安徽大学《数字图像处理（双语）》实验报告（4）

学号：WA2214014 专业：人工智能 姓名及签字：杨跃浙

实验日期：24.12.21 实验成绩： 教师签字：

【实验名称】： 基于空域模板处理的图像增强技术

【实验目的】：

1. 熟悉和掌握基于空域模板的图像增强原理
2. 通过 MATLAB 编程实现均值平滑滤波的图像增强技术
3. 通过 MATLAB 编程实现局部直方图均衡的的图像增强
4. 通过 MATLAB 编程实现基于拉普拉斯滤波的图像增强

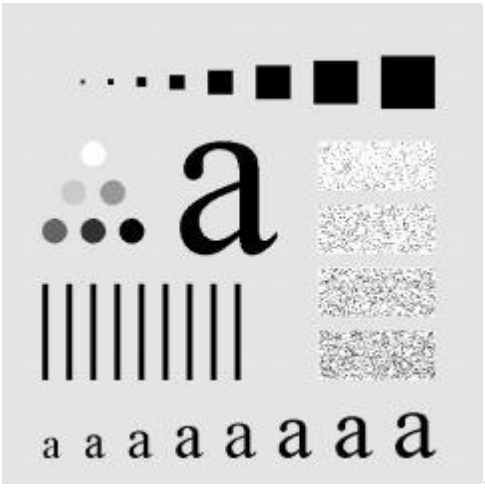
【实验内容】

PROJECT 03-07

Spatial Filtering

Write program to perform spatial filtering of an image "Fig_test_pattern_blurring_orig.tif". Change the size of the spatial mask at 3 x 3, 5 x 5, 9 x 9, 15 x 15, 35 x 35 and compare your results with the textbook

square average



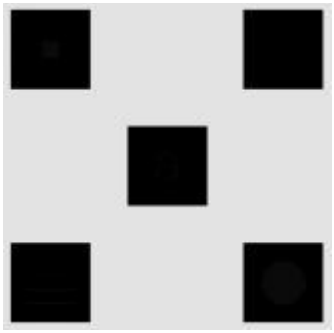
PROJECT 03-08

Local histogram processing

(a) Implement the local histogram equalization on image "embedded_square_noisy" using

neighborhood of size 3 × 3. Note that different from the spatial filtering, you can just replace the original 3 × 3 subimage by the histogram equalized values and move 3 pixels for each step.

(b) Compare the result with the global histogram equalization.



PROJECT 03-09

Averaging filter and Median Filter

The following image is corrupted by salt-and-papper noise.

(a) Implement 3 ×3 averaging filter on image "ckt_board_saltpep" .

(b) Implement 3 ×3 median filter on image "ckt_board_saltpep" .

(c) Compare and explain the differences between the averaging and media filters

PROJECT 03-010

Enhancement Using the Laplacian

(a) Write program to perform Laplacian enhancement technique on image "Fig_blurry_moon.tif" with both the two masks

0	-1	0
-1	5	-1
0	-1	0

-1	-1	-1
-1	9	-1
-1	-1	-1

(b) Compare the results

【实验代码和结果】

Project03-07：

Code:

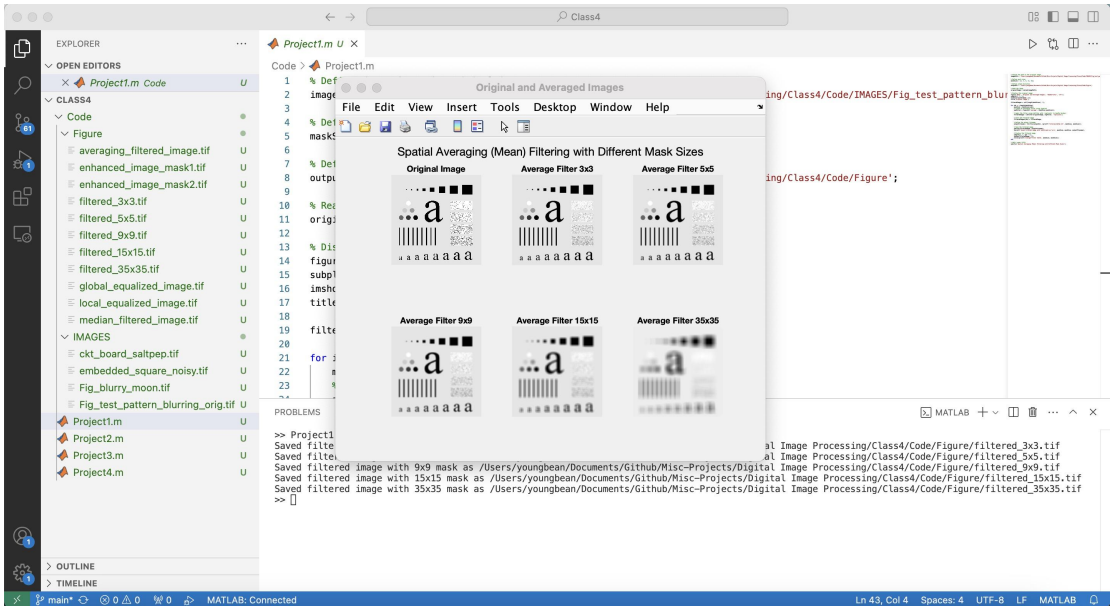
Project1.m:

```
% Define the path to the original image
imagePath =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image
```

```
Processing/Class4/Code/IMAGES/Fig_test_pattern_blurring_o
rig.tif';

% Define mask sizes
maskSizes = [3, 5, 9, 15, 35];

% Define output directory
outputDir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class4/Code/Figure';

% Read the image
originalImage = imread(imagePath);

% Display the original image
figure('Name', 'Original and Averaged Images', 'NumberTitle',
'off');
subplot(2, 3, 1);
imshow(originalImage, []);
title('Original Image');

filteredImages = cell(length(maskSizes), 1);

for idx = 1:length(maskSizes)
maskSize = maskSizes(idx);
% Create an averaging filter using fspecial
avgFilter = fspecial('average', [maskSize maskSize]);
% Apply the filter using imfilter with 'replicate' to handle
borders
filteredImage = imfilter(originalImage, avgFilter,
'replicate');
% Store the filtered image
filteredImages{idx} = filteredImage;
% Define the output filename
outputFilename = fullfile(outputDir,
sprintf('filtered_%dx%d.tif', maskSize, maskSize));
% Save the filtered image
imwrite(filteredImage, outputFilename);
```

```
fprintf('Saved filtered image with %dx%d mask as %s\n',
maskSize, maskSize, outputFilename);
% Display the filtered image
subplot(2, 3, idx + 1);
imshow(filteredImage, []);
title(sprintf('Average Filter %dx%d', maskSize, maskSize));
end

% Add a super title
sgtitle('Spatial Averaging (Mean) Filtering with Different
Mask Sizes');
```

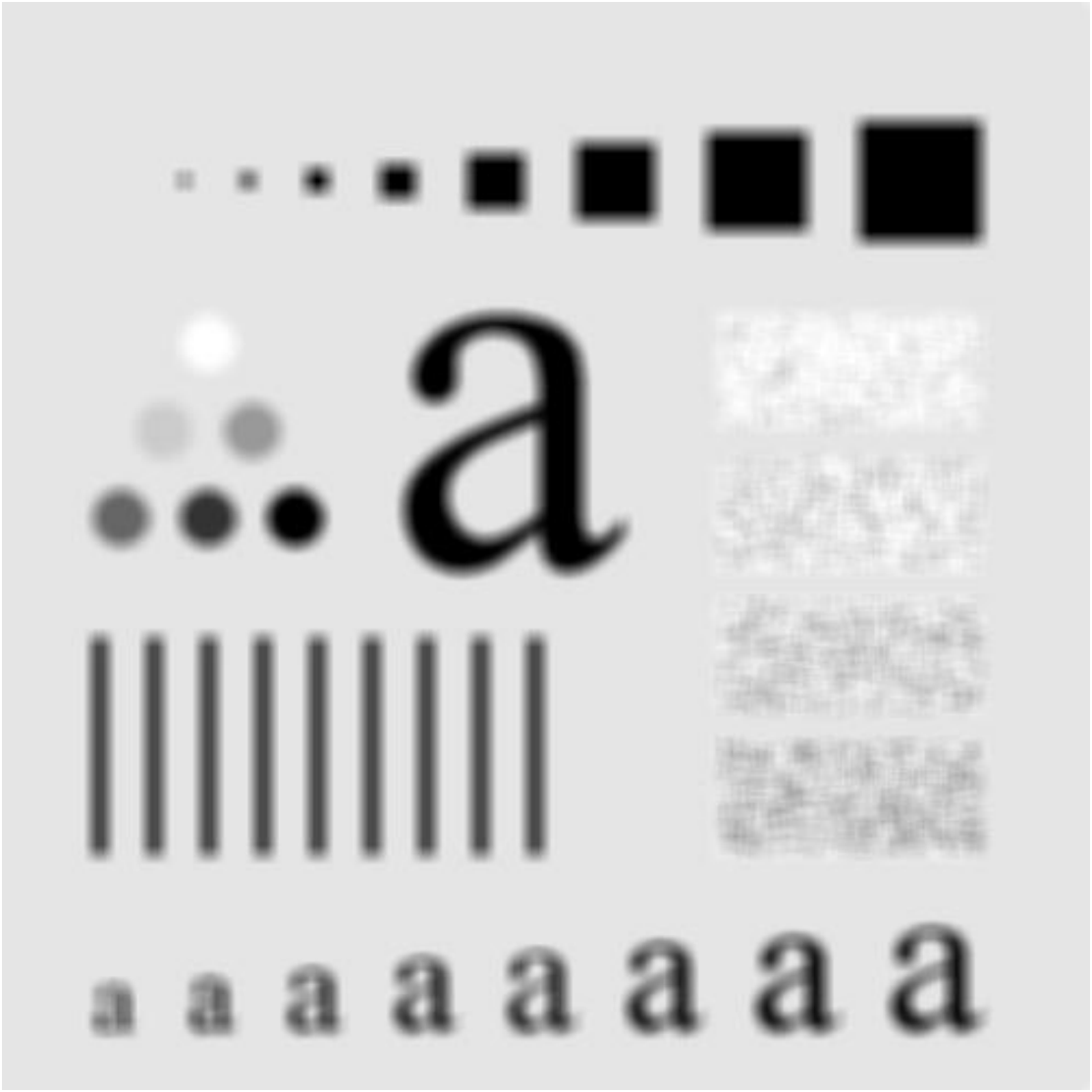
Result:



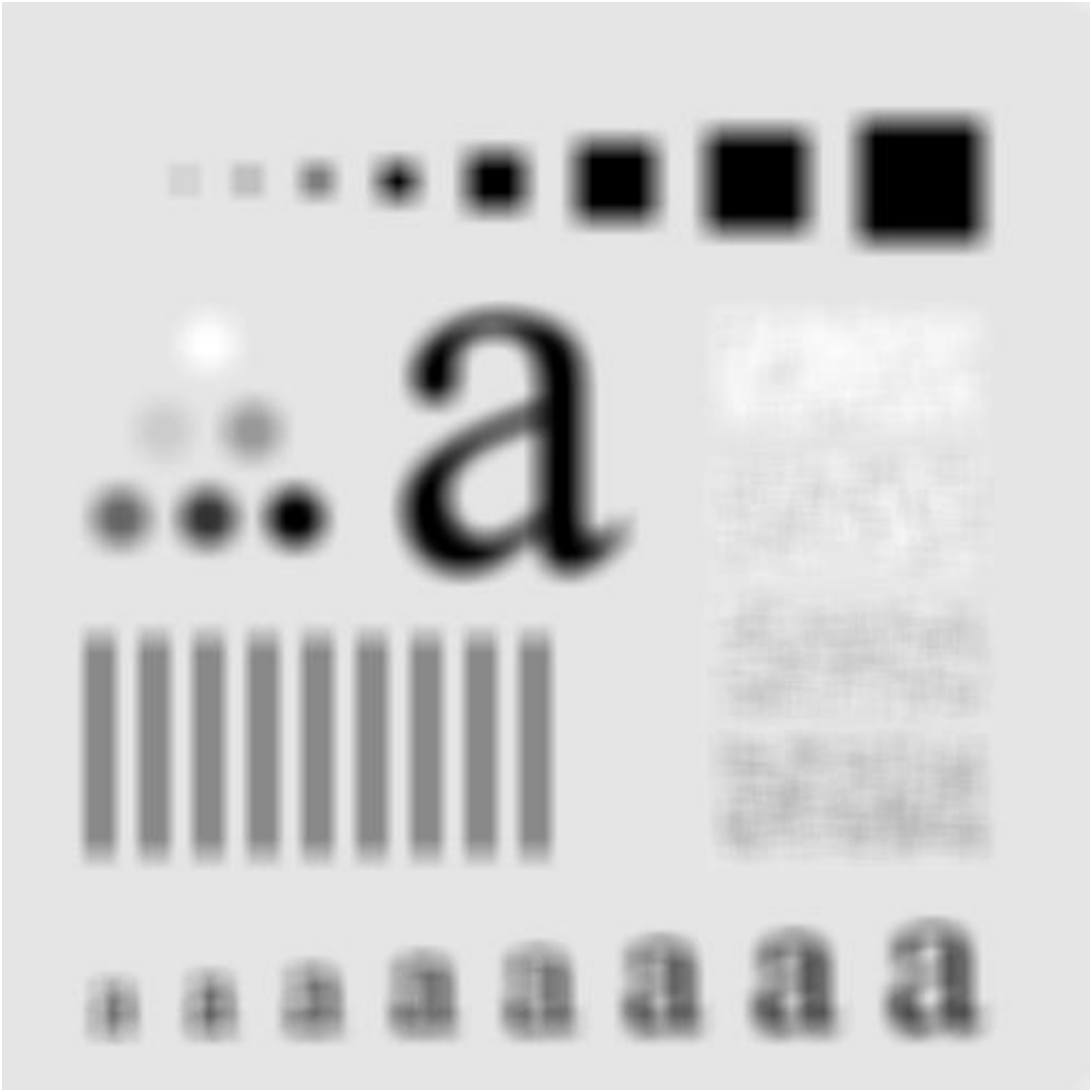
filtered_3x3

filtered_5x5

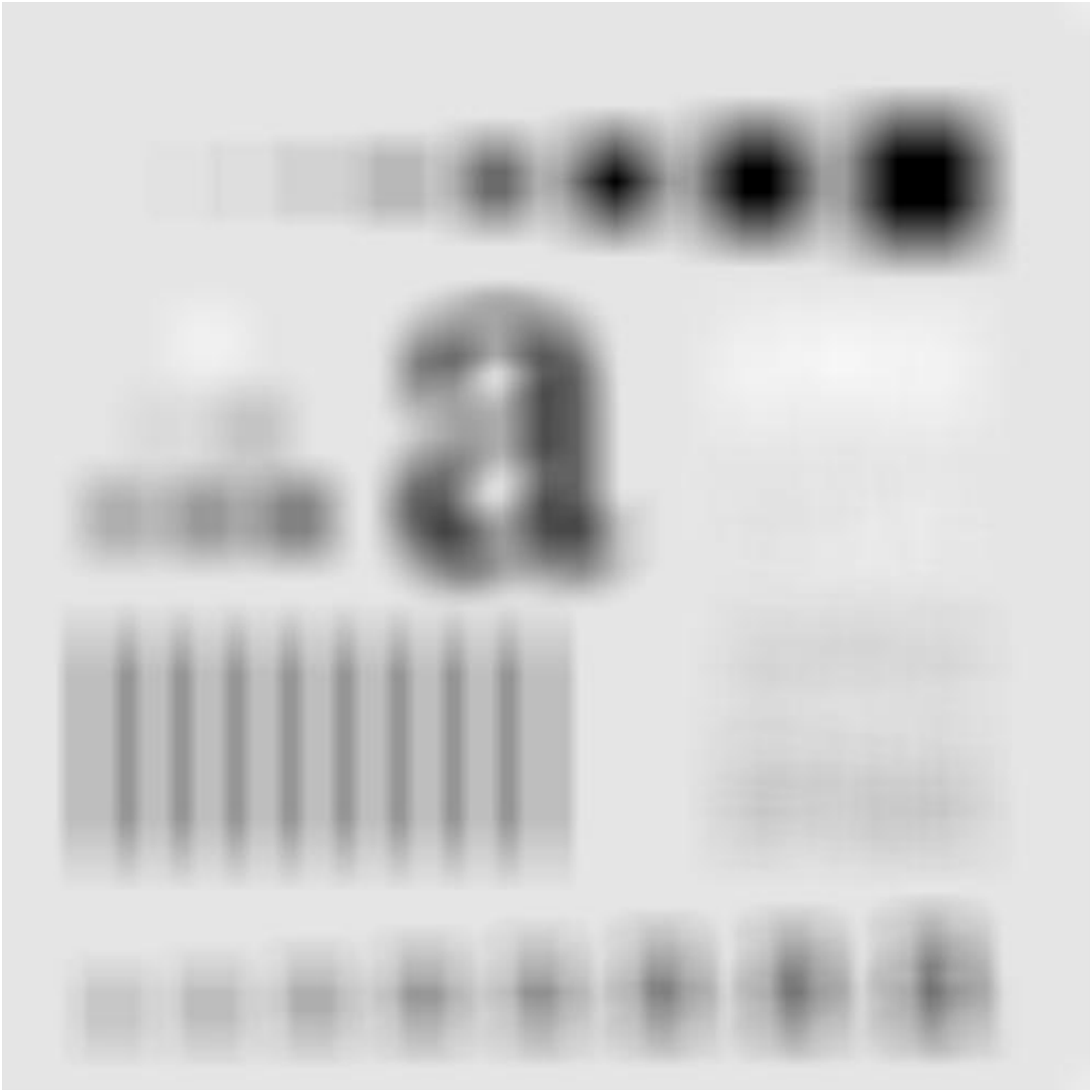
filtered_9x9



filtered_15x15



filtered_35x35



In this project, I used mean smoothing filters with varying mask sizes (3x3, 5x5, 9x9, 15x15, and 35x35) to enhance the image "Fig_test_pattern_blurring_orig.tif". From the results, I observed that increasing the mask size resulted in greater smoothing of the image but also caused a significant loss of details.

When using a 3x3 mask, the image retained most of its details while achieving moderate noise reduction. However, as the mask size increased to 35x35, the image became heavily smoothed, with fine details blurred

almost completely. This highlights that while mean filters are effective at reducing noise, they are not detail-preserving, especially when larger masks are used. As such, mean filtering is better suited for images with minimal detail requirements but significant noise. The choice of mask size must balance between noise suppression and detail retention based on the specific application.

Project03-08:

Code:

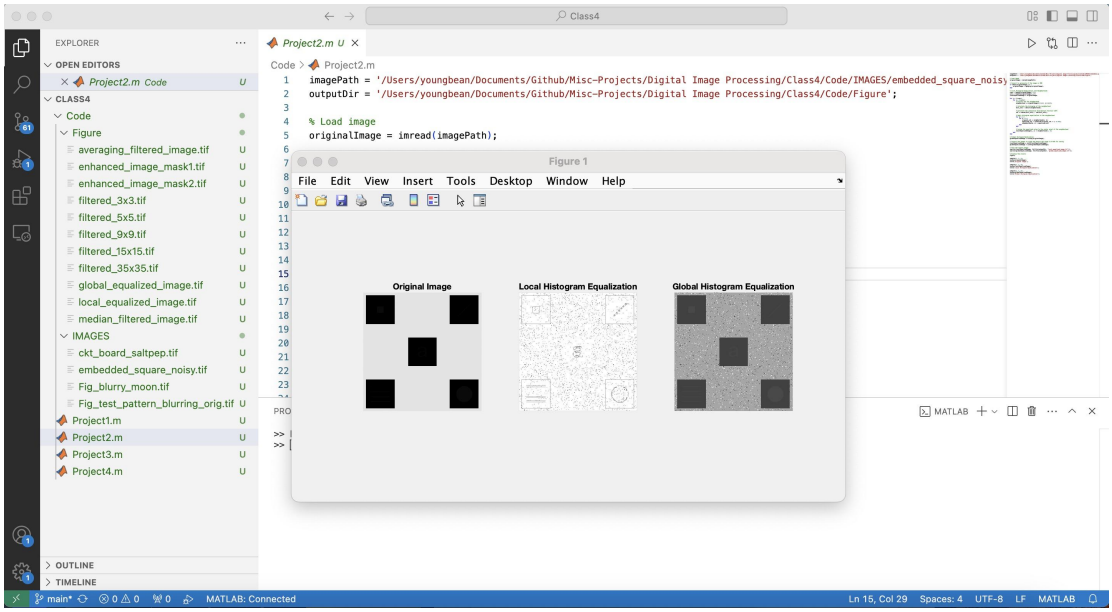
Project2.m:

```
imagePath =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image
Processing/Class4/Code/IMAGES/embedded_square_noisy.tif';
outputDir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class4/Code/Figure';

% Load image
originalImage = imread(imagePath);

% Convert to grayscale if the image is RGB
if ndims(originalImage) == 3
originalImage = rgb2gray(originalImage);
end

% Local Histogram Equalization (3x3 Neighborhood)
rows = length(originalImage(:,1));
cols = length(originalImage(1,:));
localEqualizedImage = originalImage;
```

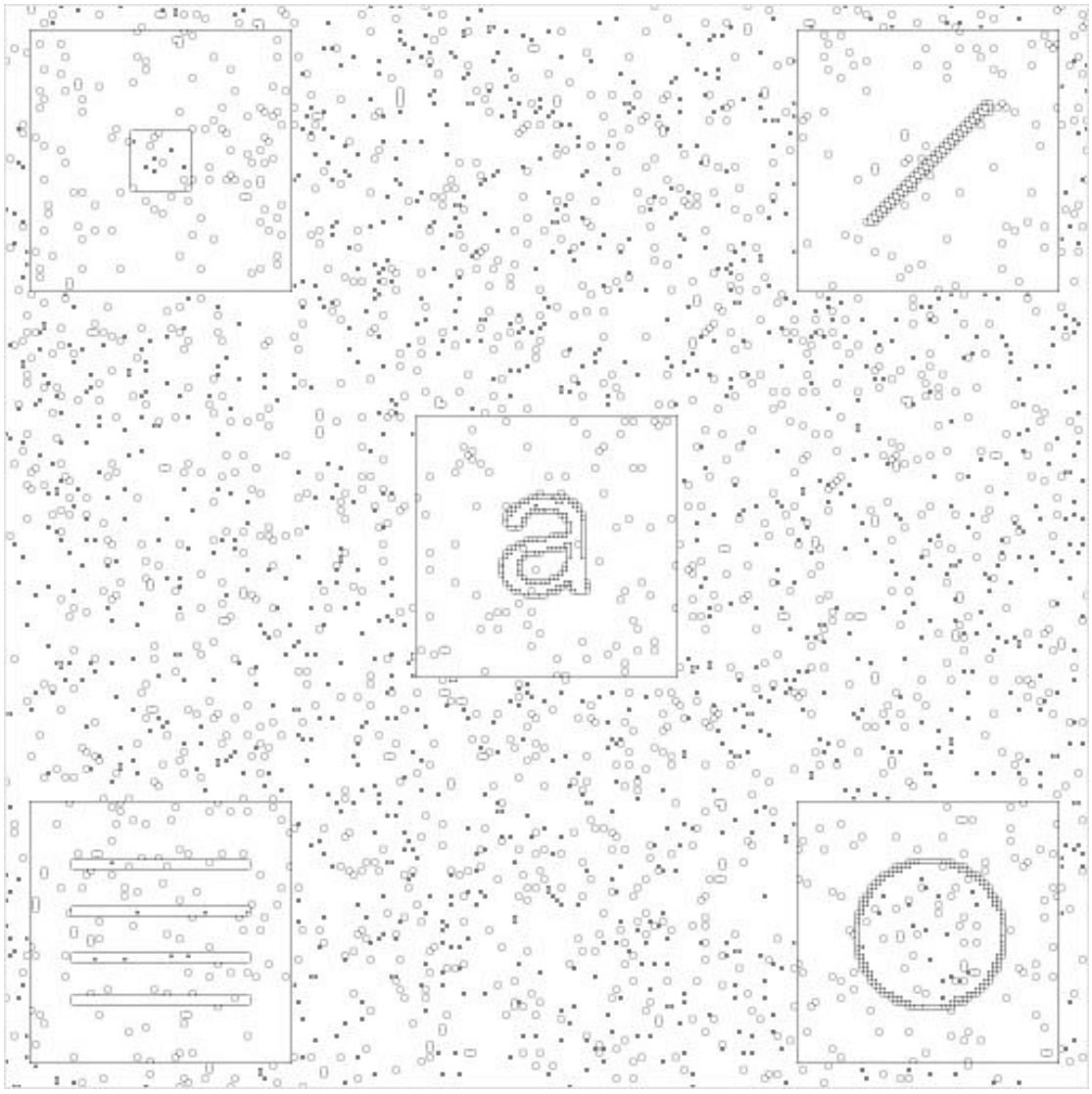
```
for i = 2:rows-1
for j = 2:cols-1
% Extract the 3x3 neighborhood
neighborhood = originalImage(i-1:i+1, j-1:j+1);
% Calculate the histogram of the neighborhood
hist_vals = imhist(neighborhood);
% Calculate the cumulative distribution function (CDF)
cdf = cumsum(hist_vals) / sum(hist_vals);
% Apply histogram equalization to the neighborhood
for m = 1:3
for n = 1:3
original_val = neighborhood(m, n);
equalized_val = round(cdf(original_val + 1) * 255);
neighborhood(m, n) = equalized_val;
end
end
% Assign the equalized value to the center pixel of the
neighborhood
localEqualizedImage(i, j) = neighborhood(2, 2);
end
end

% Global Histogram Equalization
globalEqualizedImage = histeq(originalImage);

% Convert the images to uint8 and ensure the range is 0-255
for saving
localEqualizedImage = uint8(localEqualizedImage);
globalEqualizedImage = uint8(globalEqualizedImage);

% Save the output images
imwrite(localEqualizedImage, fullfile(outputDir,
'local_equalized_image.tif'));
imwrite(globalEqualizedImage, fullfile(outputDir,
'global_equalized_image.tif'));

% Display the results
figure;
```

```
subplot(1, 3, 1);
imshow(originalImage);
title('Original Image');

subplot(1, 3, 2);
imshow(localEqualizedImage);
title('Local Histogram Equalization');

subplot(1, 3, 3);
imshow(globalEqualizedImage);
title('Global Histogram Equalization');
```

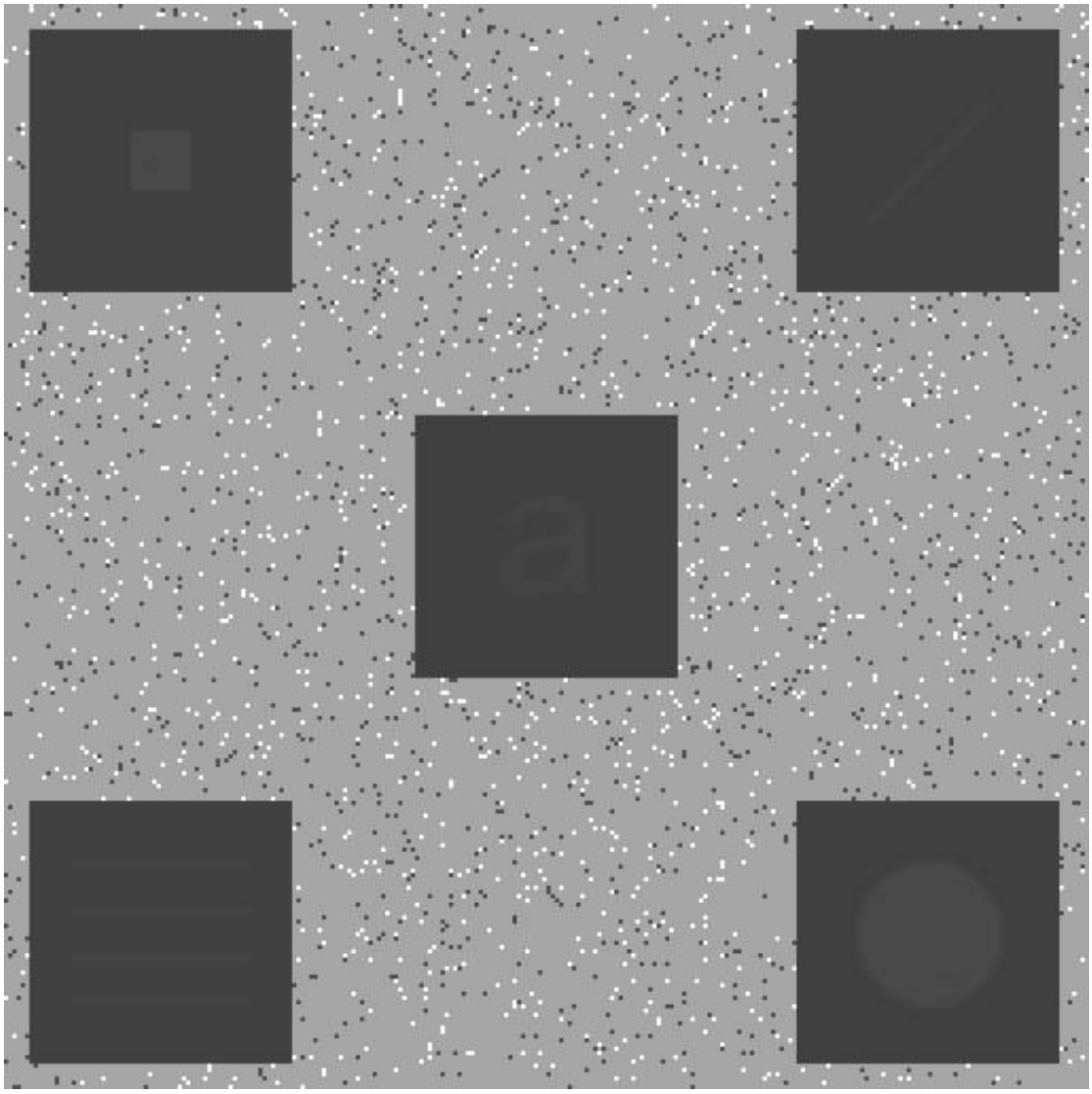
Result:



local_equalized_image



global_equalized_image



In this project, I applied both local and global histogram equalization to the image “embedded_square_noisy”. From the results, local histogram equalization proved to be superior in enhancing local contrast and preserving details, while global histogram equalization worked better for overall contrast enhancement.

Local histogram equalization divides the image into smaller regions (3x3 neighborhoods in this case) and performs histogram equalization independently for each region. This localized operation allowed for better

contrast improvement in small, detailed regions, but it also introduced slight inconsistencies in overall brightness across the image. On the other hand, global histogram equalization processed the entire image as a whole, providing uniform brightness and improved overall contrast. However, it failed to preserve fine details and did not handle localized noise as effectively.

Thus, local histogram equalization is more effective for enhancing images where detail preservation is critical, while global histogram equalization is better suited for images with overall low contrast.

Project03-09：

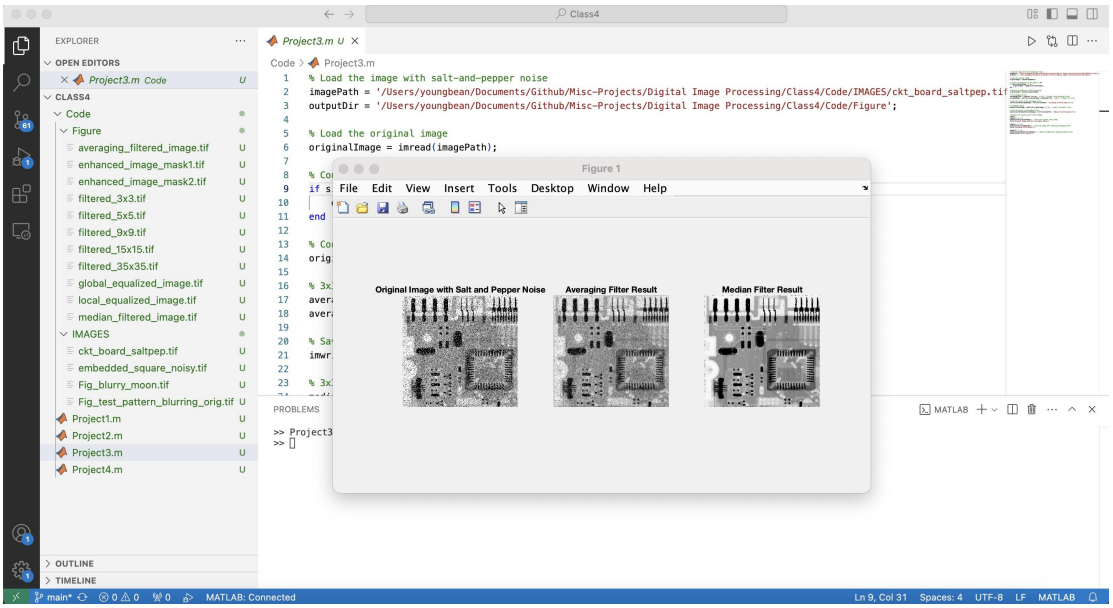
Code:

Project3.m:

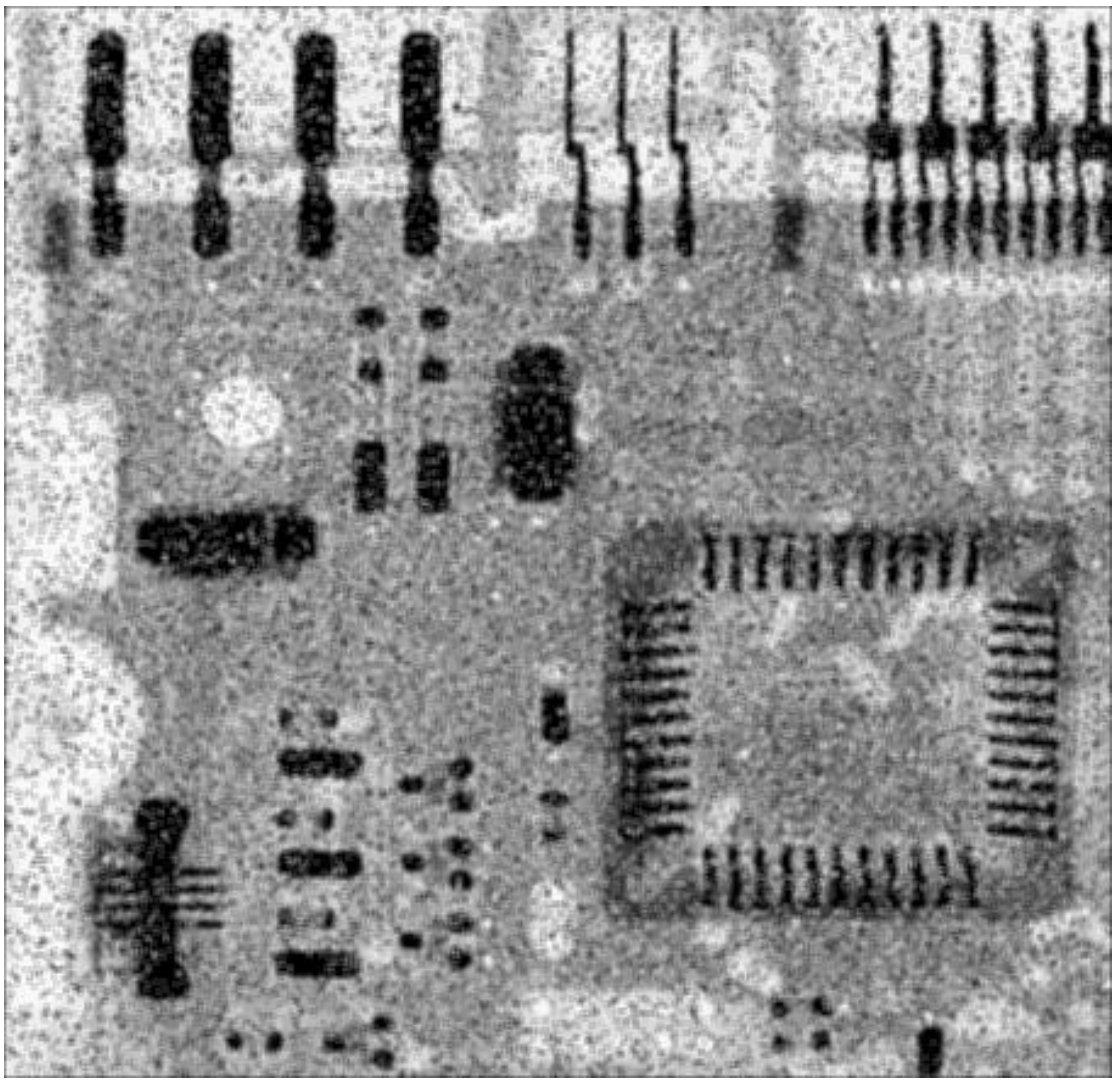
```
% Load the image with salt-and-pepper noise
imagePath =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image
Processing/Class4/Code/IMAGES/ckt_board_saltpep.tif';
outputDir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class4/Code/Figure';

% Load the original image
originalImage = imread(imagePath);

% Convert to grayscale if the image is RGB
if size(originalImage, 3) == 3
originalImage = rgb2gray(originalImage);
```

```
end

% Convert to double for filter processing
originalImage = double(originalImage);

% 3x3 Averaging Filter
averagingFilter = fspecial('average', [3 3]); % Create a 3x3
averaging filter
averagedImage = imfilter(originalImage, averagingFilter,
'same'); % Apply the filter

% Save the result of the averaging filter
imwrite(uint8(averagedImage), fullfile(outputDir,
'averaging_filtered_image.tif'));

% 3x3 Median Filter
medianFilteredImage = medfilt2(originalImage, [3 3]); %
Apply the median filter

% Save the result of the median filter
imwrite(uint8(medianFilteredImage), fullfile(outputDir,
'median_filtered_image.tif'));

% Display the original and filtered images
figure;
subplot(1, 3, 1);
imshow(uint8(originalImage)); % Show the original noisy
image
title('Original Image with Salt and Pepper Noise');

subplot(1, 3, 2);
imshow(uint8(averagedImage)); % Show the image after
applying averaging filter
title('Averaging Filter Result');

subplot(1, 3, 3);
imshow(uint8(medianFilteredImage)); % Show the image after
applying median filter
title('Median Filter Result');
```

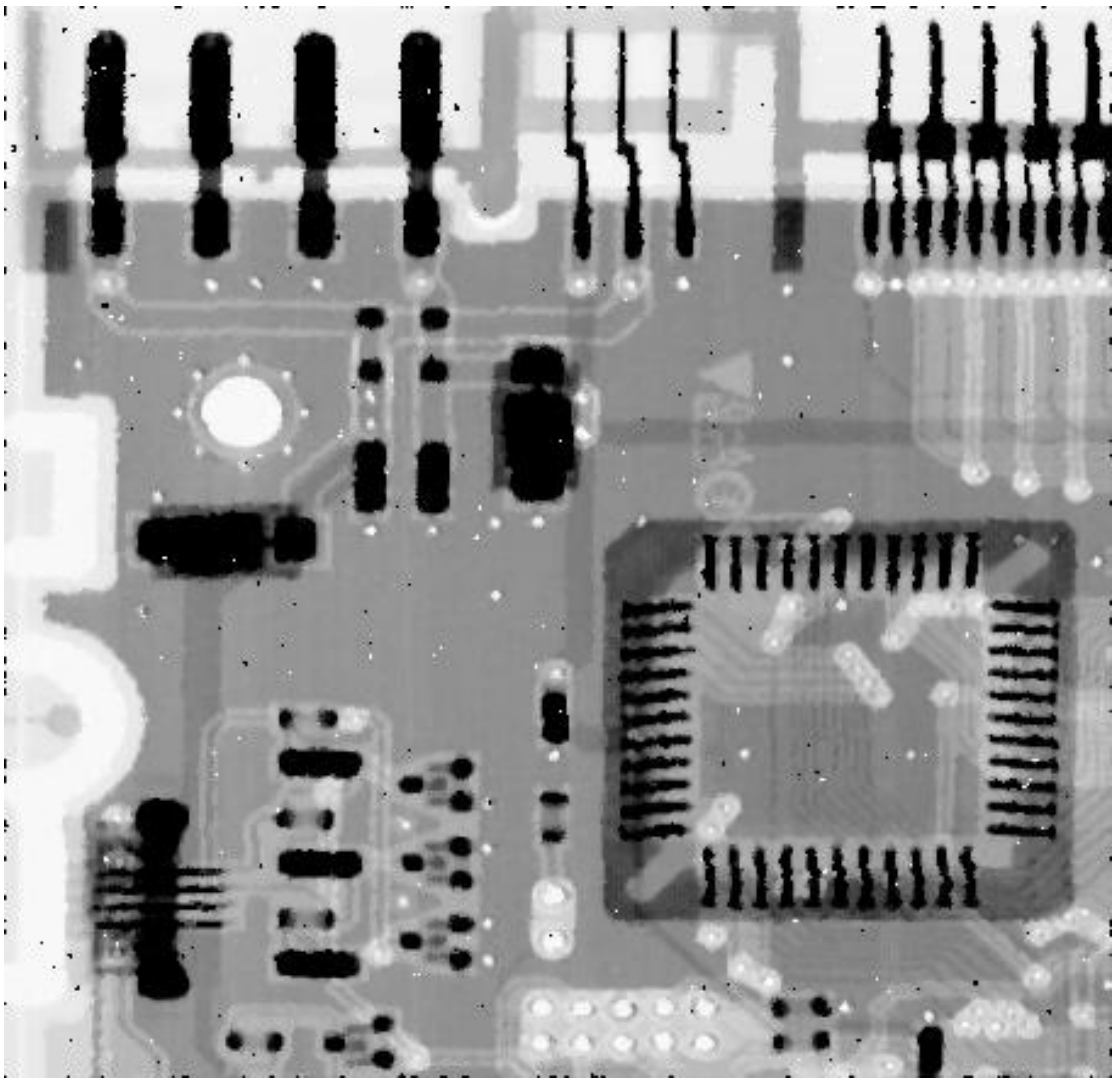
Result:



averaging_filtered_image



median_filtered_image



In this project, I applied both a 3x3 averaging filter and a 3x3 median filter to the salt-and-pepper noise-corrupted image “ckt_board_saltpep”. The results revealed that while both filters reduced noise, the median filter significantly outperformed the averaging filter in removing salt-and-pepper noise and preserving edges.

The averaging filter computes the mean of the neighborhood pixels, which is sensitive to extreme values (e.g., salt-and-pepper noise). This resulted in residual noise and blurring of edges in the filtered image. In contrast, the median filter replaces each pixel with the median value of its

neighborhood, effectively removing the extreme values associated with salt-and-pepper noise. As a result, the median filter produced a cleaner image while retaining sharp edges.

In conclusion, while the averaging filter can smooth the image, it is less effective for removing salt-and-pepper noise. The median filter is the preferred choice for such noise types, as it preserves edges and eliminates outliers effectively.

Project03-10：

Code:

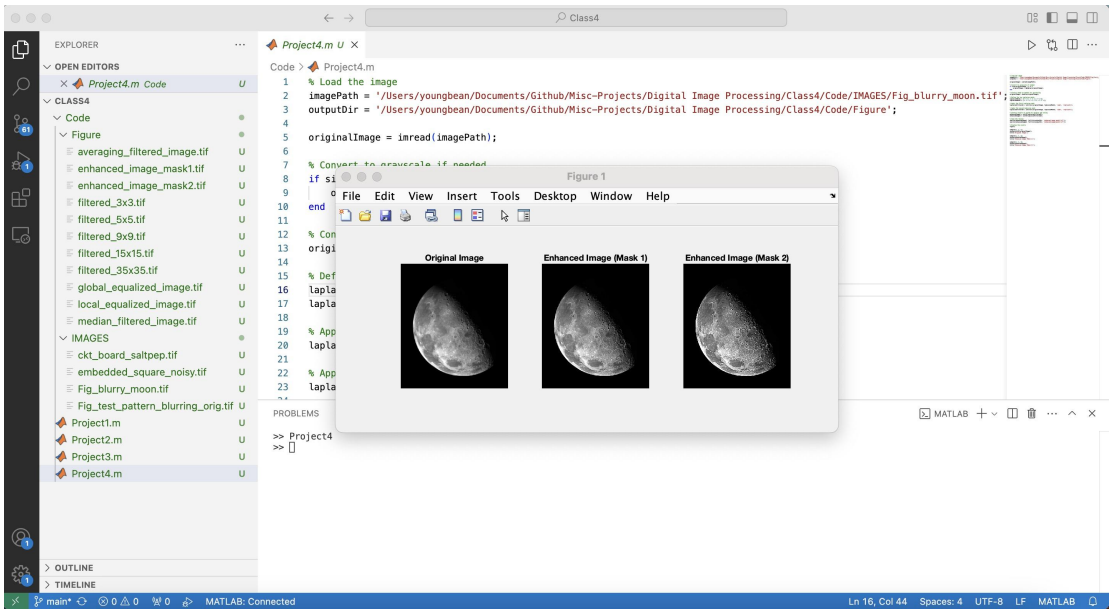
Project4.m:

```
% Load the image
imagePath =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class4/Code/IMAGES/Fig_blurry_moon.tif';
outputDir =
'/Users/youngbean/Documents/Github/Misc-Projects/Digital
Image Processing/Class4/Code/Figure';

originalImage = imread(imagePath);

% Convert to grayscale if needed
if size(originalImage, 3) == 3
originalImage = rgb2gray(originalImage);
end

% Convert image to double for processing
originalImage = double(originalImage);

% Define the two Laplacian masks
```

```
laplacianMask1 = [0 -1 0; -1 5 -1; 0 -1 0];
laplacianMask2 = [-1 -1 -1; -1 9 -1; -1 -1 -1];

% Apply the first Laplacian mask
laplacianFiltered1 = imfilter(originalImage, laplacianMask1,
'same', 'replicate');

% Apply the second Laplacian mask
laplacianFiltered2 = imfilter(originalImage, laplacianMask2,
'same', 'replicate');

% Convert results to uint8 for display and saving
enhancedImage1 = uint8(laplacianFiltered1);
enhancedImage2 = uint8(laplacianFiltered2);

% Save the results
imwrite(enhancedImage1, fullfile(outputDir,
'enhanced_image_mask1.tif'));
imwrite(enhancedImage2, fullfile(outputDir,
'enhanced_image_mask2.tif'));

% Display the results
figure;

subplot(1, 3, 1);
imshow(uint8(originalImage));
title('Original Image');

subplot(1, 3, 2);
imshow(enhancedImage1);
title('Enhanced Image (Mask 1)');

subplot(1, 3, 3);
imshow(enhancedImage2);
title('Enhanced Image (Mask 2)');
```

Result:

Digital Image Processing-Lab 4



enhanced_image_mask1

enhanced_image_mask2

In this project, I applied two different Laplacian masks to enhance the blurry image “Fig_blurry_moon.tif”. Both masks successfully enhanced the image by increasing edge sharpness, but their results differed significantly.

• **Mask 1**:

This mask provided a smoother enhancement by slightly emphasizing the edges while preserving the overall structure of the image. It introduced minimal noise amplification, making it suitable for images with low levels of noise.

• **Mask 2**:

This mask produced a more aggressive enhancement, significantly sharpening the edges. However, it also amplified noise and created slight artifacts, making it less suitable for images with high noise levels.

In summary, Mask 1 is better suited for subtle edge enhancement with minimal noise amplification, while Mask 2 is more effective for strong edge enhancement but may require additional noise handling. The choice of mask depends on the specific requirements of the image enhancement task.

【小结或讨论】

在本次实验中，我通过 MATLAB 编程实现了多种基于空域模板的图像增强技术，并对不同方法的效果进行了对比分析。

空域滤波与均值平滑滤波

对于 **Project 03-07**，我使用均值平滑滤波对原始图像进行了增强。均值滤波器的核心思想是将模板内的像素值取平均，公式如下：

$$g(x,y) = \frac{1}{n \times n} \sum_{(s,t) \in \text{Template}} f(x+s, y+t)$$

其中, $f(x,y)$ 是原始图像, $g(x,y)$ 是滤波后的图像, $n \times n$ 为模板大小。

通过改变模板的大小（如 3x3、5x5 等），我发现模板越大，平滑效果越显著，但也会导致图像细节被模糊。小模板可以在一定程度上平滑噪声，但不能完全消除模糊现象。

局部直方图均衡化与全局直方图均衡化

在 **Project 03-08** 中，我分别实现了局部直方图均衡化和全局直方图均衡化。直方图均衡化的数学原理是利用累计分布函数（CDF）对像素值进行重新映射:

$$s = (L-1) \times \text{CDF}(r)$$

其中, r 为输入像素值, s 为输出像素值, L 是灰度级的总数, $\text{CDF}(r)$ 是归一化后的累计直方图。

局部直方图均衡化在每个 3x3 邻域内对直方图进行均衡处理，增强了局部对比度;全局均衡化则对整幅图像进行处理,提升整体对比度。实验结果表明，局部均衡化可以更好地保留图像细节，而全局均衡化在整体对比度较低时效果更明显。

均值滤波与中值滤波

在 **Project 03-09** 中，我对含有椒盐噪声的图像分别应用了均值滤波和中值滤波。均值滤波的计算公式与 **Project 03-07** 一致，而中值滤波的计算方法是将模板内的像素值取中值:

$$g(x,y) = \text{Median}\{f(x+s,y+t)|(s,t) \in \text{Template}\}$$

实验结果表明，均值滤波对椒盐噪声的去除效果较差，因为均值会被极值（黑点或白点）拉低或拉高，导致残留噪声；中值滤波在去除椒盐噪声方面效果显著，并且能够较好地保留边缘信息。

拉普拉斯滤波增强

在 **Project 03-10** 中，我使用两种拉普拉斯模板对模糊图像进行了增强。拉普拉斯滤波的数学形式为:

$$g(x,y) = f(x,y) - \nabla^2 f(x,y)$$

其中，$\nabla^2 f(x,y)$ 是图像的拉普拉斯算子。

模板 1:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

模板 2:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

两种模板的实验结果表明，模板 1 的增强效果较为平滑，适合强调边缘的同时避免噪声放大；而模板 2 的增强效果更为强烈，能突出边缘，但可能引入伪影和噪声。

本次实验让我加深了对图像增强技术的理解。空域滤波作为一种基础的增强方法，主要用于平滑图像并降低噪声，但其对细节的处理能力较为有限，容易导致图像模糊。直方图均衡化则是一种有效提升图像对比度的技术，其中全局直方图均衡化适合整体对比度较低的图像处理，而局部直方图均衡化由于在局部范围内进行操作，能够更好地保留图像的细节信息和局部特征。对于含有椒盐噪声的图像，中值滤波表现出了比均值滤波更优越的去噪能力，因为中值滤波通过取邻域像素的中值，能够有效消除椒盐噪声并同时保留边缘信息，而均值滤波容易受到极值点的影响，导致残留噪声或模糊边缘。在拉普拉斯增强技术中，我尝试了两种不同的拉普拉斯模板。结果显示，拉普拉斯增强能够有效地提升图像边缘的对比度，其中不同的模板可以实现不同程度的增强效果，模板 1 增强较为平滑，适合边缘提取和噪声较低的图像，而模板 2 的增强更为强烈，能突出细节但可能引入伪影和噪声。通过实验，我不仅掌握了这些图像增强方法的原理与实现，还能够根据不同图像的特性选择适合的技术，为后续的图像处理与分析提供了重要参考。