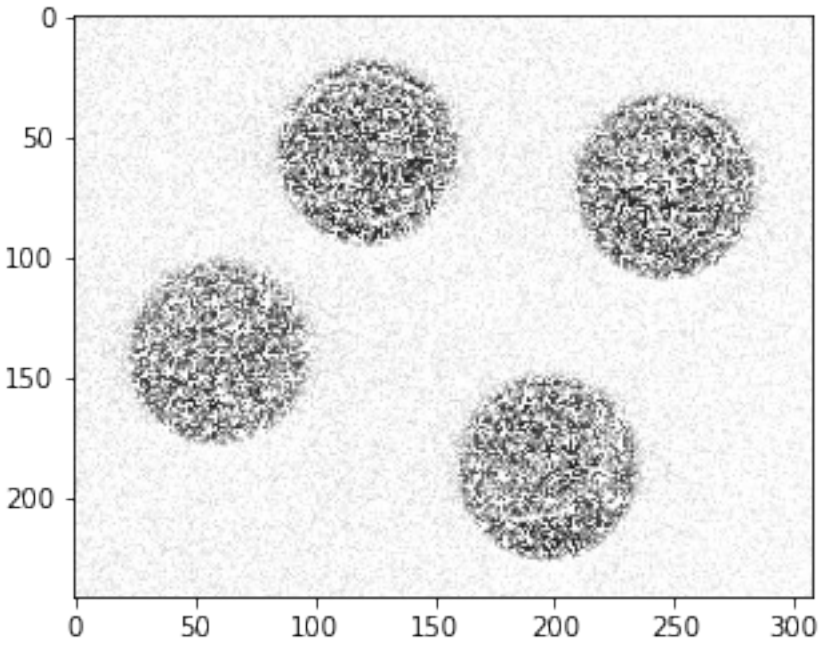
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

EXP-1

```
# Load the image
image_gray = cv2.imread('eight.jpg', cv2.IMREAD_GRAYSCALE)

# Display the grayscale image
plt.imshow(image_gray, cmap='gray')
plt.axis('off')  # Turn off axis labels
plt.title('Grayscale Image')
plt.show()
```

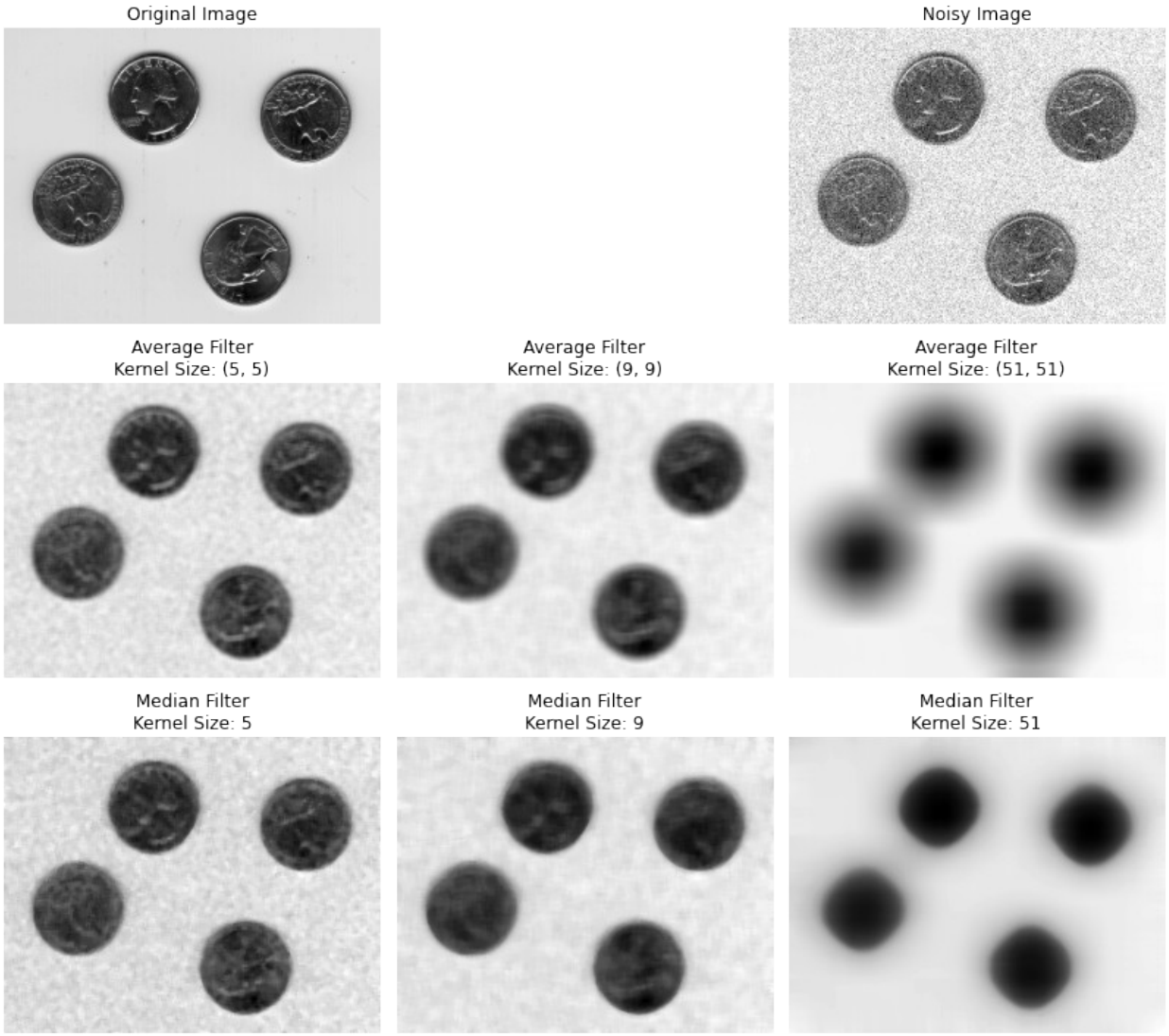

```
def add_gaussian_noise(image, mean=0, std_dev=25):

    #Gaussian noise
    noise = np.random.normal(mean, std_dev,
image.shape).astype(np.uint8)

    #noise
    noisy_image = cv2.add(image, noise)

    noisy_image = np.clip(noisy_image, 0, 255)
```

```
    return noisy_image

# Add Gaussian noise to the grayscale image
noisy_image = add_gaussian_noise(image_gray, mean=0, std_dev=25)

# Display noisy image
plt.imshow(noisy_image, cmap='gray')
plt.show()
```




```
def apply_average_filter(image, kernel_size):

    smoothed_image = cv2.blur(image, kernel_size)
    return smoothed_image

def apply_median_filter(image, kernel_size):

    smoothed_image = cv2.medianBlur(image, kernel_size)
    return smoothed_image

average_kernel_sizes = [(5,5 ), (9, 9), (51,51 )]
median_kernel_sizes = [5, 9 , 51]

smoothed_images_average = [apply_average_filter(noisy_image,
kernel_size)
                           for kernel_size in average_kernel_sizes]

smoothed_images_median = [apply_median_filter(noisy_image,
kernel_size)
```

```
                          for kernel_size in median_kernel_sizes]


plt.figure(figsize=(15, 10))

plt.subplot(3, 4, 1)
plt.imshow(image_gray, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(3, 4, 3)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')

for i, kernel_size in enumerate(average_kernel_sizes):
    plt.subplot(3, 4, i+5)
    plt.imshow(smoothed_images_average[i], cmap='gray')
    plt.title(f'Average Filter\nKernel Size: {kernel_size}')
    plt.axis('off')

for i, kernel_size in enumerate(median_kernel_sizes):
    plt.subplot(3, 4, i+9)
    plt.imshow(smoothed_images_median[i], cmap='gray')
    plt.title(f'Median Filter\nKernel Size: {kernel_size}')
    plt.axis('off')

plt.tight_layout()
plt.show()
```



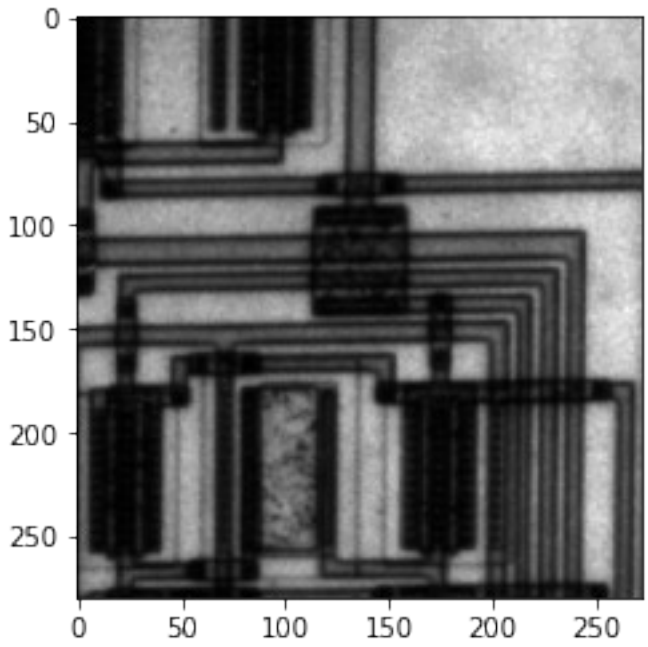

```
"""
image = cv2.imread('eight.jpg', cv2.IMREAD_GRAYSCALE)

# Function to add Gaussian noise to the image
def add_gaussian_noise(image, mean=0, sigma=25):
    row, col = image.shape
    gauss = np.random.normal(mean, sigma, (row, col))
    noisy = np.clip(image + gauss, 0, 255)
    return noisy.astype(np.uint8)
# Apply Gaussian noise
noisy_image = add_gaussian_noise(image)
# Function to apply smoothing filters
def apply_smoothing(image, filter_type, window_size):
    if filter_type == 'average':
        smoothed = cv2.blur(image, (window_size, window_size))
    elif filter_type == 'median':
```

```
        smoothed = cv2.medianBlur(image, window_size)

    else: raise ValueError("Unknown filter type")
    return smoothed




# Apply different smoothing filters
window_sizes = [3, 5, 9]  # Different window sizes for smoothing
filters
filtered_images = {}
for filter_type in ['average', 'median']:
    for window_size in window_sizes:
        key = f"{filter_type}_filter_{window_size}"
        filtered_images[key] = apply_smoothing(noisy_image,
filter_type, window_size)
# Plotting
plt.figure(figsize=(12, 8))
plt.subplot(3, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(3, 3, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')
# Plot filtered images
index =3
for key,value in filtered_images.items():
    plt.subplot(3, 3, index)
    plt.imshow(value, cmap='gray')
    plt.title(key)
    plt.axis('off')
    index += 1

plt.tight_layout()
#plt.show()
"""

'\nimage = cv2.imread(\'eight.jpg\', cv2.IMREAD_GRAYSCALE)\n\n#
Function to add Gaussian noise to the image\ndef
add_gaussian_noise(image, mean=0, sigma=25): \n    row, col =
image.shape\n    gauss = np.random.normal(mean, sigma, (row, col)) \n
noisy = np.clip(image + gauss, 0, 255)\n    return
noisy.astype(np.uint8)\n# Apply Gaussian noise\nnoisy_image =
```

```
add_gaussian_noise(image)\n# Function to apply smoothing filters\ndef apply_smoothing(image, filter_type, window_size):\n    if filter_type == \'average\':\n        smoothed = cv2.blur(image, (window_size, window_size))\n    elif filter_type == \'median\':\n        smoothed = cv2.medianBlur(image, window_size)\n            \n    else: raise ValueError("Unknown filter type")\n    return smoothed\n\n            \n\n\n\n\n\n# Apply different smoothing filters\nwindow_sizes = [3, 5, 9]  # Different window sizes for smoothing filters\nfiltered_images = {}\nfor filter_type in [\'average\', \'median\']:\n    for window_size in window_sizes:\n key = f"{filter_type}_filter_{window_size}"\n filtered_images[key] = apply_smoothing(noisy_image, filter_type, window_size)\n# Plotting\nplt.figure(figsize=(12, 8))\nplt.subplot(3, 3, 1) \nplt.imshow(image, cmap=\'gray\') \nplt.title(\'Original Image\')\nplt.axis(\'off\')\n\nplt.subplot(3, 3, 2) \nplt.imshow(noisy_image, cmap=\'gray\') \nplt.title(\'Noisy Image\') \nplt.axis(\'off\')\n# Plot filtered images\nindex =3\nfor key,value in filtered_images.items(): \n    plt.subplot(3, 3, index)\n plt.imshow(value, cmap=\'gray\') \n    plt.title(key)\n plt.axis(\'off\')\n    index += 1\n \nplt.tight_layout()\n#plt.show()\n'
```

Experiment-2

```
circuit_image = cv2.imread('circuit.jpg', cv2.IMREAD_GRAYSCALE)

plt.imshow(circuit_image, cmap='gray')
plt.show()
```

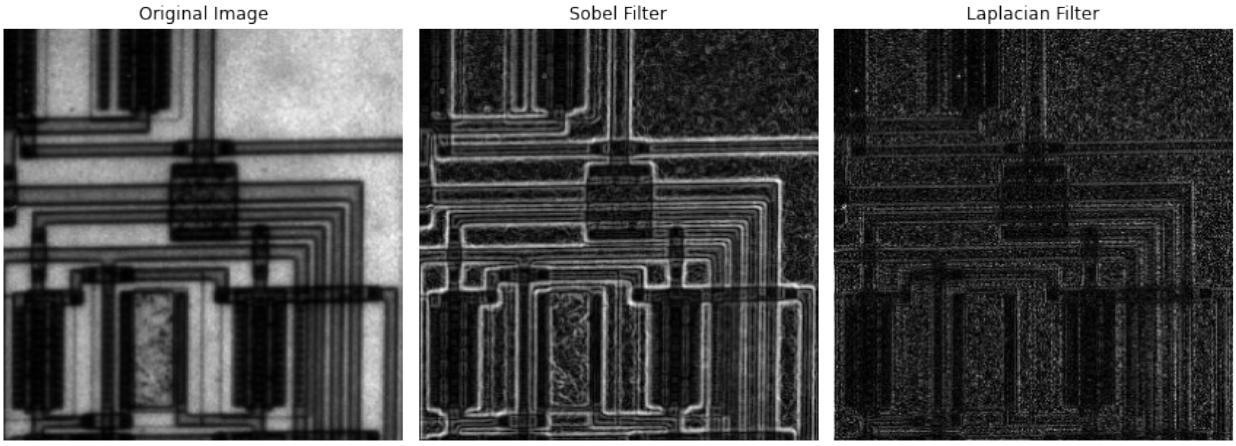



```
def apply_sobel_filter(image):

    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    sobel = cv2.addWeighted(cv2.convertScaleAbs(sobel_x), 0.5,
cv2.convertScaleAbs(sobel_y), 0.5, 0)
    return sobel

def apply_laplacian_filter(image):
    laplacian = cv2.Laplacian(image, cv2.CV_64F)
    laplacian = np.uint8(np.absolute(laplacian))
    return laplacian


"""detecting edges: Sobel and Laplacian.
Sobel uses horizontal and vertical kernels,
while Laplacian uses one symmetrical kernel"""
```

```
'detecting edges: Sobel and Laplacian. \nSobel uses horizontal and
vertical kernels, \nwhile Laplacian uses one symmetrical kernel'
```

```
sobel_image = apply_sobel_filter(circuit_image)

laplacian_image = apply_laplacian_filter(circuit_image)

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.imshow(circuit_image, cmap='gray')
plt.title('Original Image')
```

```
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(sobel_image, cmap='gray')
plt.title('Sobel Filter')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(laplacian_image, cmap='gray')
plt.title('Laplacian Filter')
plt.axis('off')

plt.tight_layout()
plt.show()
```

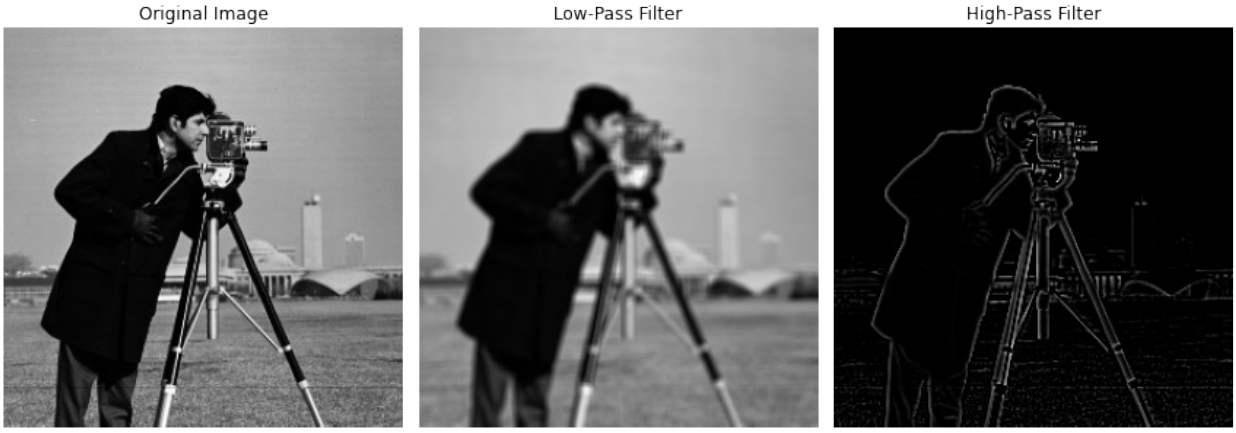


Experiment-3:

```
cameraman_image = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)

plt.imshow(cameraman_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.show()
```


```
def create_low_pass_filter(size):

    kernel = np.ones((size, size), dtype=np.float32) / (size * size)
    return kernel

def create_high_pass_filter(size):

    kernel = np.ones((size, size), dtype=np.float32) / (size * size)
    kernel[size // 2, size // 2] = -1
    return kernel

def apply_filter(image, kernel):

    filtered_image = cv2.filter2D(image, -1, kernel)
    return filtered_image

# Define filter kernel sizes
kernel_size = 5

#low-pass filter
low_pass_kernel = create_low_pass_filter(kernel_size)

#high
high_pass_kernel = create_high_pass_filter(kernel_size)


low_pass_image = apply_filter(cameraman_image, low_pass_kernel)

high_pass_image = apply_filter(cameraman_image, high_pass_kernel)

plt.figure(figsize=(12, 6))
```

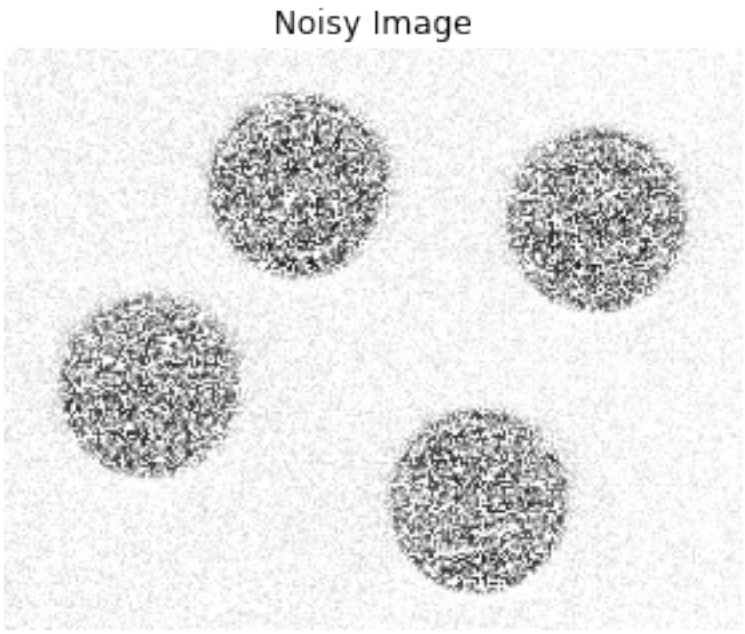
```
plt.subplot(1, 3, 1)
plt.imshow(cameraman_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(low_pass_image, cmap='gray')
plt.title('Low-Pass Filter')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(high_pass_image, cmap='gray')
plt.title('High-Pass Filter')
plt.axis('off')

plt.tight_layout()
plt.show()
```



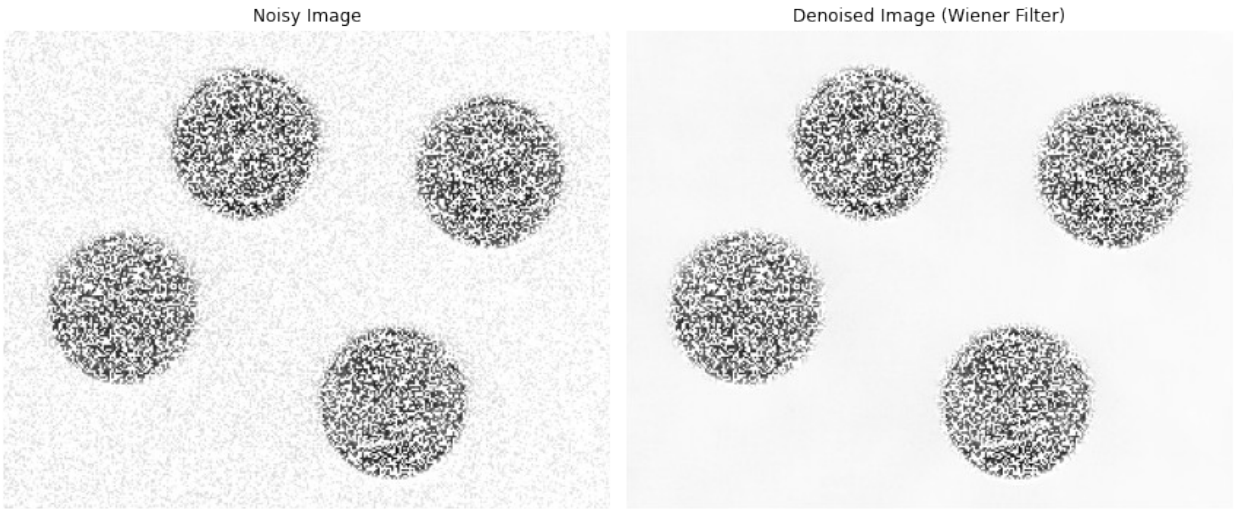
Experiment-4:

```
eight_image = cv2.imread('eight.jpg', cv2.IMREAD_GRAYSCALE)

plt.imshow(eight_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.show()
```


```
def add_gaussian_noise(image, mean=0, std_dev=25):

    noise = np.random.normal(mean, std_dev,
image.shape).astype(np.uint8)

    noisy_image = cv2.add(image, noise)

    noisy_image = np.clip(noisy_image, 0, 255)

    return noisy_image

noisy_eight_image = add_gaussian_noise(eight_image)

plt.imshow(noisy_eight_image, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')
plt.show()
```




```
def apply_wiener_filter(image):

    # Apply Wiener filter (non-local means denoising)
    denoised_image = cv2.fastNlMeansDenoising(image, None, h=10,
templateWindowSize=7, searchWindowSize=21)
    return denoised_image

denoised_eight_image = apply_wiener_filter(noisy_eight_image)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(noisy_eight_image, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(denoised_eight_image, cmap='gray')
plt.title('Denoised Image (Wiener Filter)')
plt.axis('off')

plt.tight_layout()
plt.show()
```



Experiment-5:

```
cameraman_image = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)

plt.imshow(cameraman_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.show()
```


```
def add_salt_and_pepper_noise(image, salt_prob=0.01,
pepper_prob=0.01):
```

```
    noisy_image = np.copy(image)

    # salt
    salt_mask = np.random.rand(*image.shape) < salt_prob
    noisy_image[salt_mask] = 255

    #pepper
    pepper_mask = np.random.rand(*image.shape) < pepper_prob
    noisy_image[pepper_mask] = 0

    return noisy_image

# Apply salt and pepper
noisy_cameraman_image = add_salt_and_pepper_noise(cameraman_image)

plt.imshow(noisy_cameraman_image, cmap='gray')
plt.title('Noisy Image with Salt and Pepper Noise')
plt.axis('off')
plt.show()
```


```
def apply_arithmetic_mean_filter(image, kernel_size=3):

    return cv2.blur(image, (kernel_size, kernel_size))

def apply_median_filter(image, kernel_size=3):

    return cv2.medianBlur(image, kernel_size)

def apply_max_filter(image, kernel_size=3):
```

```
    return cv2.dilate(image, np.ones((kernel_size, kernel_size),
np.uint8))

def apply_min_filter(image, kernel_size=3):

    return cv2.erode(image, np.ones((kernel_size, kernel_size),
np.uint8))

# Define kernel size for filters
kernel_size = 3

# Apply arithmetic mean filter
arithmetic_mean_filtered =
apply_arithmetic_mean_filter(noisy_cameraman_image, kernel_size)

# Apply median filter
median_filtered = apply_median_filter(noisy_cameraman_image,
kernel_size)

# Apply maximum filter
max_filtered = apply_max_filter(noisy_cameraman_image, kernel_size)

# Apply minimum filter
min_filtered = apply_min_filter(noisy_cameraman_image, kernel_size)

# Display the original noisy image and images after applying filters
plt.figure(figsize=(15, 10))

plt.subplot(2, 3, 1)
plt.imshow(noisy_cameraman_image, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(arithmetic_mean_filtered, cmap='gray')
plt.title('Arithmetic Mean Filter')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(median_filtered, cmap='gray')
plt.title('Median Filter')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(max_filtered, cmap='gray')
plt.title('Max Filter')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(min_filtered, cmap='gray')
plt.title('Min Filter')
```

```
plt.axis('off')

plt.tight_layout()
plt.show()
```