暨南大学计算机科学系

《图像工程与处理》课程设计

成绩：

题　　目：	维纳滤波器与噪声幅度估计
姓　　名：	吕帅
学　　号：	202234261009
专　　业：	电子信息（计算机技术）
课程类别：	专业学位课
任课老师：	彭青玉
提交日期：	2023 年 7 月 6 日

目录

1 引言

1.1 选题背景

所选题目描述：自行实现一个二维的维纳滤波器的设计程序。用该程序去除图片中的颗粒噪声。假定噪声是白噪声，并且用平坦区域上计算出的灰度直方图来估计噪声的幅度。（难度系数 1.20，及格要求：完成二维维纳滤波器的程序，并且能够去除颗粒噪声。加分点：估计噪声幅度）

通过实现维纳滤波器并应用于去噪过程，可以提高图像的质量，恢复丢失的细节信息，使图像更加清晰和可视化。使用平坦区域上的灰度直方图来估计噪声幅度，可以根据噪声的特性来优化滤波过程，从而更准确地去除颗粒噪声。

1.2 白噪声与运动模糊

白噪声（图 1 下）是一种具有平均功率谱密度的随机信号。[1]它的功率谱密度在所有频率上均匀分布，表示为所有频率上的功率都相等。白噪声在时域上表现为随机的、不相关的样本值，没有明显的结构。白噪声的功率谱密度为常数：

$$N(u,v) = \sigma^2 \tag{1}$$

其中，$N(u,v)$是噪声的功率谱密度，σ^2是噪声的方差。**运动模糊**（图 1 上）是一种由于相机或物体的移动而导致图像模糊的现象。[2]当相机在拍摄图像时，如果相机或被拍摄物体有相对运动，图像上的物体会模糊成一个运动轨迹。运动模糊通常由于相机移动或被摄物体移动而导致。它可以在图像中呈现出线性模糊的效果。本实验中同时添加了白噪声与运动模糊。

图 1 运动模糊与白噪声

1.3 维纳滤波器与逆滤波器

维纳滤波器和逆滤波器都是信号处理中常用的滤波器，用于恢复被噪声或失真影响的信号。**维纳滤波器**（Wiener Filter）是一种最小均方误差（Minimum Mean Square Error，MMSE）滤波器，用于恢复被加性噪声污染的信号。[3]它基于信号和噪声的统计特性，并尽可能地减小恢复信号与原始信号之间的均方误差。维纳滤波器的输出可以通过以下公式计算：

$$F(u,v) = H(u,v) \cdot G(u,v) \tag{2}$$

其中，$F(u,v)$是滤波器的输出，$H(u,v)$是维纳滤波器的频率响应，表示为信号的频谱与噪声的频谱之比，$G(u,v)$是被噪声污染的信号的频谱。

逆滤波器（Inverse Filter）是一种理想滤波器的近似，旨在恢复被线性模糊和加性噪声污染的信号。[4]它假设信号在频域上的失真只是线性模糊和加性噪声引起的，而忽略了非线性失真和其他因素。逆滤波器的输出可以通过以下公式计算：

$$F(u,v) = \frac{1}{H(u,v)} \cdot G(u,v) \tag{3}$$

其中，$F(u,v)$是滤波器的输出，$H(u,v)$是逆滤波器的频率响应，表示为信号的频谱的倒数，$G(u,v)$是被模糊和噪声污染的信号的频谱。

维纳滤波器可以看作是逆滤波器的改进版本，考虑了噪声对恢复信号的影响。维纳滤波器在频域上对信号进行加权，以最小化恢复信号与原始信号之间的均方误差。逆滤波器则没有考虑噪声，直接对信号的频谱取倒数。维纳滤波器的频率响应$H(u,v)$可以通过逆滤波器的频率响应$H(u,v)$和噪声功率谱密度$S(u,v)$的关系来表示：

$$H(u,v) = \frac{1}{H(u,v)} \cdot \frac{|\,S(u,v)\,|^2}{|\,S(u,v)\,|^2 + K} \tag{4}$$

其中，K是一个正则化参数，用于避免对噪声功率谱密度过于敏感。综上所述，维纳滤波器是一种考虑了噪声影响的最小均方误差滤波器，而逆滤波器是维纳滤波器的特殊情况，没有考虑噪声。

1.4 估计噪声的幅度

根据平坦区域上计算出的灰度直方图，并假设该部分主要由噪声组成，从中估计噪声的均值μ和方差σ^2。用平坦区域上计算出的灰度直方图来估计噪声的幅度的原理如下：

1. 选择图像中的一个平坦区域，即一个灰度均匀的区域，可以是背景区域或任意没有明显纹理的区域。

2. 计算该区域的灰度直方图，即统计该区域中每个灰度级别的像素数量。

3. 假设该区域的像素值由原始信号和噪声组成，其中噪声是白噪声。根据维纳滤波器的原理，噪声的功率谱密度等于噪声的方差。

4. 通过观察直方图中灰度级别较高的部分，可以估计噪声的功率谱密度。由于噪声是白噪声，可以假设噪声功率谱密度在频率上是均匀的。

具体来说，我的方法的原理是基于以下的假设：一个平滑的区域在空间域中的灰度值变化很小，在频率域中的高频分量很少。[5]因此，如果对一个图像进行低通滤波，那么平滑区域会保持不变，而细节区域会被模糊掉。如果再对滤波后的图像进行边缘检测，那么平滑区域会得到很低的边缘强度，而细节区域会得到较高的边缘强度。如果再对边缘强度图进行二值化和形态学操作，那么平滑区域会形成一个连通的大块，而细节区域会形成一些零散的小块。最后，如果再对二值化图进行连通域分析，那么就可以找到最大的连通区域，即最平滑的区域。从这个区域中选取一个子区域作为平坦区域，就可以用来估计噪声的幅度。

2 开发环境与库函数说明

2.1 开发环境

本实验中采用了 Python 编程语言进行开发，并在 Windows 操作系统下进行了实验。为了方便编写和管理代码，选择 PyCharm 作为集成开发环境（IDE），其强大的代码编辑、调试和版本控制功能，使得开发过程更加高效和便捷，适用于大规模项目的开发和管理；同时选择 Jupyter Notebook 作为交互式开发环境，其允许在网页浏览器中编写和运行代码，并能够直观地展示代码和结果的交互过程，适用于快速原型设计和数据分析。通过这两个工具的结合使用，能够更好地组织和管理代码，并有利于自适应中值滤波器相关的实验过程的记录和分析。

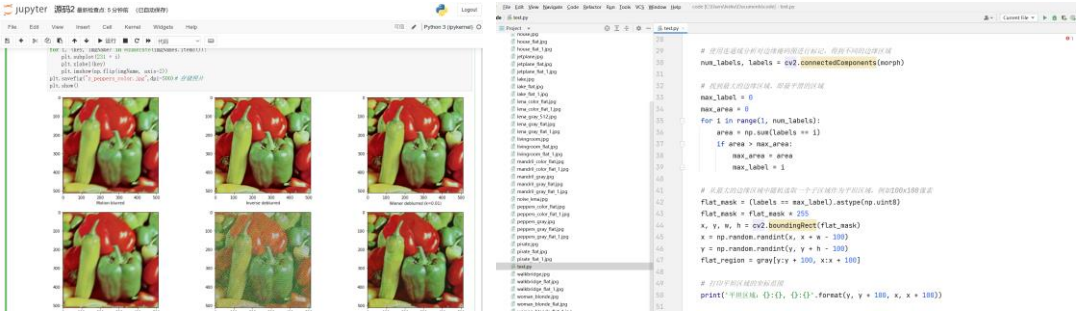


图 2 开发环境示意图

2.2 库函数说明

1. matplotlib.pyplot：Matplotlib 库的一个模块，用于绘图和可视化数据，提供了广泛的绘图函数和方法，可以创建线型图、散点图、柱状图、饼图等，并支持自定义图形属性、标签、标题等。pyplot 模块常用于数据可视化、结果展示和图形分析，在本实验中用于绘制柱状图等。

2. numpy：Python 数值计算工具包，提供了多维数组对象和执行数学、逻辑和统计操作的函数，其核心是 ndarray（N-dimensional array）对象，能够高效地进行数组运算和处理，在本实验中用于数组的创建。

3. cv2：OpenCV（Open Source Computer Vision Library）库的 Python 接口，提供了广泛的计算机视觉和图像处理功能，包含了各种图像处理算法、图像增强、目标检测、人脸识别、图像分割等功能，在本实验中用于读取和展示图片。

4. os：内置模块，用于与操作系统交互和进行文件/目录操作，在本实验中用于批量读取和处理图像文件。

5. math：Python 内置模块，提供了各种数学函数和常量，包括了三角函数、指数函数、对数函数、幂函数等数学运算函数，在本实验中用于高斯函数等公式编写。

6. np.fft：NumPy 的 fft 模块提供了用于快速傅里叶变换（FFT）的函数。在这段代码中，使用 np.fft.fft2 和 np.fft.ifft2 来进行二维傅里叶变换和逆变换。

2.3 编程思想介绍

首先，使用了结构化编程（SP）思想，即复杂的程序分解成简单的子程序，通过顺序、选择和循环等控制结构来组织代码，提高代码的可读性和可维护性。在本实验中，通过 SP 的思想使用函数来封装不同的功能模块，比如添加白噪声、设计维纳滤波器、估计噪声的幅度等，这样可以提高代码的复用性和可读性。

其次，使用了面向对象编程（OOP）思想，即数据和操作封装成对象，通过继承、多态和封装实现代码的复用和扩展。在本实验中，通过 OOP 的思想使用类来定义不同类型的滤波器，例如，可视化模板、滤波器模块、加噪模块等。

最后，使用了面向服务编程（SOA）思想，即程序分解成独立的服务，通过标准化的接口和协议来实现服务的调用和组合，提高代码的可复用性和可扩展性。在本实验中，使用 cv2 库来进行图像的读取、转换和滤波，这样可以提高代码的兼容性和功能性；使用 numpy 库来进行矩阵运算和傅里叶变换，这样可以提高

代码的效率和简洁性。

3 程序设计详细介绍（仅含核心代码）

由于绘图部分的代码相对冗余，且不涉及核心概念，因此**在程序设计详细部分可能仅会展示核心代码，对于一些不重要的绘图代码会进行舍弃，完整代码请参考源代码文件。**

3.1 相关库导入

相关库的导入方便后续的图像加噪、去噪与滤波器设计：

```
# 导入必要的库
import matplotlib.pyplot as plt
import numpy as np
from numpy import fft # 傅里叶变换
import math
import random
import cv2
import os
```

3.2 干扰数据集（添加白噪声、添加运动模糊）

首先展示 3.2 节的流程图，如下：

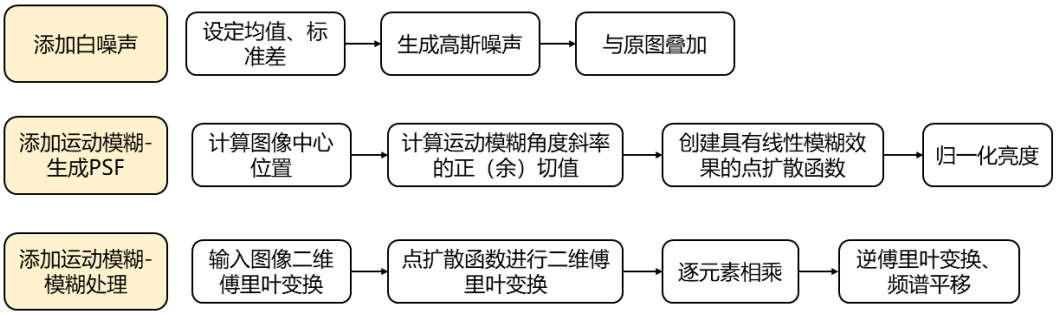


图 3 《干扰数据集》流程图

在为数据集添加干扰时，除了添加噪声之外，还可以额外添加运动模糊。运动模糊是一种常见的图像模糊类型，它模拟了相机或被拍摄物体的运动过程中的模糊效果。运动模糊的原理是在图像中引入了运动轨迹，使得物体或场景在图像上呈现出模糊的效果。首先是添加白噪声函数设计，其通过生成高斯分布的噪声添加到原图来实现：

```
def add_gaussian_white_noise(input_image, ratio):
    """
    添加加性高斯白噪声到输入图像。

    参数:
        input_image (ndarray): 输入图像数组。
        ratio (float): 噪声强度比例。

    返回:
        intput_signal_cp (ndarray): 添加噪声后的图像数组。
    """
    intput_signal_cp = np.copy(input_image)# 输入图像的副本
    noise = np.random.normal(0, 0.1 * input_image.std(),
size=intput_signal_cp.shape)# 生成高斯噪声
    intput_signal_cp += noise# 添加高斯白噪声

    return intput_signal_cp
```

接着是运动模糊的函数设计，apply_motion_blur 用于对输入图像进行运动模糊处理；generate_motion_PSF 用于生成仿真的运动模糊点扩散函数（PSF）：

```
def generate_motion_PSF(image_shape, angle_of_motion):
    """
    生成仿真的运动模糊点扩散函数(PSF)。

    参数:
        image_shape (tuple): 图像形状，形如 (height, width)。
        angle_of_motion (float): 运动模糊的角度，以度为单位。

    返回:
        psf (ndarray): 形状为 image_shape 的点扩散函数数组。
    """
    psf = np.zeros(image_shape)  # 创建形状为 image_shape 的零数组
    center = (image_shape[0] - 1) / 2  # 计算图像中心位置

    slope_tan = math.tan(angle_of_motion * math.pi / 180)  # 计
算斜率的正切值
    slope_cot = 1 / slope_tan  # 计算斜率的余切值

    if slope_tan <= 1:
        for i in range(15):
            offset = round(i * slope_tan)  # 计算每个像素的偏移量
            row = int(center + offset)
```

```
            col = int(center - offset)
            psf[row, col] = 1  # 设置对应位置的像素为 1
        return psf / psf.sum()  # 对点扩散函数进行归一化亮度
    else:
        for i in range(15):
            offset = round(i * slope_cot)  # 计算每个像素的偏移量
            row = int(center - offset)
            col = int(center + offset)
            psf[row, col] = 1  # 设置对应位置的像素为 1
        return psf / psf.sum()  # 对点扩散函数进行归一化亮度

def apply_motion_blur(input_image, psf, epsilon):
    """
    对输入图像进行运动模糊处理。

    参数:
        input_image (ndarray): 输入图像数组。
        psf (ndarray): 点扩散函数数组。
        epsilon (float): 避免除零错误的小值。

    返回:
        blurred_image (ndarray): 经过运动模糊处理后的图像数组。
    """
    input_fft = fft.fft2(input_image)  # 进行二维数组的傅里叶变换
    psf_fft = fft.fft2(psf) + epsilon
    blurred_fft = input_fft * psf_fft
    blurred_image =
np.abs(fft.fftshift(fft.ifft2(blurred_fft)))  # 取绝对值并进行频
谱平移
    return blurred_image
```

3.3 滤波器设计（维纳滤波器与逆滤波器）

首先展示 3.3 节的流程图，如下：

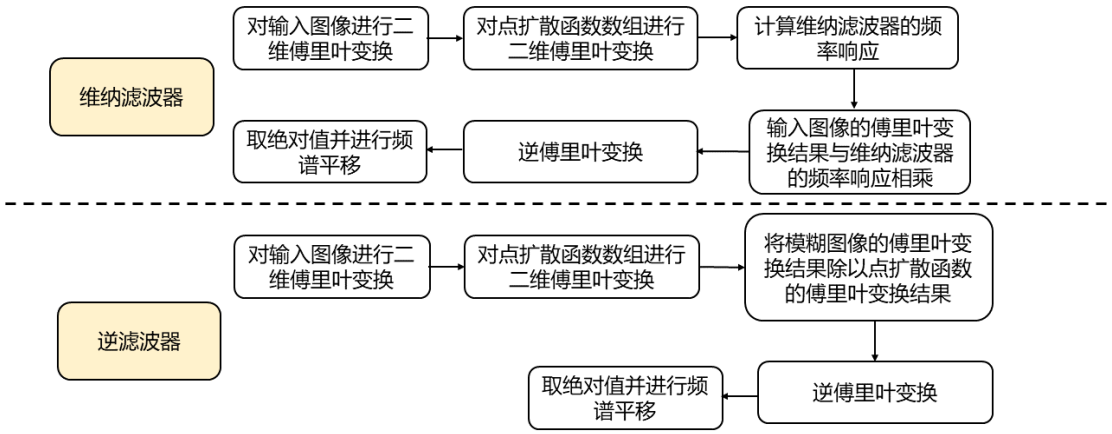


图 4 《滤波器设计》流程图

本节实现了维纳滤波和逆卷积，用于恢复经过模糊和噪声影响的图像。其中，`wiener_filter`函数实现了维纳滤波器，其主要过程包括对输入图像和点扩散函数进行傅里叶变换，计算维纳滤波器的频率响应，将输入图像的频谱乘以维纳滤波器的频率响应，然后进行逆傅里叶变换和频谱平移，最终返回恢复后的图像。`inverse_filter`函数实现了逆卷积，主要步骤包括对模糊图像和点扩散函数进行傅里叶变换，进行频域的除法操作，然后进行逆傅里叶变换和频谱平移，最终返回恢复后的图像。首先是维纳滤波器：

```
def wiener_filter(input, PSF, eps, K=0.01):
    """
    维纳滤波函数，用于恢复经过模糊和噪声影响的图像。

    参数:
        input (ndarray): 输入图像数组。
        PSF (ndarray): 点扩散函数数组。
        eps (float): 避免除零错误的小值。
        K (float): 维纳滤波的正则化参数，默认值为 0.01。

    返回:
        result (ndarray): 恢复后的图像数组。
    """
    input_fft = fft.fft2(input)  # 进行二维数组的傅里叶变换
    PSF_fft = fft.fft2(PSF) + eps  # 计算点扩散函数的傅里叶变换，
添加 epsilon 避免除零错误
    PSF_fft_1 = np.conj(PSF_fft) / (np.abs(PSF_fft) ** 2 + K)  #
维纳滤波器的频率响应
    result = fft.ifft2(input_fft * PSF_fft_1)  # 傅里叶反变换
    result = np.abs(fft.fftshift(result))  # 取绝对值并进行频谱平
移
    return result
```

接着是逆滤波器：

```
def inverse_filter(blurred_image, point_spread_function,
epsilon):
    """
    逆去卷积函数，用于恢复经过模糊和噪声影响的图像。

    参数:
        blurred_image (ndarray): 模糊图像数组。
        point_spread_function (ndarray): 点扩散函数数组。
        epsilon (float): 避免除零错误的小值。

    返回:
        restored_image (ndarray): 恢复后的图像数组。
    """
    blurred_fft = fft.fft2(blurred_image)  # 进行模糊图像的二维傅
里叶变换
    psf_fft = fft.fft2(point_spread_function) + epsilon  # 计算
点扩散函数的傅里叶变换，添加 epsilon 避免除零错误
    restored_fft = blurred_fft / psf_fft  # 进行频域的除法操作
    restored_image =
np.abs(fft.fftshift(fft.ifft2(restored_fft)))  # 进行频谱平移并
进行傅里叶反变换，取绝对值
    return restored_image
```

3.4 通用函数设计

首先展示 3.4 节的流程图，如下：

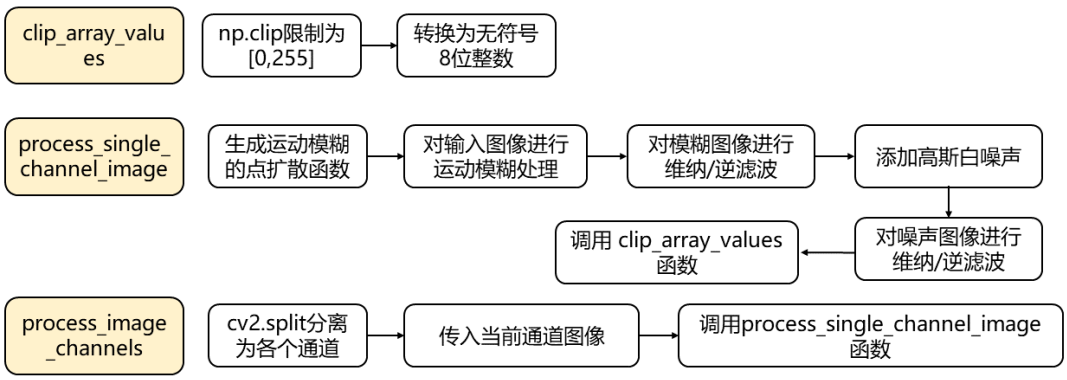


图 5 《通用函数设计》流程图

`clip_array_values`函数的目的是将输入的数组中的值限制在合理范围内。它首先使用 np.clip 函数将数组中的值限制在 0 到 255 的范围内，然后将数组的数据类型转换为无符号 8 位整数（np.uint8）。最终返回经过限制处理后的数组。

```
def clip_array_values(array):
    """
    将数组中的值限制在合理范围内。

    参数:
        array (ndarray): 输入数组。

    返回:
        clipped_array (ndarray): 经过限制处理后的数组。
    """
    clipped_array = np.clip(array, 0, 255)  # 将数组中的值限制在[0, 255] 范围内
    clipped_array = clipped_array.astype(np.uint8)  # 将数组类型转换为无符号 8 位整数
    return clipped_array
```

`process_single_channel_image`函数用于对单通道图像进行运动模糊和滤波处理，并返回不同滤波结果的列表，如下：

```
def process_single_channel_image(image):
    """
    用于对单通道图像进行运动模糊和滤波处理。

    参数:
        image (ndarray): 单通道图像数组。

    返回:
        results (list): 包含不同滤波结果的列表。
    """
    results = []
    height, width = image.shape[:2]  # 获取图像的高度和宽度

    # 运动模糊处理
    motion_PSF = generate_motion_PSF((height, width), 60)  # 生成运动模糊的点扩散函数
    blurred_image = apply_motion_blur(image, motion_PSF, 1e-3)  # 进行运动模糊处理

    # 运动模糊滤波处理
    deblurred_result = inverse_filter(blurred_image, motion_PSF, 1e-3)  # 逆去卷积恢复模糊图像
    wiener_result = wiener_filter(blurred_image, motion_PSF, 1e-3)  # 维纳滤波处理模糊图像
```

```
    # 运动模糊加入白噪声处理
    blurred_noisy_image =
add_gaussian_white_noise(blurred_image, 1)  # 在模糊图像上添加高
斯白噪声

    # 运动模糊加入白噪声滤波处理
    inverse_mo2no_result = inverse_filter(blurred_noisy_image,
motion_PSF, 0.1 + 1e-3)  # 对加噪声的图像进行逆去卷积
    wiener_mo2no_result = wiener_filter(blurred_noisy_image,
motion_PSF, 0.1 + 1e-3)  # 对加噪声的图像进行维纳滤波

    # 将处理结果添加到列表中
    results.append((clip_array_values(blurred_image),
                    clip_array_values(deblurred_result),
                    clip_array_values(wiener_result),
                    clip_array_values(blurred_noisy_image),
                    clip_array_values(inverse_mo2no_result),
                    clip_array_values(wiener_mo2no_result)))
#     # 将处理结果添加到列表中
#     results.append((clip_array_values(blurred_noisy_image),
#                     clip_array_values(inverse_mo2no_result),
#                     clip_array_values(wiener_mo2no_result)))

    return results
```

`process_image_channels`函数用于对图像的各个通道（R、G、B 通道）进行处理，并返回不同滤波结果的列表。

```
def process_image_channels(image):
    """
    对图像的各个通道进行处理。

    参数:
        image (ndarray): 输入图像数组。

    返回:
        results (list): 包含不同滤波结果的列表。
    """
    results = []
    channels = cv2.split(image.copy())  # 分离图像通道，得到灰度图
像的各个通道
```

```
    for channel in channels:
        channel_results =
process_single_channel_image(channel)  # 对每个通道的灰度图像进行
处理
        results.append(channel_results)

    return results
```

最后，相关调用测试的代码如下：

```
image = cv2.imread('standard_test_images/lena_gray_512.tif')# 读
取图像
image_results = process_image_channels(image)# 处理图像的各个通道

result_names = ["blurred", "result_blurred", "result_wiener",
"blurred_noisy", "inverse_mo2no", "wiener_mo2no"]# 定义结果图像的
名称
result_images = []# 存储结果图像的列表

# # 提取结果图像
# for i in range(3):
#     channel_images = [image_results[j][0][i] for j in range(3)]#
提取每个通道的图像
#     result_images.append(cv2.merge(channel_images))# 合并通道
图像
# result_images = dict(zip(result_names, result_images))# 将结果
图像列表与名称列表进行关联，形成字典

# 提取各个结果图像
blurred = cv2.merge([image_results[0][0][0],
image_results[1][0][0], image_results[2][0][0]])
result_blurred = cv2.merge([image_results[0][0][1],
image_results[1][0][1], image_results[2][0][1]])
result_wiener = cv2.merge([image_results[0][0][2],
image_results[1][0][2], image_results[2][0][2]])
blurred_noisy = cv2.merge([image_results[0][0][3],
image_results[1][0][3], image_results[2][0][3]])
inverse_mo2no = cv2.merge([image_results[0][0][4],
image_results[1][0][4], image_results[2][0][4]])
wiener_mo2no = cv2.merge([image_results[0][0][5],
image_results[1][0][5], image_results[2][0][5]])
```

3.5 自动化平坦域来估计噪声幅度

首先展示 3.5 节的流程图，如下：

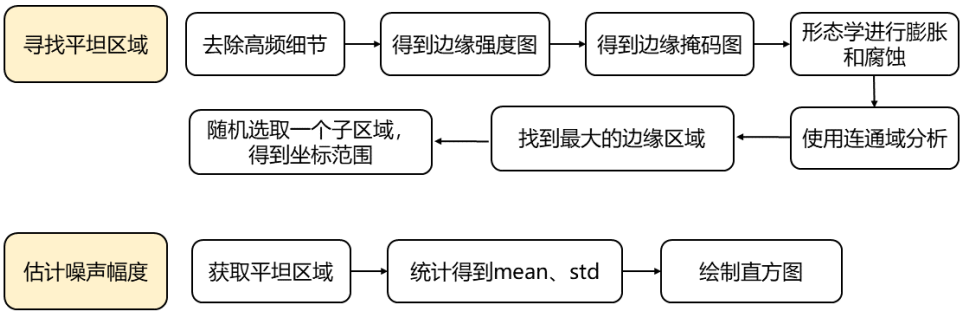


图 6 《估计噪声幅度》流程图

一方面，一幅图中的平坦区域可以通过观察进行框选，**手工设置**范围，进而估计出噪声的幅度（均值与标准差）。另一方面，一个平滑的区域在空间域中的灰度值变化很小，通过对二值化图进行连通域分析，那么就可以找到最大的连通区域，即最平滑的区域。从这个区域中**自动地**选取一个子区域作为平坦区域，就可以用来估计噪声的幅度。`find_flat_region`函数的目标是在给定的图像中自动的寻找平坦区域并显示结果与坐标。

```
def find_flat_region(image):
    """
    在给定图像中寻找平坦区域并显示结果。

    参数:
        image: 要处理的输入图像（彩色图像）

    返回值:
        无（显示结果图像）

    """
    # 图片读取
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(gray, (21, 21), 0)  # 使用高斯滤波器对图像进行低通滤波，去除高频细节
    edge = cv2.Laplacian(blur, cv2.CV_64F)  # 使用拉普拉斯算子对滤波后的图像进行边缘检测，得到边缘强度图
    thresh = cv2.threshold(edge, 10, 255, cv2.THRESH_BINARY)[1].astype(np.uint8)  # 使用阈值操作对边缘强度图进行二值化，得到边缘掩码图
```

```
    # 使用形态学操作对边缘掩码图进行膨胀和腐蚀，去除小的边缘区域和填补小的空洞
    kernel = np.ones((3, 3), np.uint8)
    morph = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
    morph = cv2.morphologyEx(morph, cv2.MORPH_CLOSE, kernel)

    # 使用连通域分析对边缘掩码图进行标记，得到不同的边缘区域
    num_labels, labels = cv2.connectedComponents(morph)

    # 找到最大的边缘区域，即最平滑的区域
    max_label = 0
    max_area = 0
    for i in range(1, num_labels):
        area = np.sum(labels == i)
        if area > max_area:
            max_area = area
            max_label = i

    # 从最大的边缘区域中随机选取一个子区域作为平坦区域，例如 100x100 像素
    flat_mask = (labels == max_label).astype(np.uint8)
    flat_mask = flat_mask * 255
    x, y, w, h = cv2.boundingRect(flat_mask)
    x = np.random.randint(x, x + w - 100)
    y = np.random.randint(y, y + h - 100)
    flat_region = gray[y:y+100, x:x+100]

    # 打印平坦区域的坐标范围
    print('平坦区域: {}:{}, {}:{}'.format(y, y + 100, x, x + 100))

    # 显示结果
    plt.figure(figsize=(12, 8))
    plt.subplot(141)
    plt.imshow(gray, cmap='gray')
    plt.title('Original image')
    plt.subplot(142)
    plt.imshow(blur, cmap='gray')
    plt.title('Blurred image')
    plt.subplot(143)
    plt.imshow(edge, cmap='gray')
    plt.title('Edge intensity image')
    plt.subplot(144)
```

```
    plt.imshow(flat_region, cmap='gray')
    plt.title('Flat region image')
    plt.savefig("peppers_color_flat.jpg",dpi=500)# 存储照片
    plt.show()
find_flat_region(imgs[14])
```

选定区域后，通过调用 np.mean 与 np.std 即可进行噪声近似统计，我们同时打印出了直方图来观察，这是由于平缓区域地直方图通常是单峰的，方便我们进一步验证。

```
img = cv2.split(imgs[14])[0]
row, col = img.shape
# 加噪
gauss = np.random.normal(10,10,(row,col))
noisy = img + gauss
# 平坦区域
smooth_part = noisy[304:404, 256:356]
# 平坦区域的均值和标准差
mean_value = np.mean(smooth_part)
std_value = np.std(smooth_part)
# 打印结果
print("平坦区域的均值：", mean_value)
print("平坦区域的标准差：", std_value)

plt.figure(figsize=(12, 12))# 图片大小
plt.subplot(221),plt.imshow(noisy,cmap = 'gray')
plt.title('Noisy Image'), plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(smooth_part,cmap = 'gray')
plt.title('Smooth Part'), plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.hist(noisy.ravel(),256,[0,256])#;
plt.show()
plt.title('Noisy Image Histogram'), plt.xticks([]),
plt.yticks([])
plt.subplot(224),plt.hist(smooth_part.ravel(),256,[0,256])#;
plt.show()
plt.title('Estimated Noise Distribution'), plt.xticks([]),
plt.yticks([])
plt.savefig("peppers_color_flat_1.jpg",dpi=500)
plt.show()
```

4 程序实验结果与分析

实验结果分为定性实验和定量实验，其中，**定性实验**结果会对去噪结果与直方图进行分析，**定量实验**结果通过平坦区域、噪声幅度估计组成。

4.1 不同方差噪声下的维纳滤波器与逆滤波器

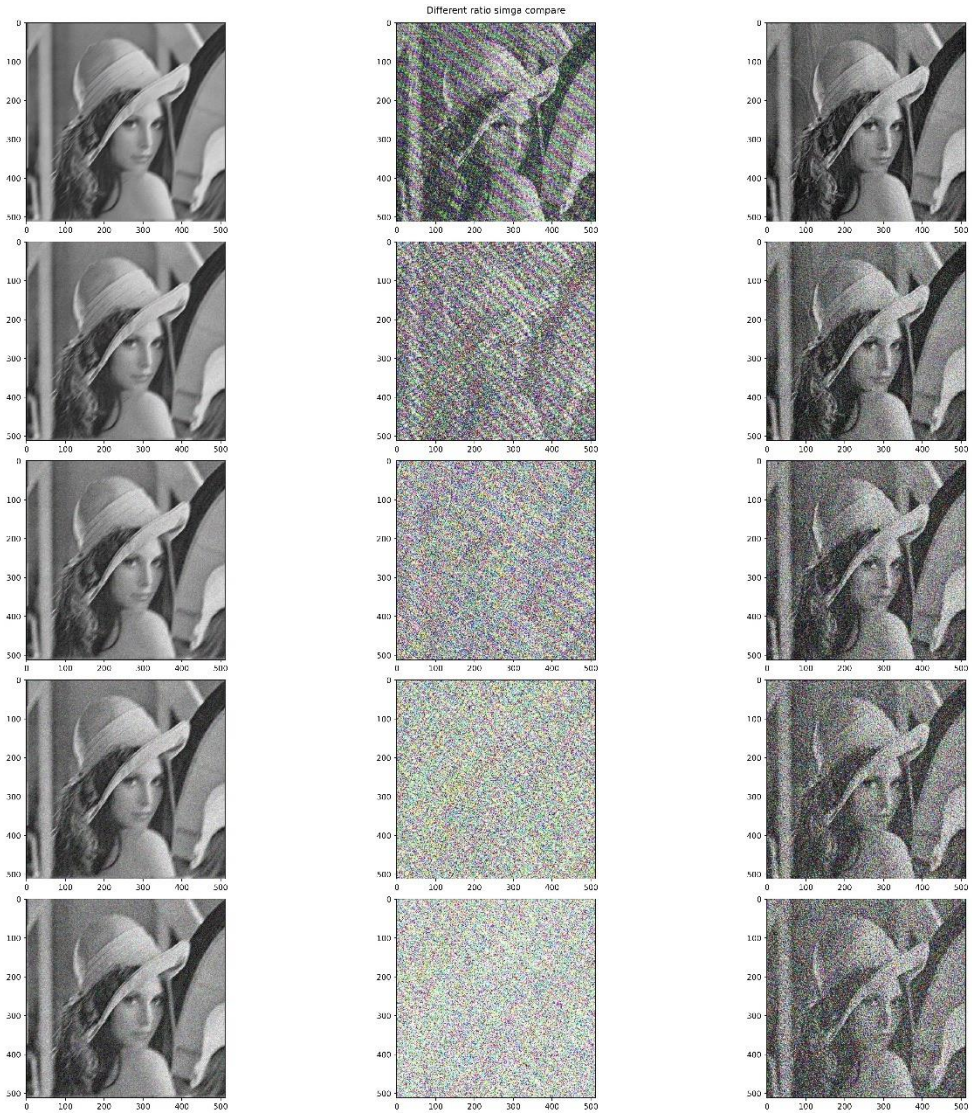


图 7 方差所占比例=0.1、0.2、0.3、0.4、0.5 下维纳滤波器（中）与逆滤波器（右）的表现

可以看到，逆滤波器对噪声是非常敏感的，当方差所占比例达到 0.3 以上之后已经无法辨认了，而维纳滤波器对噪声有一定的识别能力，但在不同程度的方差下也会受到比较严重的影响，因此，在之后的方差比例设定为 0.2。

4.2 数据集上两种滤波器的运动模糊去除与去噪对比

首先以 Lena 为例，对原始图片进行运动模糊化、高斯白噪化，如下图：

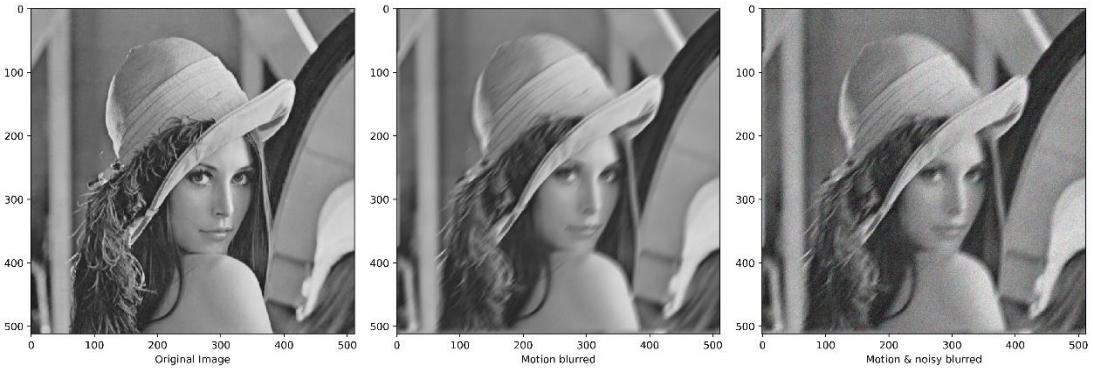
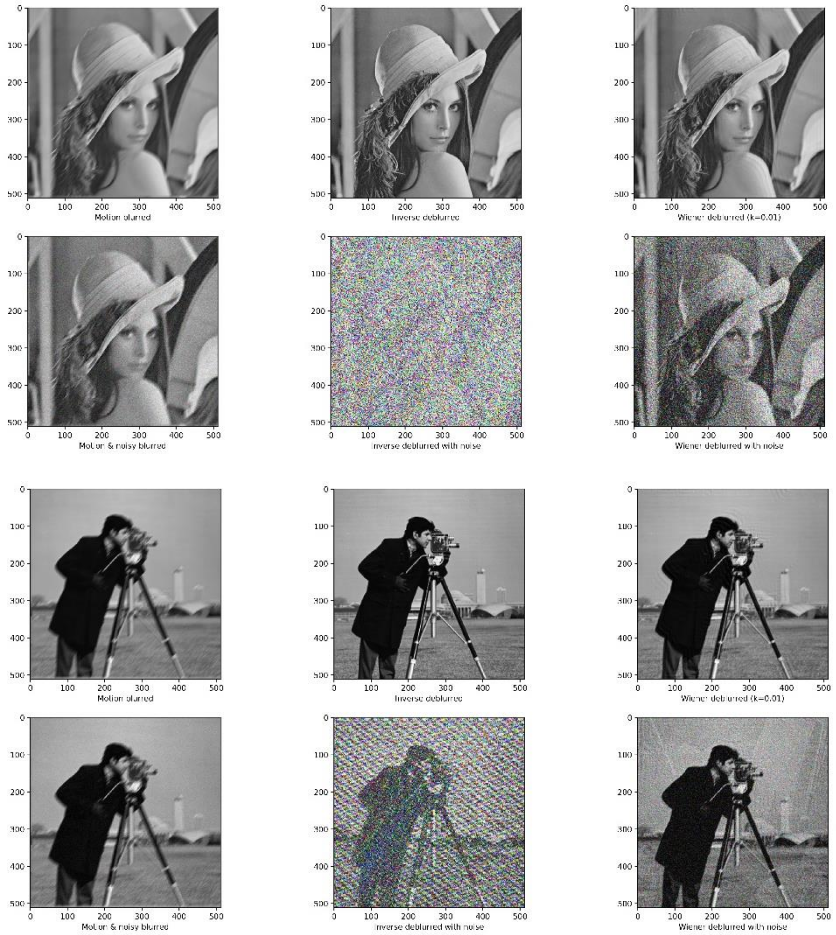
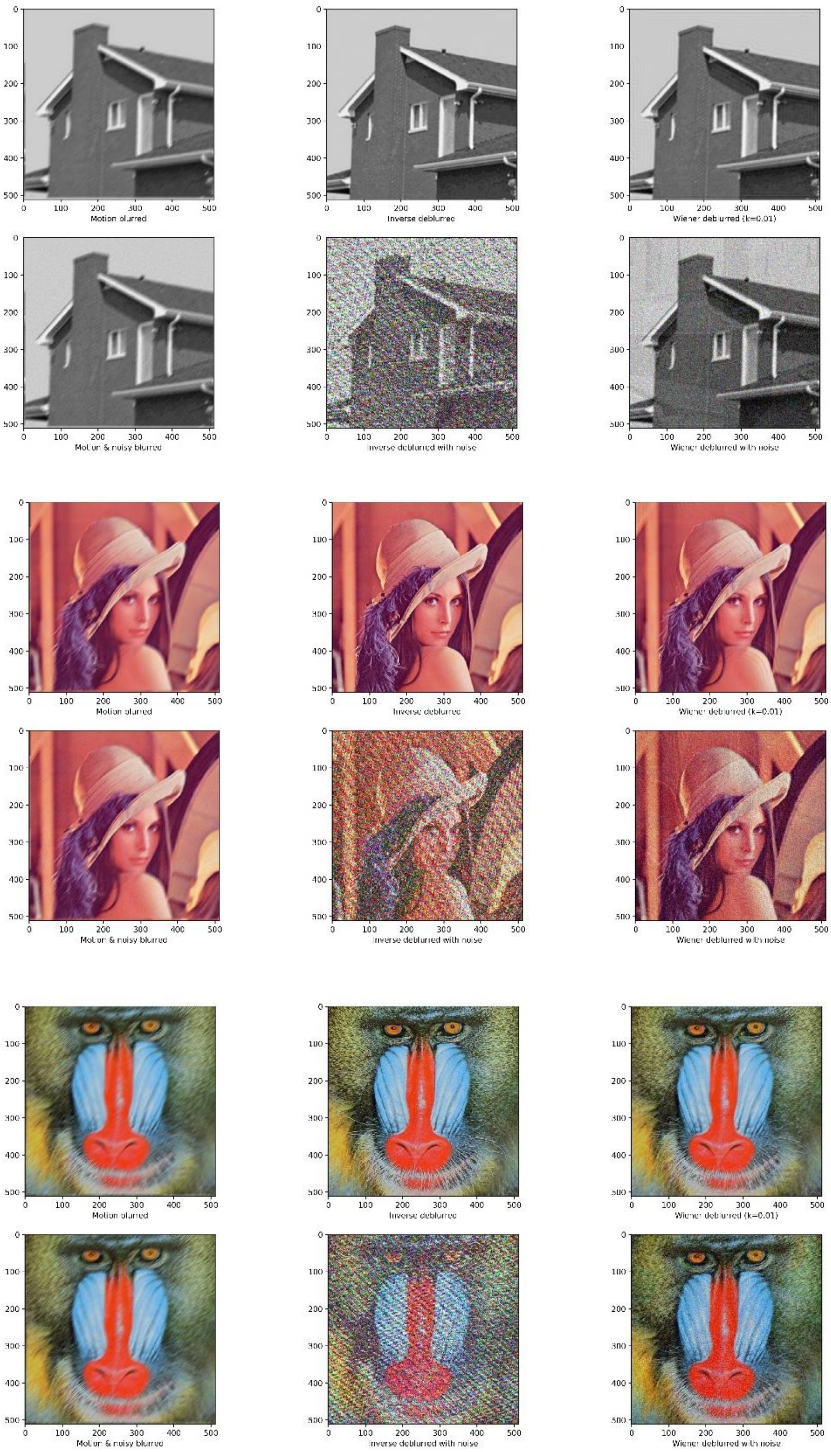


图 8 原图、运动模糊图、运动模糊加噪图

随后对数据集进行了广泛的测试，我们以 3 张灰度图、3 张 RGB 图为例进行展示：




Motion blurred
Inverse deblurred
Wiener deblurred (k=0.01)
Motion & noisy blurred
Inverse deblurred with noise
Wiener deblurred with noise
Motion blurred
Inverse deblurred
Wiener deblurred (k=0.01)
Motion & noisy blurred
Inverse deblurred with noise
Wiener deblurred with noise
Motion blurred
Inverse deblurred
Wiener deblurred (k=0.01)
Motion & noisy blurred
Inverse deblurred with noise
Wiener deblurred with noise

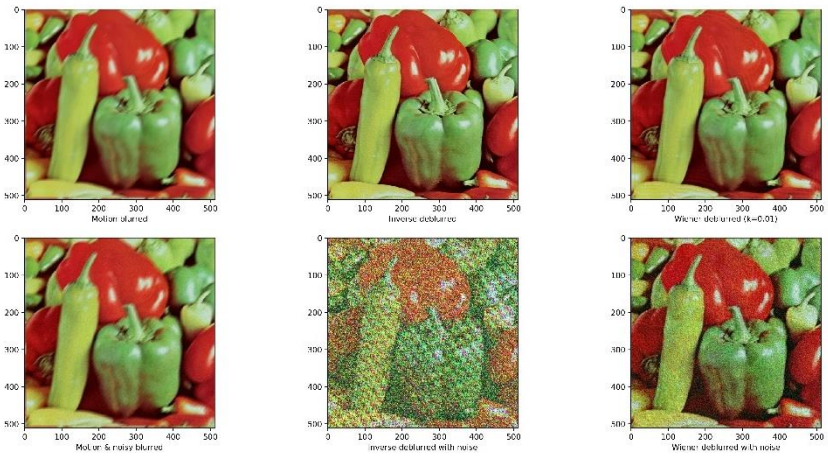
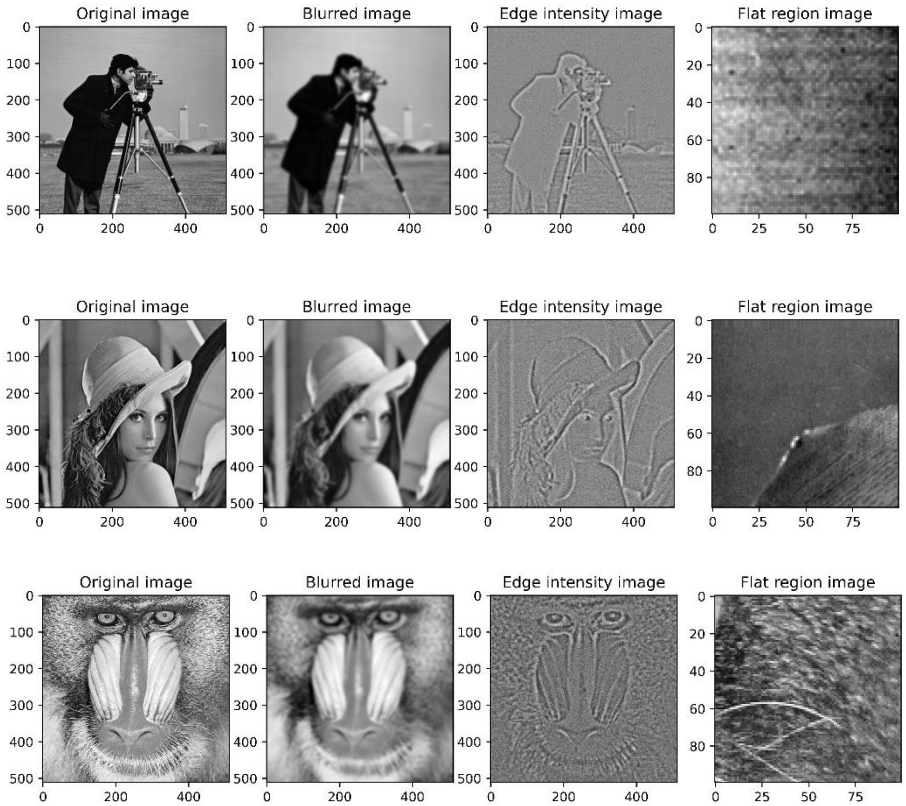


图 9 运动模糊图与运动模糊加噪图两种滤波器表现

随可以总结出，两种滤波器对运动模糊均可以很好的去模糊化，但在噪声方面，逆滤波器对噪声非常敏感，容易得到损失的图片，维纳滤波器的抗噪能力相对较高。此外，该程序的通用性很好，同时适配灰度图与 RGB 图。

4.3 平坦区域的直方图与噪声幅度估计

首先，我们需要自动取得良好的平坦区域，定性结果如下，从左至右分别原图、模糊图、边缘密度图以及选定的平坦区域，我们将展示 4 张灰度图，更多测试结果见文件夹：



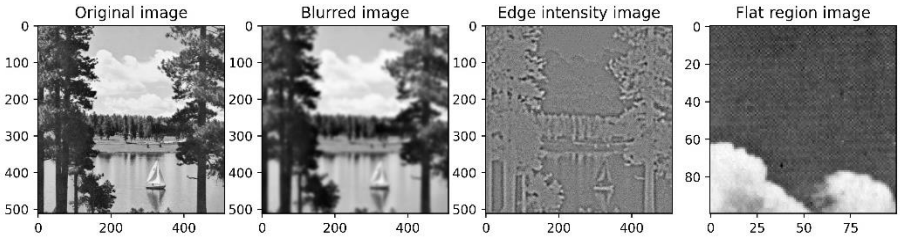


图 10 自动选取地图像平坦区域

可以看到，平坦区域均选择颜色较为简单地“衣服”、“背景”、“皮毛”与“天空”，在获得了指定平坦区域区域后，通过计算均值和方差即可对噪声幅度进行评估，见表 1：

表 1 自动取得的平坦区域像素范围、噪声幅度估计

图片	平坦区域	噪声均值估计	噪声标准差估计
cameraman.tif	(157:257, 353:453)	178.029	10.927
house.tif	(24:124, 329:429)	214.020	13.077
jetplane.tif	(59:159, 169:269)	203.454	13.327
lake.tif	(3:103, 173:273)	200.843	17.616
lena_gray_512.tif	(1:101, 114:214)	137.217	13.332
livingroom.tif	(1:101, 342:442)	132.212	32.430
mandril_gray.tif	(385:485, 139:239)	150.043	29.043
peppers_gray.tif	(137:237, 98:198)	174.787	26.299
pirate.tif	(275:375, 304:404)	144.531	37.898
walkbridge.tif	(313:413, 39:139)	148.699	42.013
woman_blonde.tif	(52:152, 263:363)	166.092	39.984
woman_darkhair.tif	(358:458, 319:419)	56.209	10.302
z_lena_color_512.tif	(279:379, 228:328)	106.303	52.314
z_mandril_color.tif	(248:348, 371:471)	86.1728	34.287
z_peppers_color.tif	(304:404, 256:356)	86.115	34.563

同时，为了估计平坦区域的像素分布情况，我们进一步对所选择的平坦区域的直方图进行了绘制，自动选择的区域呈现出良好的单峰高斯状态，但面对较为复杂的颜色变换，自动选择的区域效果较差，详细可见文件夹中的实验结果，我们同样以四张为例：

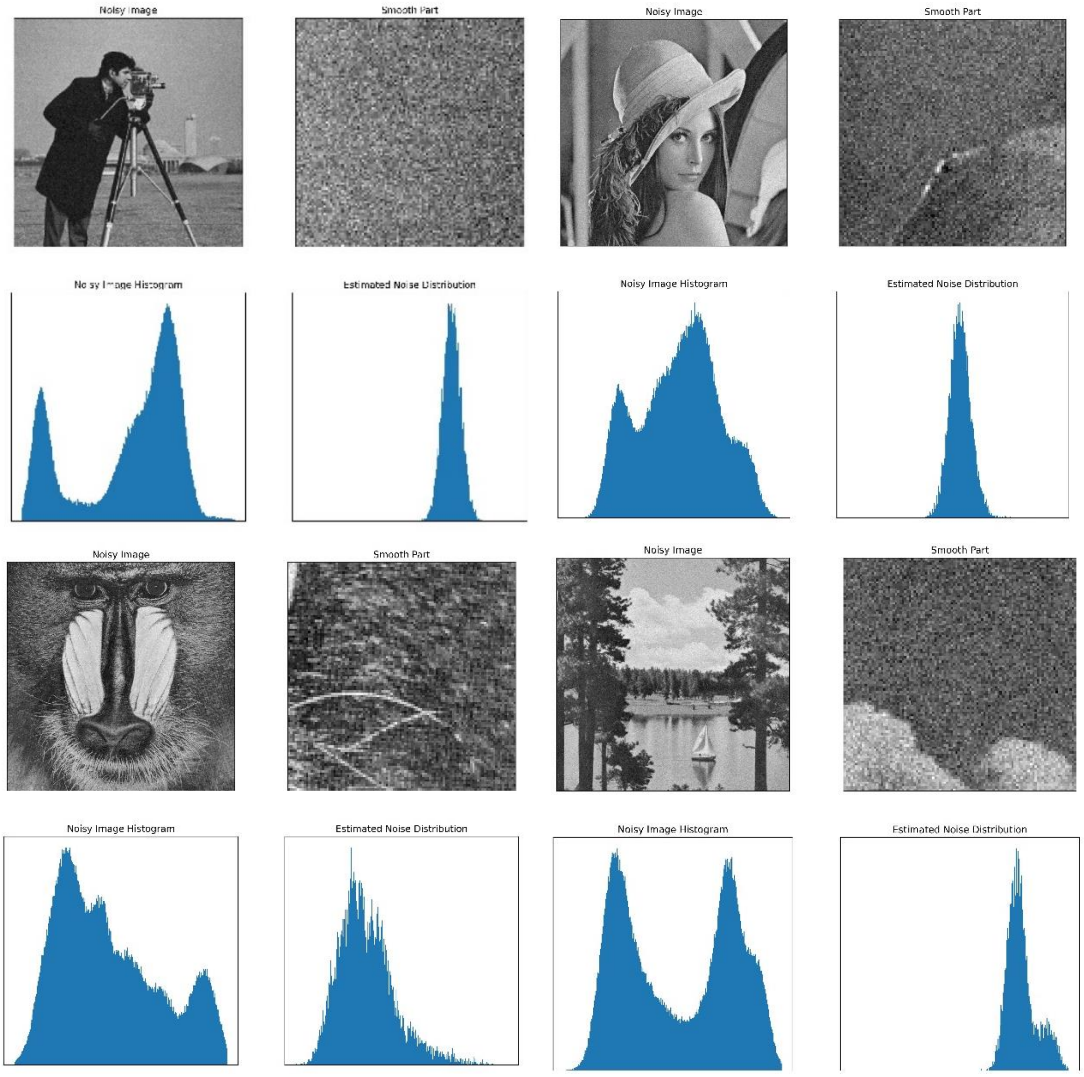


图 11 平坦区域与原图的直方图展示

5 实验总结

本实验在加噪阶段，针对 15 张灰度/RGB 图片添加了高斯白噪声，在不同比例的方差下选择比例为 0.2，同时，在加噪的基础上增加了运动模糊；在去噪阶段设计了维纳滤波器、逆滤波器，并对实验结果进行了广泛的讨论与验证，说明了维纳滤波器抗噪能力较好的特点；此外，为了估计噪声幅度，设计了算法来自动选取平坦区域，对实验结果进行了详尽的记录，说明了平坦区域像素直方图呈现单峰高斯的形状，不足的地方在于对待颜色变化多的图像难以寻优。

参考文献

[1] 郑婉蓉,谢凌云. 基于语谱图灰度变换方法的语音增强[C]//中国声学学会.2018 年全国声学大会论文集 J 通信声学与音频信号处理(含声频工程).[出版者不详],2018:18-19.

[2] 高海韬. 基于稀疏正则化的运动模糊图像复原[D].贵州大学,2022.

[3] 韩笑雪,孙尚,毛文梁.基于改进维纳滤波的自适应电力线通信降噪方法[J].电脑与信息技术,2023,31(03):55-57.

[4] 陈黎艳,熊强强,曾美琳.基于改进逆滤波的红外图像目标信息快速复原研究[J].激光杂志,2022,43(08):164-168.

[5] 严凯,陆千里,丁力等.脉冲噪声下改进型正弦信号幅度估计方法[J].航空计算技术,2021,51(06):42-45+50.

[6] python - Image noise estimation base on histogram - Stack Overflow

[7] aasp-noise-red/src/presentation.ipynb at master · lcolbois/aasp-noise-red · GitHub

[8] Wiener-Filter/Wiener Filter.ipynb at master · manasbedmutha98/Wiener-Filter · GitHub

[9] final_project | Kaggle

[10] OpenCV—Python 图像去模糊（维纳滤波，约束最小二乘方滤波）_51CTO 博客_opencv 对图像进行滤波