wiener_restored)
```

4、实验结果

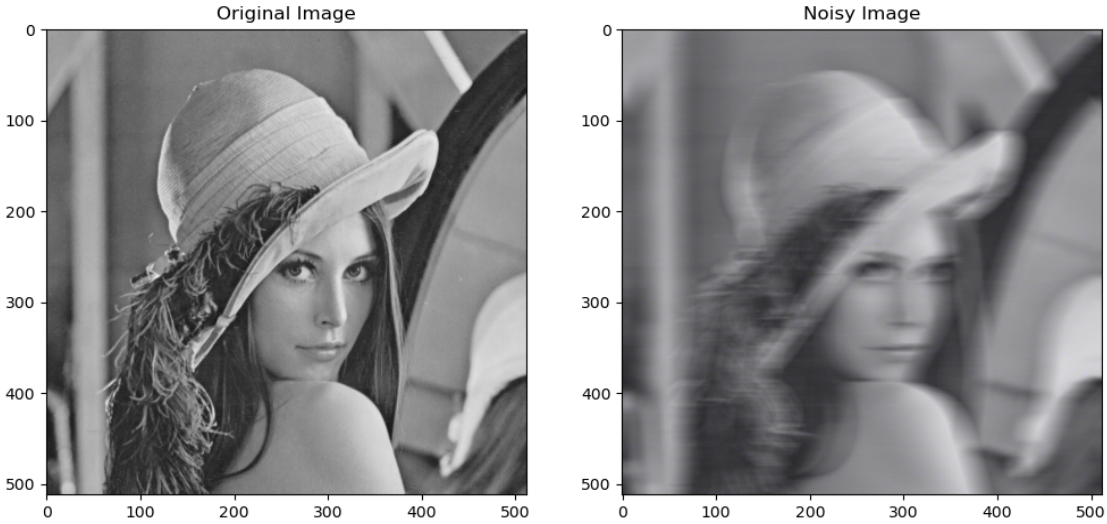
1


Original Image
Lowpass Image
Highpass Image
Bandpass Image

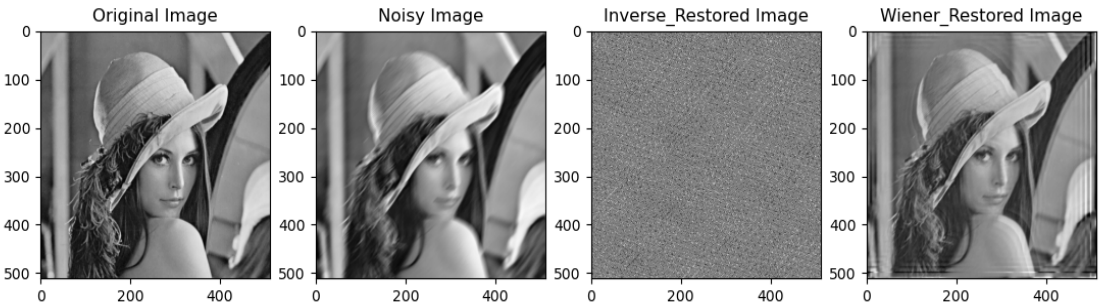
2


Original Image
Lowpass Image
Highpass Image
Bandpass Image

3



4



5、结论

1

- 低通滤波：通过保留图像中低频部分（较慢变化的部分）来模糊或降低图像的细节。这可以用于去除高频噪声或平滑图像。然而，理想低通滤波器产生了称为"振铃"的伪影效应，这意味着在滤波后的图像中

出现了明显的振铃现象。

- 高通滤波：通过去除图像中的低频部分，突出显示高频细节。这可以用于图像增强、边缘检测等应用。理想高通滤波器在频域上产生了明显的振铃效应，这会导致输出图像中出现环状伪影。
- 带通滤波：带通滤波器允许某个特定频率范围内的信号通过，并抑制其他频率范围的信号。这在一些应用中很有用，比如图像增强和信号分析。理想带通滤波器在频域上产生了振铃效应，并且具有较宽的过渡带宽。

总结而言，理想滤波器在频域上具有良好的频率选择性，但在空域上引入了振铃效应，这在实际图像处理中是不可接受的。

2

与理想滤波器相比，巴特沃斯滤波器的滤波结果会产生更平滑的过渡带宽，减少了振铃效应。这使得巴特沃斯滤波器在实际图像处理中更为可靠。

3

对于图像质量退化的仿真，可以通过以下步骤进行：

- 生成运动模糊图像：使用运动模糊模型生成运动模糊图像。运动模糊模型通常基于运动物体在曝光期间对图像的影响，可以通过应用运动模糊核对原始图像进行卷积得到模糊图像。
- 添加高斯随机噪声：对模糊图像添加一定程度的高斯随机噪声。高斯噪声是一种常见的噪声类型，可以通过在每个像素上添加服从正态分布的随机值来模拟。
- 图像退化仿真结果：生成的图像将同时受到运动模糊和高斯噪声的影响，从而模拟实际图像退化过程。这种仿真可以用于测试图像恢复算法的性能以及评估图像质量改善的效果。

4

如果已知图像退化过程的参数，可以采用逆滤波或盲去卷积等图像复原算法来恢复已退化的图像。这些算法的目标是通过估计和逆转退化过程，尽可能减少退化对图像质量的影响。

- 逆滤波是一种常用的图像复原方法，它尝试通过对退化图像进行频域上的逆滤波操作来恢复原始图像。然而，逆滤波对噪声敏感，并且可能会引入增益放大和振铃等伪影效应。
- 盲去卷积是另一种图像复原方法，它试图在不知道准确退化参数的情况下估计退化过程，并对图像进行反卷积操作。盲去卷积算法通常基于假设和正则化技术来提供更稳定的复原结果。

6、思考题

如果退化过程的参数未知，如何进行复原？请编程实现，并与参数已知时的结果进行比较。

实验代码

复原

```
# 导入numpy和scipy库
import numpy as np
import cv2
```

```
from PIL import ImageEnhance, Image
from numpy.fft import fft2, ifft2
from scipy.signal import convolve2d
import matplotlib.pyplot as plt


def show_images(noisy, restored):
    plt.figure(figsize=(12, 6))
    plt.subplot(121), plt.imshow(noisy, cmap='gray'), plt.title('Noisy Image')
    plt.subplot(122), plt.imshow(restored, cmap='gray'), plt.title('Restored
Image')
    plt.show()


def motion_blur(img, size, angle):
    # 创建运动模糊滤波器
    blur_filter = np.ones((size, size))
    blur_filter[int((size - 1) / 2), :] = 0
    blur_filter = cv2.getRotationMatrix2D((size / 2 - 1, size / 2 - 1), angle, 1)
    M = np.eye(size)
    M[:2, :3] = blur_filter
    blur_filter = cv2.warpAffine(M, blur_filter, (size, size))
    blur_filter = blur_filter / np.sum(blur_filter)
    # 应用滤波器
    return cv2.filter2D(img, -1, blur_filter)


def add_gaussian_noise(img, mean=0, sigma=10):
    # 添加高斯噪声
    noise = np.zeros_like(img)
    cv2.randn(noise, mean, sigma)
    noisy_img = cv2.add(img, noise, dtype=cv2.CV_8UC3)
    return noisy_img


# 定义一个高斯核函数
def gauss_kernel(size, sizey=None):
    size = int(size)
    if not sizey:
        sizey = size
    else:
        sizey = int(sizey)
    x, y = np.mgrid[-size:size + 1, -sizey:sizey + 1]
    g = np.exp(-(x ** 2 / float(size) + y ** 2 / float(sizey)))
    return g / g.sum()


# 定义一个维纳滤波函数
def wiener_filter(img, kernel, K):
    kernel /= np.sum(kernel)
    dummy = np.copy(img)
    dummy = fft2(dummy)
    kernel = fft2(kernel, s=img.shape)
    kernel = np.conj(kernel) / (np.abs(kernel) ** 2 + K)
```

```
    dummy = dummy * kernel
    dummy = np.abs(ifft2(dummy))
    return dummy


# 读取noisy.png文件
img = cv2.imread("noisy.png", 0)
cv2.imwrite("noisy.png", img)

# 使用高斯核模拟退化过程
kernel = gauss_kernel(3)

# 使用维纳滤波恢复图像
filtered_img = wiener_filter(img, kernel, K=0.001)
# 保存恢复后的图像
cv2.imwrite("restored.png", filtered_img)
# 创建ImageEnhance对象

f_img = Image.open("restored.png")
# filtered_img=Image.fromarray(filtered_img)
enhancer = ImageEnhance.Brightness(f_img)  # 亮度增强
enhancer = ImageEnhance.Contrast(f_img)  # 对比度增强
# enhancer = ImageEnhance.Color(f_img)  # 色彩增强
enhancer = ImageEnhance.Sharpness(f_img)  # 锐度增强

# 调整增强因子
factor = 1.2  # 可以根据需要调整

# 得到增强后的图像
img_enhanced = enhancer.enhance(factor)

show_images(img, img_enhanced)
```

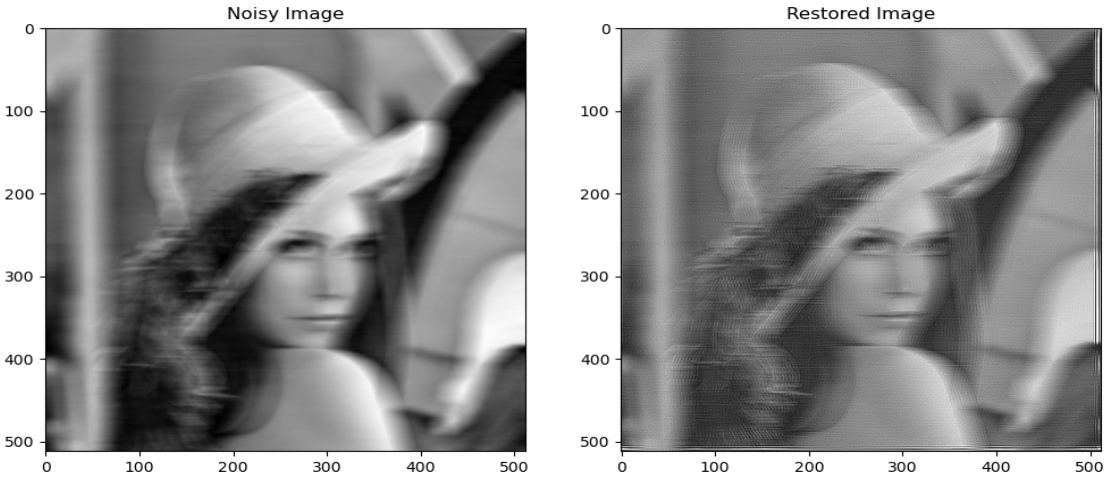
实验结果比较

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

img_wiener = cv2.imread('wiener.png')
img_restored = cv2.imread('restored.png')


def show_images(r1, r2, differ):
    plt.figure(figsize=(16, 6))
    plt.subplot(131), plt.imshow(r1, cmap='gray'), plt.title('Para-known Image')
    plt.subplot(132), plt.imshow(r2, cmap='gray'), plt.title('Para-unknown Image')
    plt.subplot(133), plt.imshow(differ, cmap='gray'), plt.title('Difference')
    plt.show()

img_wiener = np.uint8(img_wiener)
```

```
img_restored = np.uint8(img_restored)
img_diff = cv2.absdiff(img_wiener, img_restored)
show_images(img_wiener, img_restored, img_diff)
```

实验结果

复原



比较

7、收获与感想

邓栩瀛：当退化过程的参数未知时，图像复原变得更具挑战性。因为缺乏准确的参数信息，我们无法直接应用特定的复原算法。此时，需要使用盲复原算法来估计或推断退化参数，以便进行复原操作。盲复原算法在退化参数未知的情况下变得非常重要。它旨在从退化图像中推测出退化过程的参数，并进行复原。这种算法通常基于一些先验知识或假设，如退化过程的模型、图像统计特性等。因此，面对退化参数未知的图像复原问题，我

们需要充分利用盲复原算法和技术，探索更有效的参数估计方法，并结合多种算法的组合来提高复原的准确性和质量。这对于实际图像复原应用具有重要的意义。

钟欣余：在实验中，我尝试了不同的低通滤波器参数，如截止频率或滤波器大小。我观察到较低的截止频率可以保留图像的低频信息，但同时也导致了细节的模糊。而较高的截止频率则能更好地去除高频噪声，但可能会导致图像失去一些细节。然而，如果截止频率设置过高，可能会引入噪声或伪影。通过尝试不同的带通滤波器参数，我观察到带通滤波器可以选择性地保留一定范围内的频率信息。这对于特定频率范围内的信号分析和处理非常有用。通过调整巴特沃斯滤波器的阶数和截止频率，我可以更好地平衡图像的平滑度和细节保留。较高的阶数可以实现更陡峭的截止频率过渡，但可能会导致振铃效应或失真。

金思琪：在实验过程中，通过模拟运动模糊，我观察到图像的细节和边缘模糊化。模糊程度与运动速度和方向有关。这让我意识到在处理实际图像时，需要考虑和处理运动模糊的影响。通过添加高斯随机噪声，我发现图像的清晰度下降，出现了随机的视觉干扰。噪声的强度与噪声方差有关。这让我明白在实际图像处理中需要考虑噪声的抑制问题。

郑越：已知参数的复原：通过编程实现对已退化图像的复原时，我发现复原算法的选择和参数设置对复原效果至关重要。不同的算法可能会产生不同的复原结果，需要根据具体情况进行选择。同时，我也注意到复原过程中可能会出现伪影、失真或增强噪声的问题，需要进一步优化算法或采用后处理技术。总的来说，这个实验让我深入理解了数字图像处理中滤波、退化和复原的原理和方法。通过实践，我进一步掌握了理想滤波器和非理想滤波器的应用，对图像质量退化过程的仿真有了更深入的认识，并且对图像复原的实现有了更具体的了解和体验。