Report Computer Vision

Rawan Farouk 16001938

October 25, 2024

1 Canny Edge Detection

I chose the Canny Edge Detection method. Canny is a popular multi-stage edge detection algorithm known for its ability to detect a wide range of edges in images with high accuracy.

1.1 Why I chose it?

One of the primary reasons for selecting the Canny method is its ability to minimize false edges while preserving true edges, which provides a clear and refined detection of the edges in complex landscapes, such as the given grayscale image. Additionally, Canny offers adjustable thresholds that allow for precise control over the sensitivity of the edge detection process, making it versatile for various image types.

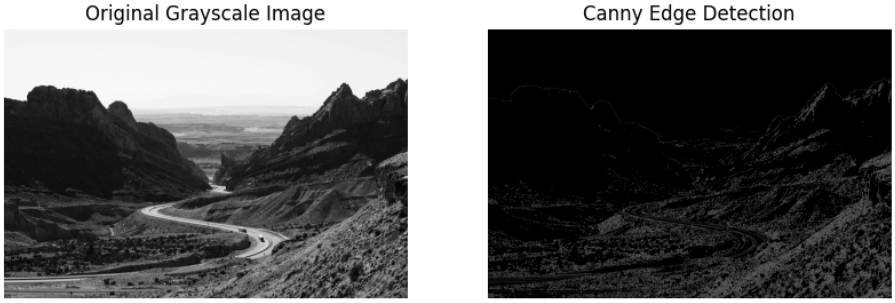


Figure 1: Original vs Canny Edge Detection

2 Types of Noises - Average Filtering

2.1 Salt and Pepper Noise

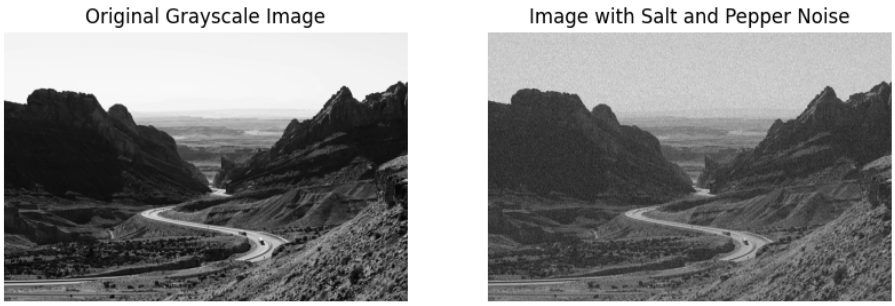


Figure 2: Original vs Salt and Pepper Noise

2.1.1 Average Filter 3x3 - 5x5 - 9x9

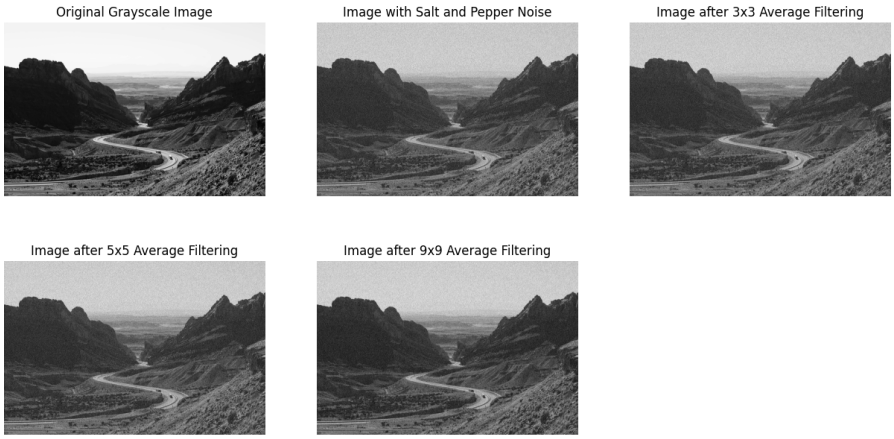


Figure 3: Original vs Salt and Pepper Noise vs Average Filtering

The effectiveness of noise reduction and image quality vary with different filter sizes (3x3, 5x5, and 9x9) used for Salt and Pepper noise reduction. The 3x3 filter provides minimal noise reduction but retains most of the image detail, making it suitable for areas requiring fine details, though it allows some noise to remain. The 5x5 filter offers a balance between noise suppression and detail preservation, producing smoother images with moderate noise removal. The 9x9 filter effectively reduces noise but significantly blurs fine details, making it less ideal for images where maintaining detail is important. Smaller filters preserve detail, while larger filters are better for noise removal.

2.2 Gaussian Noise

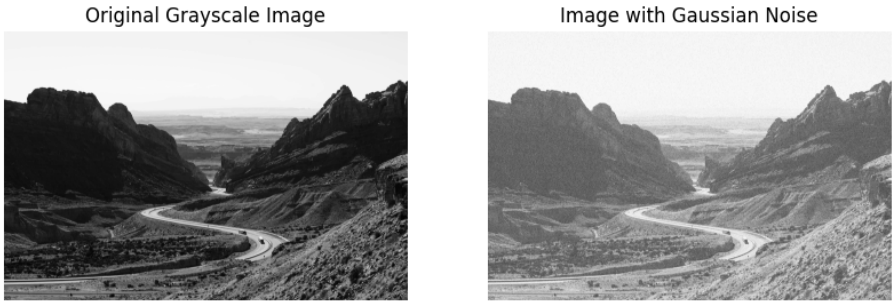


Figure 4: Original vs Guassian Noise

2.2.1 Average Filter 3x3 - 5x5 - 9x9

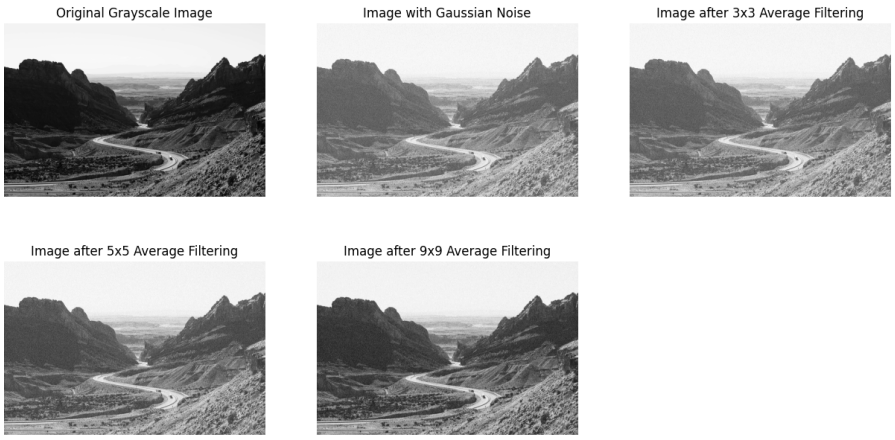


Figure 5: Original vs Gaussian Noise vs Average Filtering

In cases where an image is affected by Gaussian noise, the size of the filter significantly impacts noise reduction and detail preservation. A 3x3 filter offers minimal noise reduction while preserving fine details, allowing some noise to remain visible. This is useful when retaining image details is more important than removing noise. The 5x5 filter strikes a balance between reducing noise and preserving detail, producing a smoother image but introducing slight blurring. The 9x9 filter removes the most noise but at the cost of losing fine details due to increased blurring. It is most suitable when noise reduction is prioritized over maintaining image features.

2.3 Poisson Noise

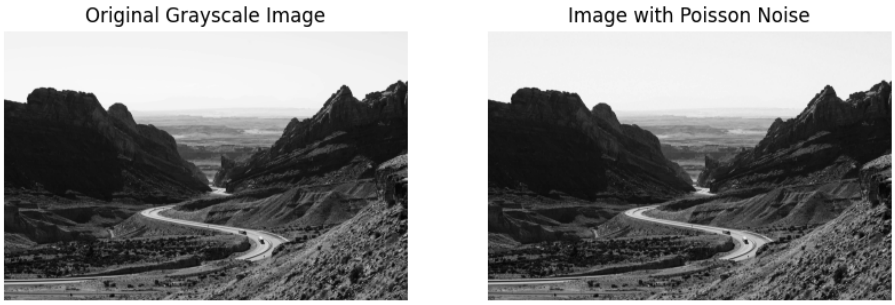


Figure 6: Original vs Poisson Noise

2.3.1 Average Filter 3x3 - 5x5 - 9x9

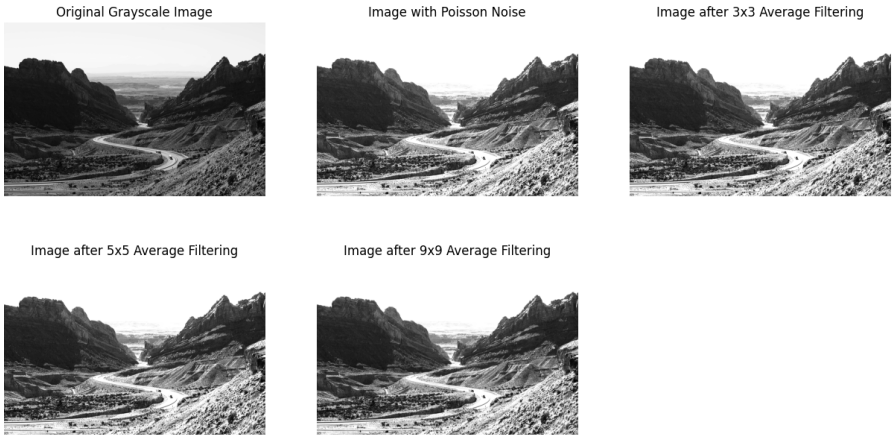


Figure 7: Original vs Poisson Noise vs Average Filtering

When dealing with Poisson noise, the performance of average filters (3x3, 5x5, 9x9) varies with their size. The 3x3 filter provides minimal noise reduction, preserving fine details while leaving noticeable noise in high-intensity areas. It offers a balance between smoothing and detail retention. The 5x5 filter reduces Poisson noise more effectively, resulting in a smoother image, though some blurring of finer details starts to appear. The 9x9 filter is the most effective for noise reduction, significantly smoothing the image but causing substantial blurring, particularly in areas with sharp edges or fine textures. As the filter size increases, noise reduction improves, but at the cost of image sharpness and detail.

2.4 Random Noise

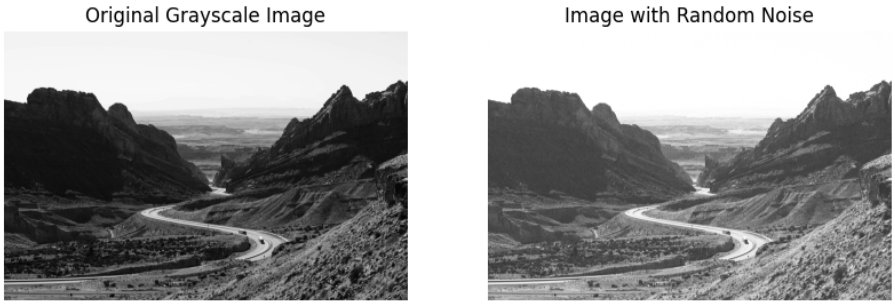


Figure 8: Original vs Random Noise

2.4.1 Average Filter 3x3 - 5x5 - 9x9

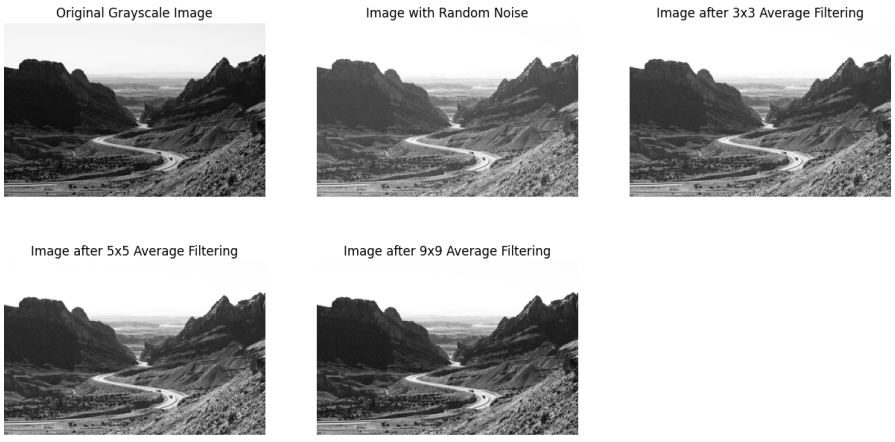


Figure 9: Original vs Random Noise vs Average Filtering

When Random Noise is applied to an image, the effectiveness of average filters depends on their size. The 3x3 filter offers minimal smoothing, reducing some noise while preserving fine details, but leaving noticeable noise. This filter is ideal for preserving details when noise reduction is less critical. The 5x5 filter achieves a better balance between noise reduction and detail retention, smoothing out more noise while maintaining reasonable image sharpness. The 9x9 filter is the most effective at removing random noise, producing a much smoother image, but with considerable blurring of fine details. Larger filters reduce more noise but sacrifice image sharpness and detail.

3 Analysis of the Effects of Different Noises and Filter Sizes on Image Quality

When the various types of noise application—Salt and Pepper, Gaussian, Poisson, and Random—are applied to the image, each affects the quality of the image differently; then the application of averaging filtering with variable filter size again changes the result.

1. **Salt and Pepper Noise:** It is a kind of noise that adds sharp black and white pixels randomly in the image. It creates huge contrasts, hence affecting the quality of the image very much.

    - **3x3 filter:** This cleans off some of the noise but still retains much of the details, so that most of the noise artifacts remain visible.
    - **5x5 filter:** It smoothes out more of the noise but maintains reasonable sharpness; it balances noise reduction and detail pretty well.
    - **9x9 filter:** Noise is largely removed, but there is marked blurring of the finer details, which makes the image overly smooth.

2. **Gaussian Noise:** Happens to be less heavy than Salt and Pepper noise. The noise generally spreads in the image in a uniform manner, providing a grainy look to the image.

    - **3x3 filter:** A small smoothing out of noise and lots of the image details remain preserved.
    - **5x5 filter:** Smoothing out of noise is way more. The image is cleaner but at the cost of minor loss in details.
    - **9x9 filter:** This filter removes most of the Gaussian noise but, in turn, severely causes edge and fine structure blurring.

3. **Poisson Noise:** This noise, since it depends on the intensity of a pixel, adds small variations that increase in higher intensities. The noise will appear fewer in darker regions and more conspicuous in brighter regions.

    - **3x3 filter:** Grossly less noise reduction, leaving most of the Poisson noise still visible.
    - **5x5 filter:** Balances noise removal and detail preservation, notably reducing the noise in bright areas.
    - **9x9 filter:** Removes most of the Poisson noise at the cost of image details, giving a very smooth yet somewhat blurred image.

4. **Random Noise:** Random noise introduces uniformly distributed noise that appears chaotic across the image. It impacts all regions of the image equally.

    - **3x3 filter:** Has minimal smoothing with quite obvious random noise.
    - **5x5 filter:** Removes most of the noise without eliminating moderate details. It's a good balance between noise removal and sharpness.
    - **9x9 filter:** Removes nearly all the random noise but results in a heavy blur on the image, quite noticeable in regions of fine detail.

4 Overall Conclusions on the Impact of Different Noise Types and Filtering Techniques

Besides, the influence of various types of noise on image quality can be drastically different: Salt and Pepper noise results in sharp pixel-level distortions, Gaussian and Poisson noise cause more evenly distributed variations, and Random noise affects the whole image chaotically.

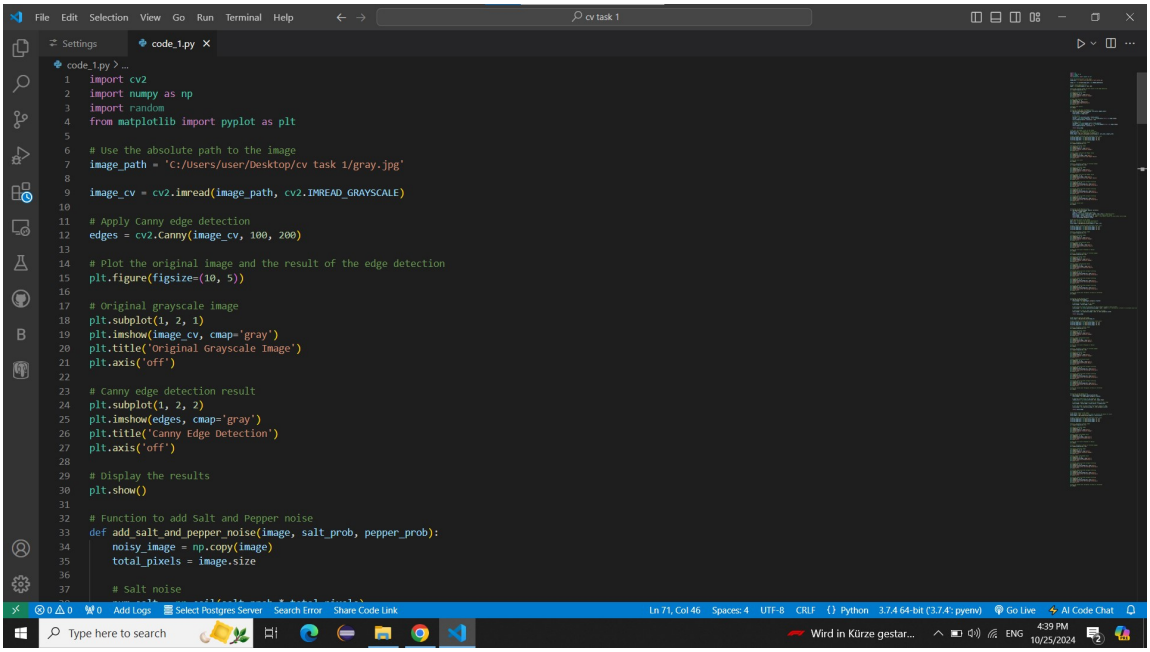
Average filtering is quite an efficient method of noise reduction; however, its efficiency directly depends on the size of the filter applied:

- With a small size, it provides basic noise reduction but maintains image details and can leave some noise artifacts.
- A medium size guarantees a good balance between noise reduction and keeping image sharpness, making this type of filter suitable for most applications.
- Large filters (9x9) are quite effective in removing noise, but this is often at the cost of losing a lot of image details and hence yielding very smooth, blurred images.

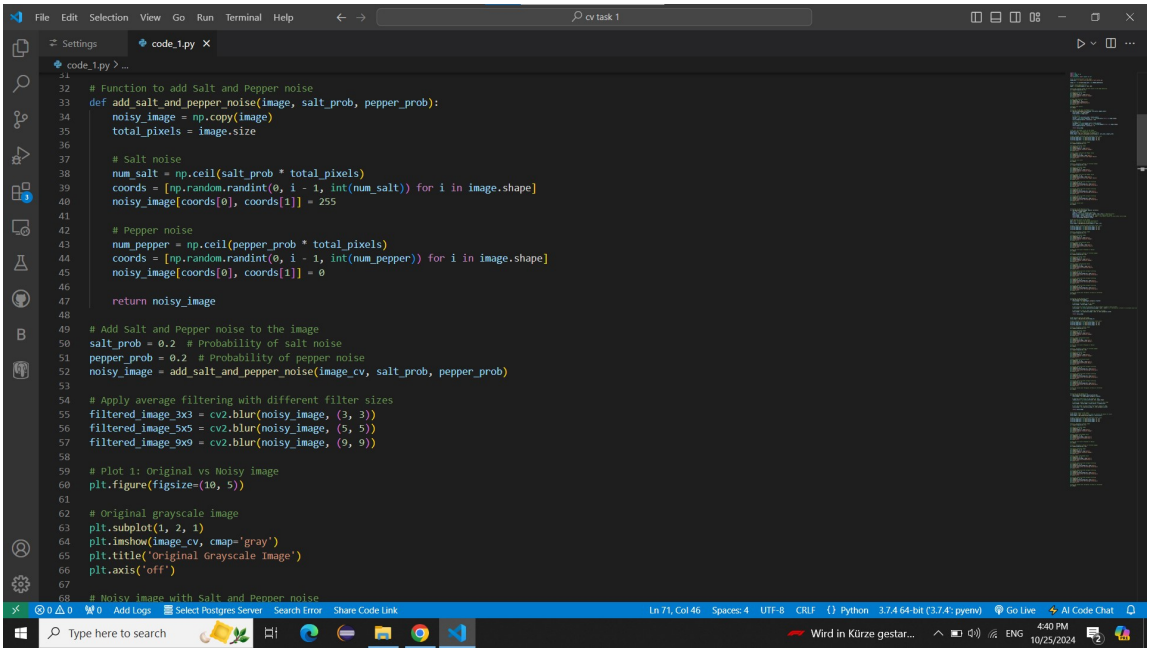
In summary, the nature of noise and size of filter selected should be compatible with the requirement on image quality. Smaller filters may be advisable when the preservation of detail is critical, while maximum noise reduction may call for larger filters, though at the expense of fine detail reduction. Different types of noise may also require different filtering strategies to achieve optimal results.

A Code Listing

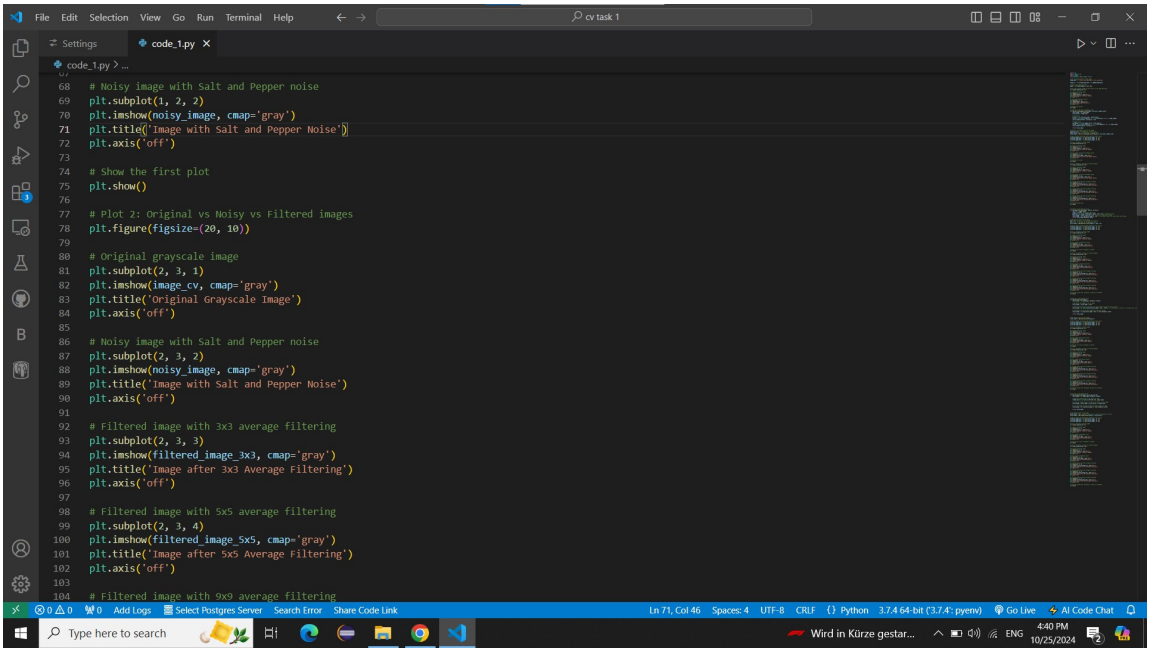
Here is the code as part of the appendix:



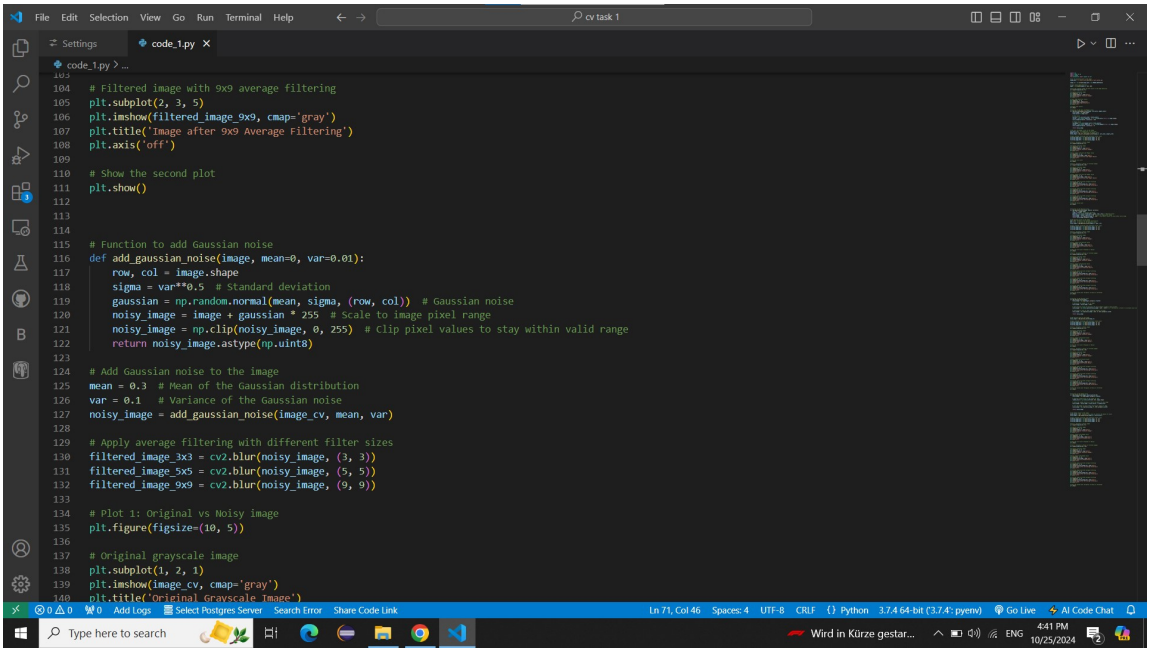
```python
import cv2
import numpy as np
import random
from matplotlib import pyplot as plt

# Use the absolute path to the image
image_path = 'C:/Users/user/Desktop/cv task 1/gray.jpg'

image_cv = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Apply Canny edge detection
edges = cv2.Canny(image_cv, 100, 200)

# Plot the original image and the result of the edge detection
plt.figure(figsize=(10, 5))

# Original grayscale image
plt.subplot(1, 2, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Canny edge detection result
plt.subplot(1, 2, 2)
plt.imshow(edges, cmap='gray')
plt.title('Canny Edge Detection')
plt.axis('off')

# Display the results
plt.show()

# Function to add Salt and Pepper noise
def add_salt_and_pepper_noise(image, salt_prob, pepper_prob):
    noisy_image = np.copy(image)
    total_pixels = image.size

    # Salt noise
```



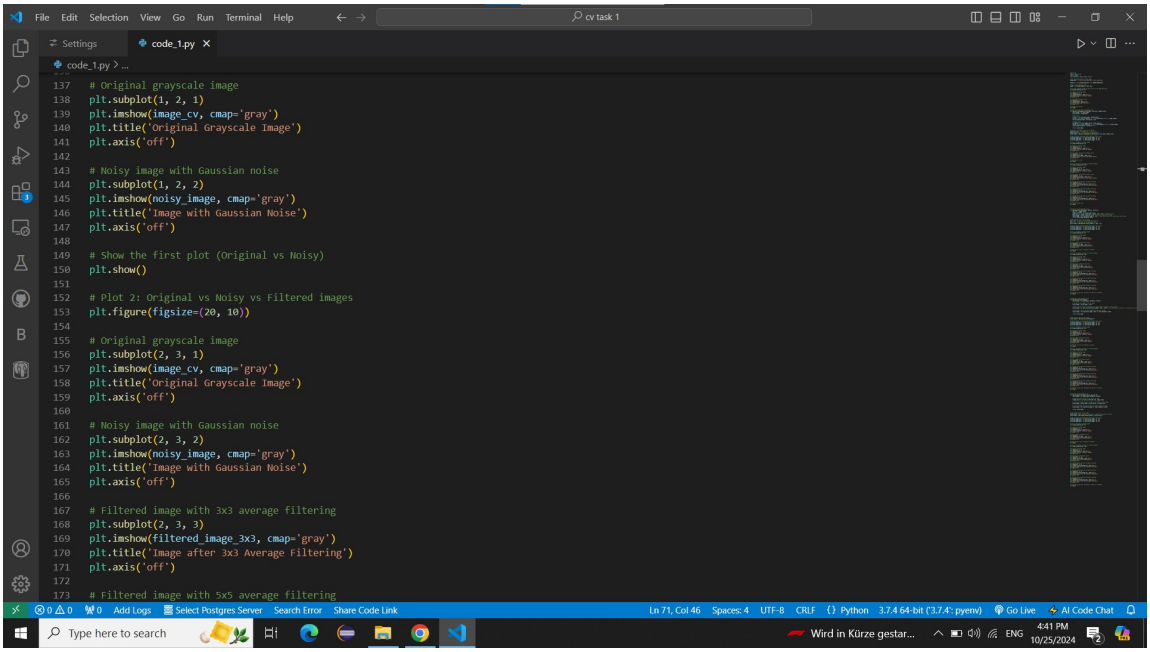
```python
# Function to add Salt and Pepper noise
def add_salt_and_pepper_noise(image, salt_prob, pepper_prob):
    noisy_image = np.copy(image)
    total_pixels = image.size

    # Salt noise
    num_salt = np.ceil(salt_prob * total_pixels)
    coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image.shape]
    noisy_image[coords[0], coords[1]] = 255

    # Pepper noise
    num_pepper = np.ceil(pepper_prob * total_pixels)
    coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in image.shape]
    noisy_image[coords[0], coords[1]] = 0

    return noisy_image

# Add Salt and Pepper noise to the image
salt_prob = 0.2  # Probability of salt noise
pepper_prob = 0.2  # Probability of pepper noise
noisy_image = add_salt_and_pepper_noise(image_cv, salt_prob, pepper_prob)

# Apply average filtering with different filter sizes
filtered_image_3x3 = cv2.blur(noisy_image, (3, 3))
filtered_image_5x5 = cv2.blur(noisy_image, (5, 5))
filtered_image_9x9 = cv2.blur(noisy_image, (9, 9))

# Plot 1: Original vs Noisy image
plt.figure(figsize=(10, 5))

# Original grayscale image
plt.subplot(1, 2, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Salt and Pepper noise
```



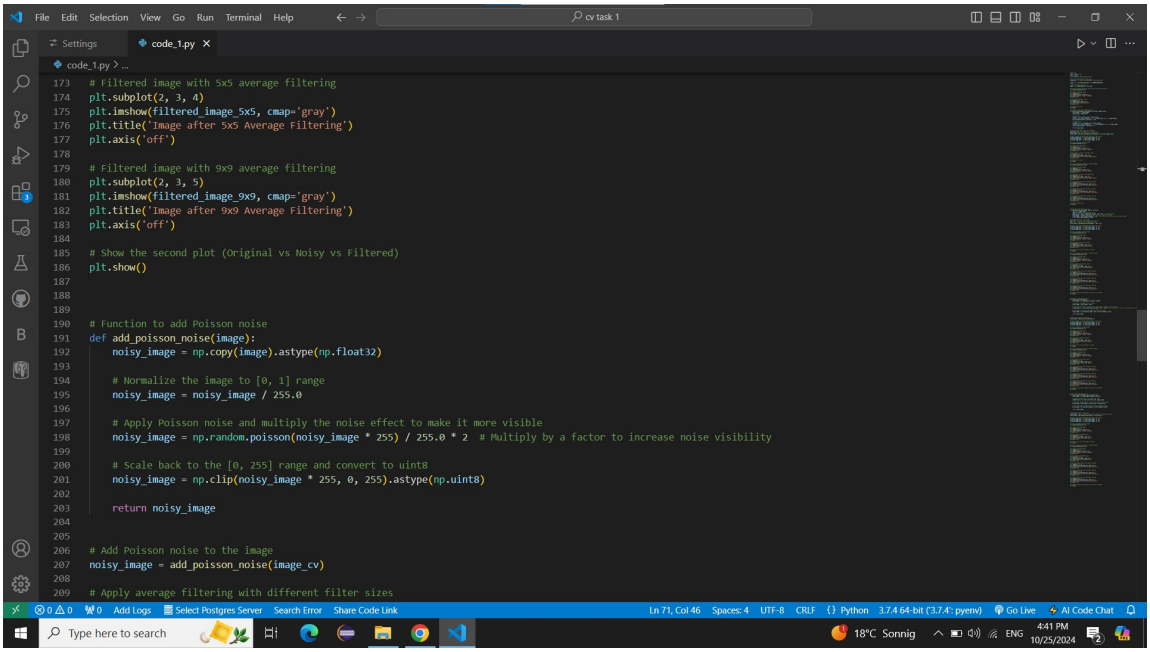
```
# Noisy image with Salt and Pepper noise
plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Salt and Pepper Noise')
plt.axis('off')

# Show the first plot
plt.show()

# Plot 2: Original vs Noisy vs Filtered images
plt.figure(figsize=(20, 10))

# Original grayscale image
plt.subplot(2, 3, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Salt and Pepper noise
plt.subplot(2, 3, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Salt and Pepper Noise')
plt.axis('off')

# Filtered image with 3x3 average filtering
plt.subplot(2, 3, 3)
plt.imshow(filtered_image_3x3, cmap='gray')
plt.title('Image after 3x3 Average Filtering')
plt.axis('off')

# Filtered image with 5x5 average filtering
plt.subplot(2, 3, 4)
plt.imshow(filtered_image_5x5, cmap='gray')
plt.title('Image after 5x5 Average Filtering')
plt.axis('off')

# Filtered image with 9x9 average filtering
```



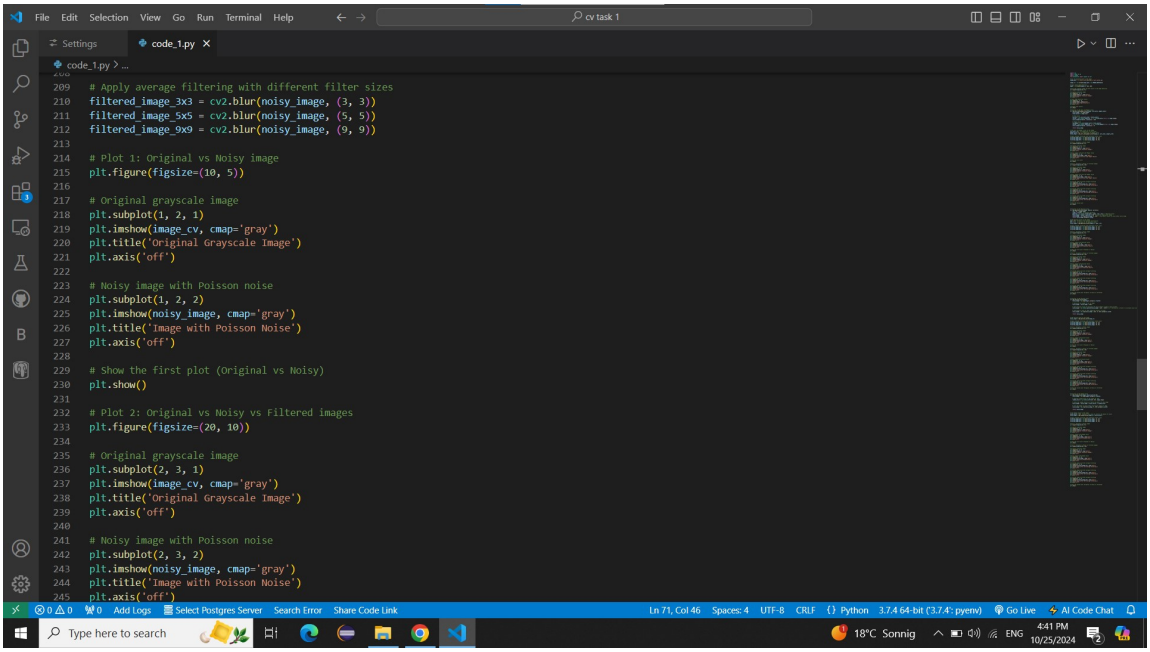
```
# Filtered image with 9x9 average filtering
plt.subplot(2, 3, 5)
plt.imshow(filtered_image_9x9, cmap='gray')
plt.title('Image after 9x9 Average Filtering')
plt.axis('off')

# Show the second plot
plt.show()



# Function to add Gaussian noise
def add_gaussian_noise(image, mean=0, var=0.01):
    row, col = image.shape
    sigma = var**0.5  # Standard deviation
    gaussian = np.random.normal(mean, sigma, (row, col))  # Gaussian noise
    noisy_image = image + gaussian * 255  # Scale to image pixel range
    noisy_image = np.clip(noisy_image, 0, 255)  # Clip pixel values to stay within valid range
    return noisy_image.astype(np.uint8)

# Add Gaussian noise to the image
mean = 0.3  # Mean of the Gaussian distribution
var = 0.1   # Variance of the Gaussian noise
noisy_image = add_gaussian_noise(image_cv, mean, var)

# Apply average filtering with different filter sizes
filtered_image_3x3 = cv2.blur(noisy_image, (3, 3))
filtered_image_5x5 = cv2.blur(noisy_image, (5, 5))
filtered_image_9x9 = cv2.blur(noisy_image, (9, 9))

# Plot 1: Original vs Noisy image
plt.figure(figsize=(10, 5))

# Original grayscale image
plt.subplot(1, 2, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
```



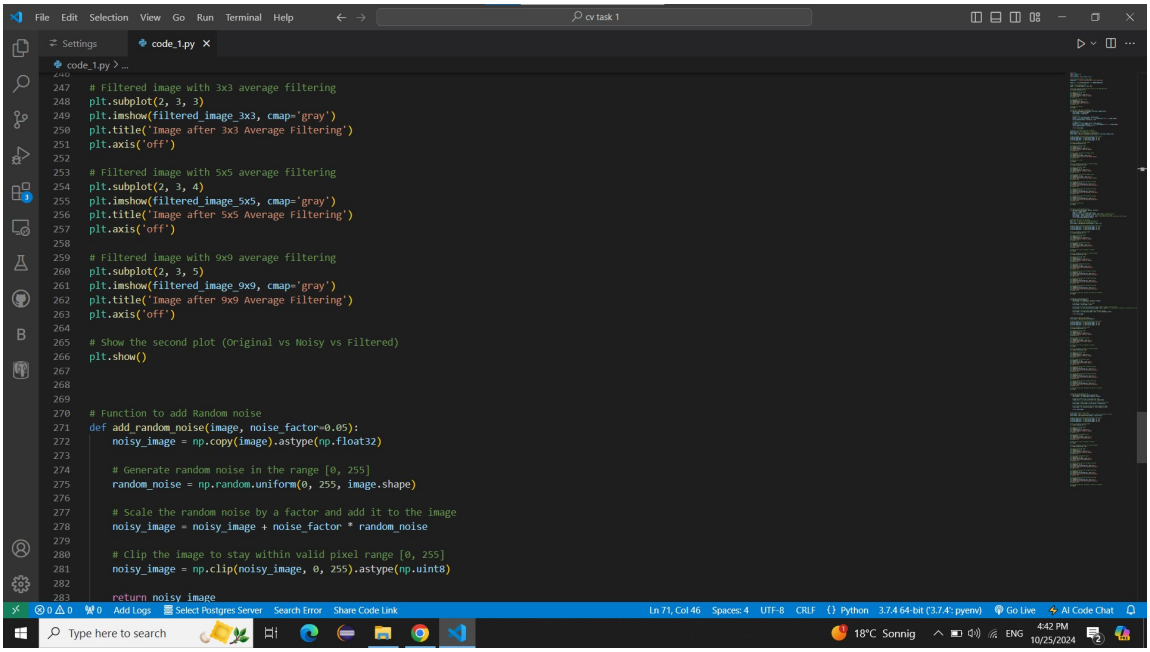
```python
# Original grayscale image
plt.subplot(1, 2, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Gaussian noise
plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Gaussian Noise')
plt.axis('off')

# Show the first plot (Original vs Noisy)
plt.show()

# Plot 2: Original vs Noisy vs Filtered images
plt.figure(figsize=(20, 10))

# Original grayscale image
plt.subplot(2, 3, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Gaussian noise
plt.subplot(2, 3, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Gaussian Noise')
plt.axis('off')

# Filtered image with 3x3 average filtering
plt.subplot(2, 3, 3)
plt.imshow(filtered_image_3x3, cmap='gray')
plt.title('Image after 3x3 Average Filtering')
plt.axis('off')

# Filtered image with 5x5 average filtering
```



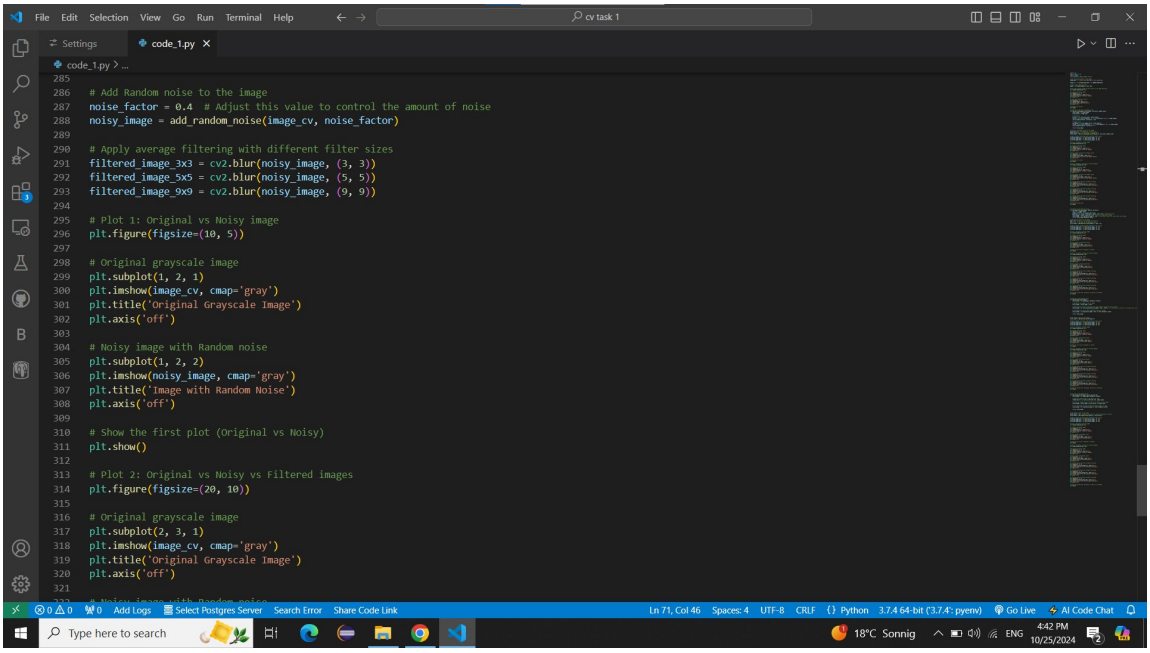
```python
# Filtered image with 5x5 average filtering
plt.subplot(2, 3, 4)
plt.imshow(filtered_image_5x5, cmap='gray')
plt.title('Image after 5x5 Average Filtering')
plt.axis('off')

# Filtered image with 9x9 average filtering
plt.subplot(2, 3, 5)
plt.imshow(filtered_image_9x9, cmap='gray')
plt.title('Image after 9x9 Average Filtering')
plt.axis('off')

# Show the second plot (Original vs Noisy vs Filtered)
plt.show()



# Function to add Poisson noise
def add_poisson_noise(image):
    noisy_image = np.copy(image).astype(np.float32)

    # Normalize the image to [0, 1] range
    noisy_image = noisy_image / 255.0

    # Apply Poisson noise and multiply the noise effect to make it more visible
    noisy_image = np.random.poisson(noisy_image * 255) / 255.0 * 2  # Multiply by a factor to increase noise visibility

    # Scale back to the [0, 255] range and convert to uint8
    noisy_image = np.clip(noisy_image * 255, 0, 255).astype(np.uint8)

    return noisy_image


# Add Poisson noise to the image
noisy_image = add_poisson_noise(image_cv)

# Apply average filtering with different filter sizes
```



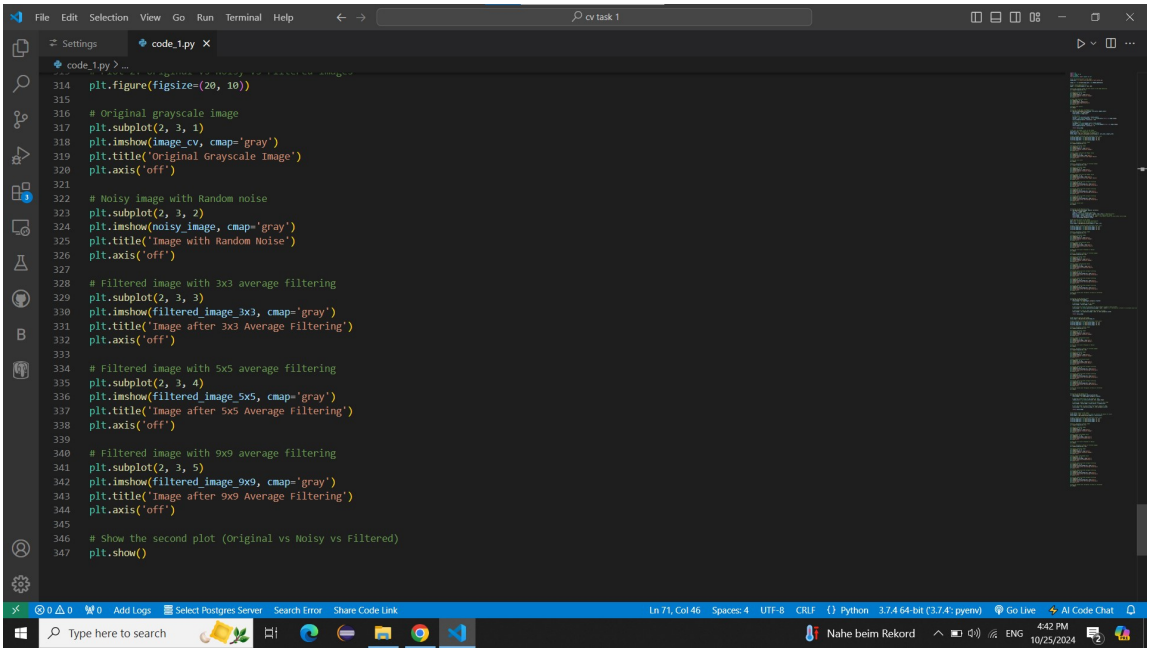
```python
# Apply average filtering with different filter sizes
filtered_image_3x3 = cv2.blur(noisy_image, (3, 3))
filtered_image_5x5 = cv2.blur(noisy_image, (5, 5))
filtered_image_9x9 = cv2.blur(noisy_image, (9, 9))

# Plot 1: Original vs Noisy image
plt.figure(figsize=(10, 5))

# Original grayscale image
plt.subplot(1, 2, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Poisson noise
plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Poisson Noise')
plt.axis('off')

# Show the first plot (Original vs Noisy)
plt.show()

# Plot 2: Original vs Noisy vs Filtered images
plt.figure(figsize=(20, 10))

# Original grayscale image
plt.subplot(2, 3, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Poisson noise
plt.subplot(2, 3, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Poisson Noise')
plt.axis('off')
```



```python
# Filtered image with 3x3 average filtering
plt.subplot(2, 3, 3)
plt.imshow(filtered_image_3x3, cmap='gray')
plt.title('Image after 3x3 Average Filtering')
plt.axis('off')

# Filtered image with 5x5 average filtering
plt.subplot(2, 3, 4)
plt.imshow(filtered_image_5x5, cmap='gray')
plt.title('Image after 5x5 Average Filtering')
plt.axis('off')

# Filtered image with 9x9 average filtering
plt.subplot(2, 3, 5)
plt.imshow(filtered_image_9x9, cmap='gray')
plt.title('Image after 9x9 Average Filtering')
plt.axis('off')

# Show the second plot (Original vs Noisy vs Filtered)
plt.show()



# Function to add Random noise
def add_random_noise(image, noise_factor=0.05):
    noisy_image = np.copy(image).astype(np.float32)

    # Generate random noise in the range [0, 255]
    random_noise = np.random.uniform(0, 255, image.shape)

    # Scale the random noise by a factor and add it to the image
    noisy_image = noisy_image + noise_factor * random_noise

    # Clip the image to stay within valid pixel range [0, 255]
    noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)

    return noisy_image
```



```
# Add Random noise to the image
noise_factor = 0.4  # Adjust this value to control the amount of noise
noisy_image = add_random_noise(image_cv, noise_factor)

# Apply average filtering with different filter sizes
filtered_image_3x3 = cv2.blur(noisy_image, (3, 3))
filtered_image_5x5 = cv2.blur(noisy_image, (5, 5))
filtered_image_9x9 = cv2.blur(noisy_image, (9, 9))

# Plot 1: Original vs Noisy image
plt.figure(figsize=(10, 5))

# Original grayscale image
plt.subplot(1, 2, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Random noise
plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Random Noise')
plt.axis('off')

# Show the first plot (Original vs Noisy)
plt.show()

# Plot 2: Original vs Noisy vs Filtered images
plt.figure(figsize=(20, 10))

# Original grayscale image
plt.subplot(2, 3, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')
```



```
plt.figure(figsize=(20, 10))

# Original grayscale image
plt.subplot(2, 3, 1)
plt.imshow(image_cv, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')

# Noisy image with Random noise
plt.subplot(2, 3, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Image with Random Noise')
plt.axis('off')

# Filtered image with 3x3 average filtering
plt.subplot(2, 3, 3)
plt.imshow(filtered_image_3x3, cmap='gray')
plt.title('Image after 3x3 Average Filtering')
plt.axis('off')

# Filtered image with 5x5 average filtering
plt.subplot(2, 3, 4)
plt.imshow(filtered_image_5x5, cmap='gray')
plt.title('Image after 5x5 Average Filtering')
plt.axis('off')

# Filtered image with 9x9 average filtering
plt.subplot(2, 3, 5)
plt.imshow(filtered_image_9x9, cmap='gray')
plt.title('Image after 9x9 Average Filtering')
plt.axis('off')

# Show the second plot (Original vs Noisy vs Filtered)
plt.show()
```