Lab Report: 08

Title: Noise Simulation and Removal Techniques

Course title: Digital Image Processing Laboratory
Course code: CSE-406
4th Year 1st Semester Examination 2023

Date of Submission: 27/10/2024

Submitted to-

Dr. Md. Golam Moazzam
Professor
Department of Computer Science and Engineering
Jahangirnagar University
&
Dr. Morium Akter
Professor
Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Class Roll	Exam Roll	Name
353	202165	Shanjida Alam

Experiment No: 01

Experiment Name: How will a image look after adding Gaussian, Rayleigh and Erlang noise

Objectives:

1. The primary objective of this code is to add different types of noise (Gaussian, Rayleigh, and Erlang) to an image and visualize their effects.
2. It can also be used to understand how various noise types affect image quality, a key concept in image processing for preparing data, filtering, and studying the robustness of image analysis algorithms.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('nature.jpeg', cv2.IMREAD_GRAYSCALE)
def add_gaussian_noise(image, mean=0, std=25):
   gauss = np.random.normal(mean, std, image.shape).astype('float32')
   noisy_image = image + gauss
   noisy_image = np.clip(noisy_image, 0, 255).astype('uint8')
   return noisy_image
def add_rayleigh_noise(image, scale=25):
   rayleigh = np.random.rayleigh(scale, image.shape).astype('float32')
   noisy_image = image + rayleigh
   noisy_image = np.clip(noisy_image, 0, 255).astype('uint8')
   return noisy_image
def add_erlang_noise(image, shape=2, scale=15):
   erlang = np.random.gamma(shape, scale, image.shape).astype('float32')
   noisy_image = image + erlang
   noisy_image = np.clip(noisy_image, 0, 255).astype('uint8')
   return noisy_image
gaussian_noisy_image = add_gaussian_noise(image)
rayleigh_noisy_image = add_rayleigh_noise(image)
erlang_noisy_image = add_erlang_noise(image)
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.subplot(2, 2, 2)
plt.title("Gaussian Noise")
plt.imshow(gaussian_noisy_image, cmap='gray')
plt.axis('off')
plt.subplot(2, 2, 3)
plt.title("Rayleigh Noise")
plt.imshow(rayleigh_noisy_image, cmap='gray')
plt.axis('off')
plt.subplot(2, 2, 4)
plt.title("Erlang Noise")
plt.imshow(erlang_noisy_image, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()
```

Output:

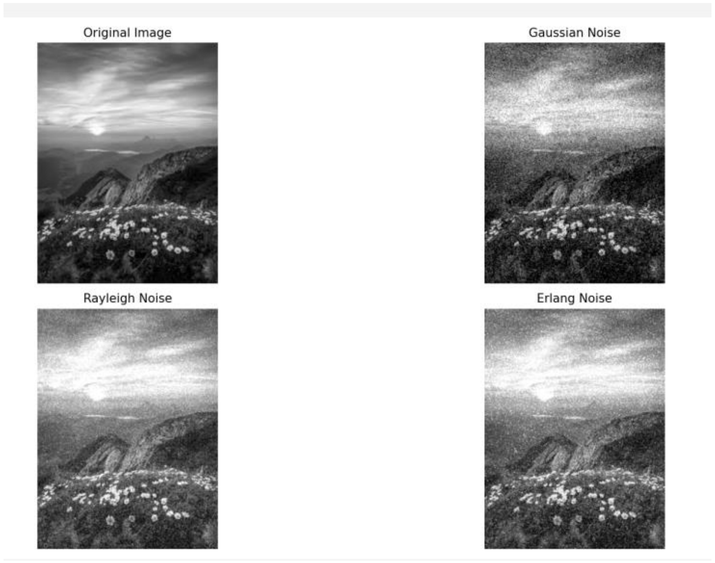

Figure 1.1: Adding Gaussian, Rayleigh and Erlang noise in Python

Explanation:

1. Import libraries.
2. Adds gaussian noise with a normal distribution (mean=0 and std=25).
3. The noise is generated using `np.random.normal` and then added to the original image.
4. Generated using the Rayleigh distribution (scale=25). This noise is added to the image similarly to the Gaussian noise, with clipping to maintain valid pixel values.
5. Adds noise using `np.random.gamma`(shape, scale).
6. Parameter's shape=2 and scale=15 define the characteristics of the Erlang distribution.

Code-02: MATLAB

```
image = imread('nature.jpeg');
image = rgb2gray(image);
gaussian_noisy_image = imnoise(image, 'gaussian', 0, 0.01);
rayleigh_noise = raylrnd(25, size(image));
rayleigh_noisy_image = double(image) + rayleigh_noise;
rayleigh_noisy_image = uint8(min(rayleigh_noisy_image, 255));
shape = 2;
scale = 15;
erlang_noise = gamrnd(shape, scale, size(image));
erlang_noisy_image = double(image) + erlang_noise;
erlang_noisy_image = uint8(min(erlang_noisy_image, 255));
figure;
subplot(2, 2, 1);
imshow(image);
title('Original Image');
subplot(2, 2, 2);
imshow(gaussian_noisy_image);
title('Gaussian Noise');
subplot(2, 2, 3);
imshow(rayleigh_noisy_image);
title('Rayleigh Noise');
subplot(2, 2, 4);
imshow(erlang_noisy_image);
title('Erlang Noise');
```

Output:

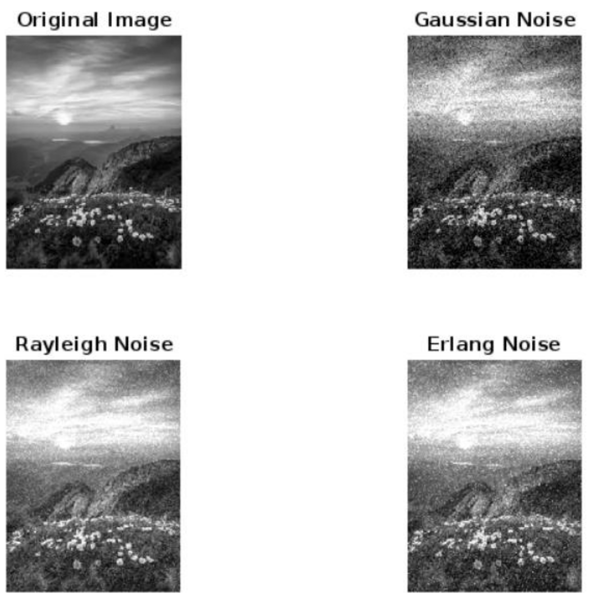

Figure 1.2: Adding Gaussian, Rayleigh and Erlang noise in Python

Explanation:

1. The original image (nature.jpeg) is loaded using `imread`. Since the analysis focuses on a grayscale version, the image is converted from RGB to grayscale using `rgb2gray`.
2. `imnoise` function is used to add Gaussian noise to the grayscale image. Mean (0) and variance (0.01) are specified to control the noise characteristics.
3. Rayleigh noise is generated using `raylrnd`, with a mode parameter of 25. The `size(image)` argument ensures the noise array has the same dimensions as the image.
4. Erlang noise (a type of Gamma noise) is added with `parameters shape = 2` and `scale = 15` using `gamrnd`, which simulates the noise distribution.

Experiment No: 02

Experiment Name: Noise Simulation (Gaussian Noise Adding and Removing)

Objectives:

1. To study and analyze the effects of different noise types—Gaussian, Rayleigh, and Erlang—on a grayscale image.
2. To understand the methods for generating and adding noise distributions to images.
3. To visualize and compare the impact of these noise distributions on image quality and appearance.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def add_gaussian_noise(image, mean=0, var=0.01):
    row, col = image.shape
    sigma = var ** 0.5
    gaussian = np.random.normal(mean, sigma, (row, col))
    noisy_image = image + gaussian * 255
    return np.clip(noisy_image, 0, 255).astype(np.uint8)
def arithmetic_mean_filter(image, kernel_size=3):
    return cv2.blur(image, (kernel_size, kernel_size)),
def geometric_mean_filter(image, kernel_size=3):
    image_float = image.astype(np.float32)
    img_log = np.log1p(image_float)
    kernel = np.ones((kernel_size, kernel_size), np.float32) / (kernel_size * kernel_size)
    log_filtered = cv2.filter2D(img_log, -1, kernel)
    geometric_mean_img = np.expm1(log_filtered)
    return np.clip(geometric_mean_img, 0, 255).astype(np.uint8)
image = cv2.imread('nature.jpeg', 0)
noisy_image = add_gaussian_noise(image)
arithmetic_filtered_image = arithmetic_mean_filter(noisy_image)
geometric_filtered_image = geometric_mean_filter(noisy_image)
plt.figure(figsize=(10, 5))
plt.subplot(1, 4, 1), plt.imshow(image, cmap='gray'), plt.title('Original Image')
plt.subplot(1, 4, 2), plt.imshow(noisy_image, cmap='gray'), plt.title('Noisy (Gaussian)')
plt.subplot(1, 4, 3), plt.imshow(arithmetic_filtered_image, cmap='gray'), plt.title('Arithmetic
Mean Filter')
plt.subplot(1, 4, 4), plt.imshow(geometric_filtered_image, cmap='gray'), plt.title('Geometric Mean
Filter')
plt.show()
```

Output:

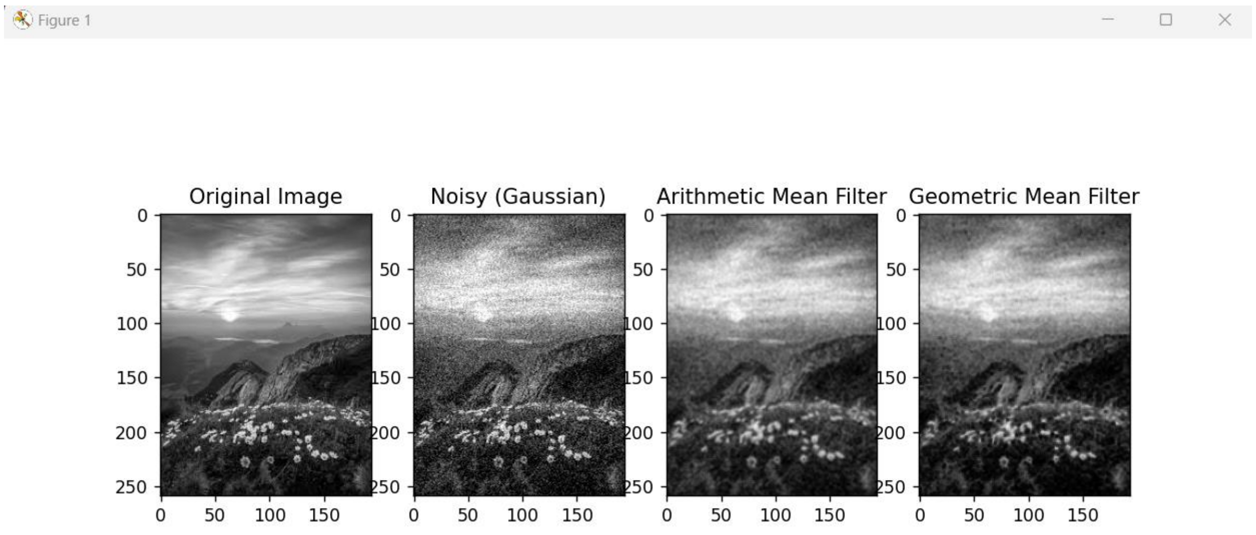

Figure 2.1: Noise Simulation (Gaussian Noise Adding and Removing) in Python code

Explanation:

1. Image Loading and Grayscale Conversion.
2. Gaussian noise is added using the `imnoise` function. Parameters are set to a mean of 0 and variance of `0.01`, adding subtle random intensity variations that simulate electronic noise.

3. Rayleigh noise is generated with a mode parameter of 25, creating an image-sized noise matrix. Adding this noise to the image requires converting it to double to prevent overflow. The pixel values are clipped and cast back to uint8 to retain the image format.
4. Erlang noise is generated using the gamma distribution with parameters `shape = 2` and `scale = 15`. The addition and clipping process mirrors that of Rayleigh noise.

Code-02: MATLAB

```
img = imread('nature.jpeg');
img = rgb2gray(img);
noisy_img = imnoise(img, 'gaussian', 0, 0.01);
arithmetic_mean_img = imfilter(noisy_img, fspecial('average', [3 3]));
[rows, cols] = size(noisy_img);
geo_img = double(noisy_img);
for i = 2:rows-1
    for j = 2:cols-1
        patch = geo_img(i-1:i+1, j-1:j+1);
        geo_mean = exp(mean(mean(log(double(patch) + 1))));
        geo_img(i,j) = geo_mean - 1;
    end
end
geo_img = uint8(geo_img);
figure;
subplot(2, 2, 1), imshow(img), title('Original Image');
subplot(2, 2, 2), imshow(noisy_img), title('Noisy (Gaussian) Image');
subplot(2, 2, 3), imshow(arithmetic_mean_img), title('Arithmetic Mean Filtered
Image');
subplot(2, 2, 4), imshow(geo_img), title('Geometric Mean Filtered Image');
```

Output:

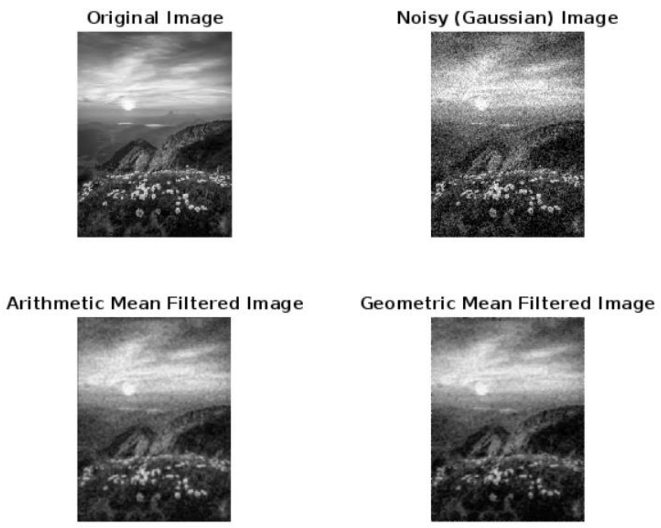

Figure 2.2: Noise Simulation (Gaussian Noise Adding and Removing) in MATLAB

Explanation:

1. Image Loading and Conversion to Grayscale.
2. Gaussian noise is added to the grayscale image using `imnoise`. The mean (0) and variance (0.01) parameters control the amount and spread of the noise. The result is a noisy version of the `image (noisy_img)`, simulating random variations in pixel intensities.
3. This line applies an arithmetic mean filter (also known as an averaging filter) to reduce noise. `fspecial('average', [3 3])` creates a 3x3 averaging filter, and imfilter applies this filter to `noisy_img`, resulting in a smoothed image (`arithmetic_mean_img`). This method reduces noise by averaging pixel values in a 3x3 neighborhood, which helps reduce sharp noise variations.
4. Here, the code initializes `geo_img` as a copy of noisy_img in double format (to accommodate non-integer calculations) and determines the size of the image.

Experiment No: 03

Experiment Name: Noise Simulation (Periodic Noise Adding and Removing)

Objectives:

1. The objective of this experiment is to understand the effects of Periodic noise on images and to explore different filtering techniques, namely the arithmetic mean filter and the geometric mean filter, for noise reduction.
2. The experiment aims to evaluate the effectiveness of each filtering method and compare the results visually.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('nature.jpeg', 0)
dft = np.fft.fft2(image)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20 * np.log(np.abs(dft_shift))
def notch_filter(dft_shift, center, radius=10):
    rows, cols = dft_shift.shape
    crow, ccol = int(rows / 2), int(cols / 2)
    mask = np.ones((rows, cols), np.uint8)
    cv2.circle(mask, center, radius, 0, thickness=-1)
    fshift = dft_shift * mask
    return fshift
filtered_dft = notch_filter(dft_shift, (130, 130), radius=15)
filtered_dft = notch_filter(filtered_dft, (170, 170), radius=15)
f_ishift = np.fft.ifftshift(filtered_dft)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)
plt.figure(figsize=(12, 6))
plt.subplot(1, 3, 1), plt.imshow(image, cmap='gray'), plt.title('Original Image')
plt.subplot(1, 3, 2), plt.imshow(magnitude_spectrum, cmap='gray'), plt.title('Magnitude
Spectrum')
plt.subplot(1, 3, 3), plt.imshow(img_back, cmap='gray'), plt.title('Filtered Image')
plt.show()
```

Output:

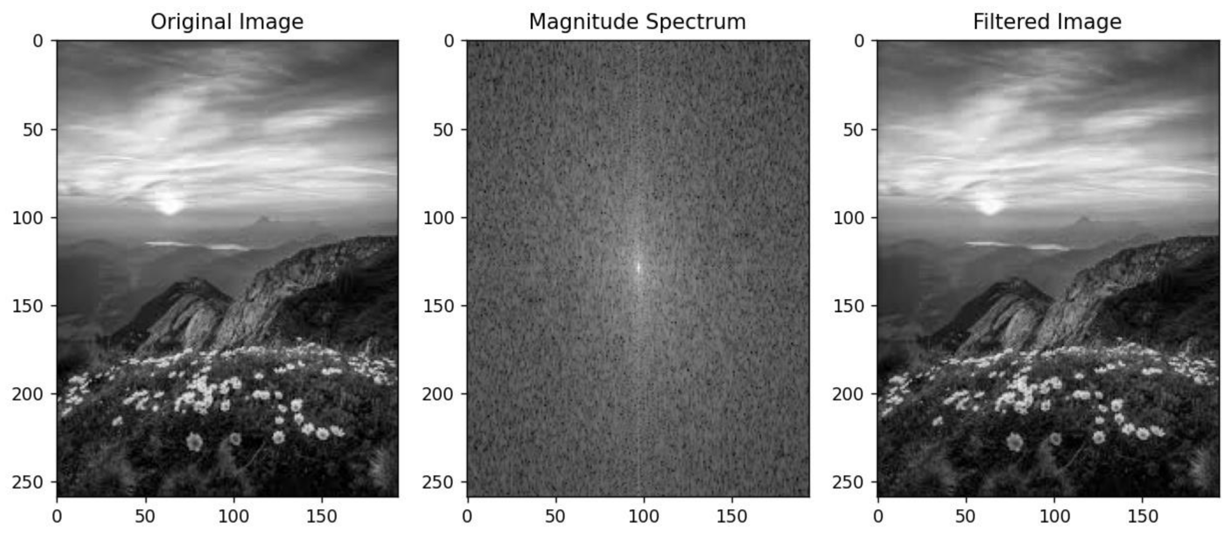

Figure 3.1: Noise Simulation (Periodic Noise Adding and Removing) using Python

Explanation:

1. Image Loading and Conversion to Grayscale.
2. Gaussian noise is added to the grayscale image with a mean of 0 and a variance of 0.01. This simulates random variations in pixel intensity, creating a noisy version of the image.

3. An arithmetic mean filter is applied using a 3x3 averaging filter kernel. This filter reduces noise by averaging the values in a 3x3 neighborhood around each pixel, producing a smoother, less noisy image (`arithmetic_mean_img`).
4. Initializes `geo_img` as a copy of `noisy_img` with a double type to accommodate precise calculations.
5. Converts `geo_img` back to `uint8` format to be compatible with image display functions.

Code-02: MATLAB

```
img = imread('nature.jpeg');
img = rgb2gray(img);
F = fft2(double(img));
F_shifted = fftshift(F);
magnitude_spectrum = log(1 + abs(F_shifted));
figure;
subplot(2, 2, 1), imshow(img, []), title('Original Image');
subplot(2, 2, 2), imshow(magnitude_spectrum, []), title('Magnitude Spectrum');
function F_filtered = notch_filter(F_shifted, center, radius)
    [rows, cols] = size(F_shifted);
    mask = ones(rows, cols);
    [X, Y] = meshgrid(1:cols, 1:rows);

    mask(((X - center(2)).^2 + (Y - center(1)).^2) < radius^2) = 0;

    F_filtered = F_shifted .* mask;
end
F_filtered = notch_filter(F_shifted, [130, 130], 15);
F_filtered = notch_filter(F_filtered, [170, 170], 15);
F_ishifted = ifftshift(F_filtered);
img_filtered = ifft2(F_ishifted);
img_filtered = abs(img_filtered);
subplot(2, 2, 3), imshow(img_filtered, []), title('Filtered Image');
```

Output:

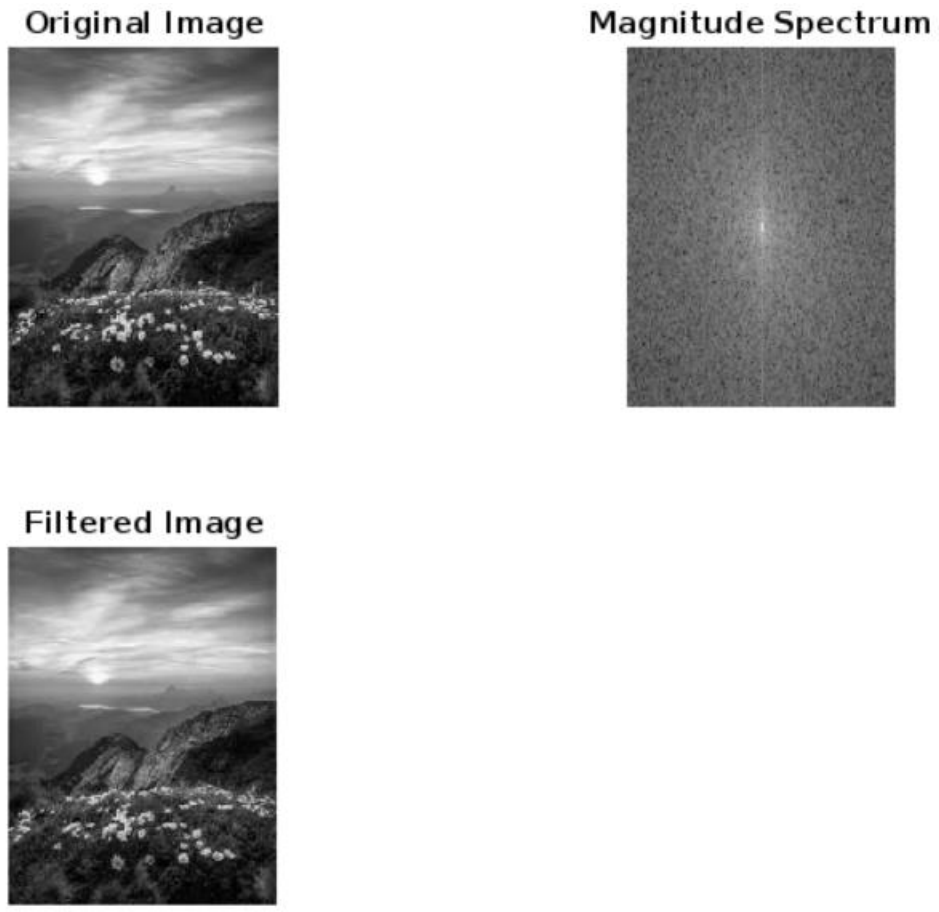

Figure 3.2 Noise Simulation (Periodic Noise Adding and Removing) using MATLAB

Explanation:

1. Image Loading and Grayscale Conversion.
2. The fft2 function is used to compute the 2D Fourier Transform of the image (`img`), converting it from the spatial domain to the frequency domain. This results in F, a complex-valued matrix representing the image in the frequency domain.
3. The `fftshift` function is applied to `F` to shift the zero-frequency component to the center of the spectrum. This makes it easier to visualize and manipulate specific frequencies in the frequency domain.
4. The magnitude of the frequency components is calculated by taking the absolute value of `F_shifted`. Applying `log(1 + abs(F_shifted))` enhances the visibility of the

spectrum by reducing the dynamic range, which helps in visualizing lower-intensity frequencies.

Experiment No: 04

Experiment Name: Noise Simulation (Salt and Pepper Noise Adding and Removing) Objectives:

1. Add artificial salt-and-pepper noise to a grayscale image.
2. Use a median filter to remove the salt-and-pepper noise.
3. Visualize the original, noisy, and denoised images side-by-side for comparison.

Code-01: Python

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
def add_salt_and_pepper_noise(image, salt_prob=0.05, pepper_prob=0.05):
    noisy_image = np.copy(image)
    num_salt = np.ceil(salt_prob * image.size)
    coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image.shape]
    noisy_image[coords[0], coords[1]] = 255
    num_pepper = np.ceil(pepper_prob * image.size)
    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
def remove_salt_and_pepper_noise(image, kernel_size=3):
    denoised_image = cv2.medianBlur(image, kernel_size)
    return denoised_image
image = cv2.imread('nature.jpeg', 0)
noisy_image = add_salt_and_pepper_noise(image)
denoised_image = remove_salt_and_pepper_noise(noisy_image)
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1), plt.imshow(image, cmap='gray'), plt.title('Original')
plt.subplot(1, 3, 2), plt.imshow(noisy_image, cmap='gray'), plt.title('Noisy (Salt & Pepper)')
plt.subplot(1, 3, 3), plt.imshow(denoised_image, cmap='gray'), plt.title('Denoised (Median Filter)')
plt.show()
```

Output:

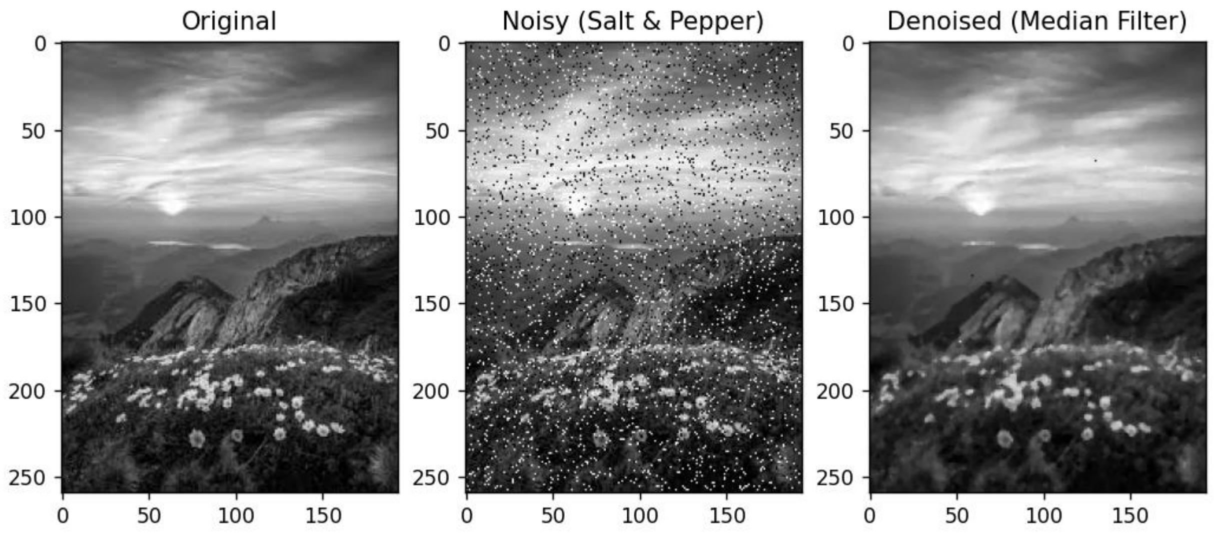

Figure 4.1: Noise Simulation (Salt and Pepper Noise Adding and Removing) using python code

Explanation:

1. The necessary libraries are imported, including cv2 for image processing, numpy for numerical operations, and matplotlib.pyplot for visualization.
2. Randomly selects pixel coordinates to `add salt (white pixels) and pepper (black pixels)`, using `np.random.randint`.
3. Takes the noisy image and a `kernel_size` parameter.
4. This technique allows for visualizing the frequency components of the image, highlighting areas with low and high-frequency content.

Code-02: MATLAB

```
img = imread('nature.jpeg');
img = rgb2gray(img);
noisy_img = imnoise(img, 'salt & pepper', 0.05);
denoised_img = medfilt2(noisy_img, [3 3]);
figure;
subplot(2, 2, 1), imshow(img), title('Original Image');
subplot(2, 2, 2), imshow(noisy_img), title('Noisy (Salt & Pepper) Image');
subplot(2, 2, 4), imshow(denoised_img), title('Denoised (Median Filter) Image');
subplot(1, 2, 2);
imshow(magnitudeFFTLog, []);
title('Magnitude Spectrum (FFT)');
```

Output:

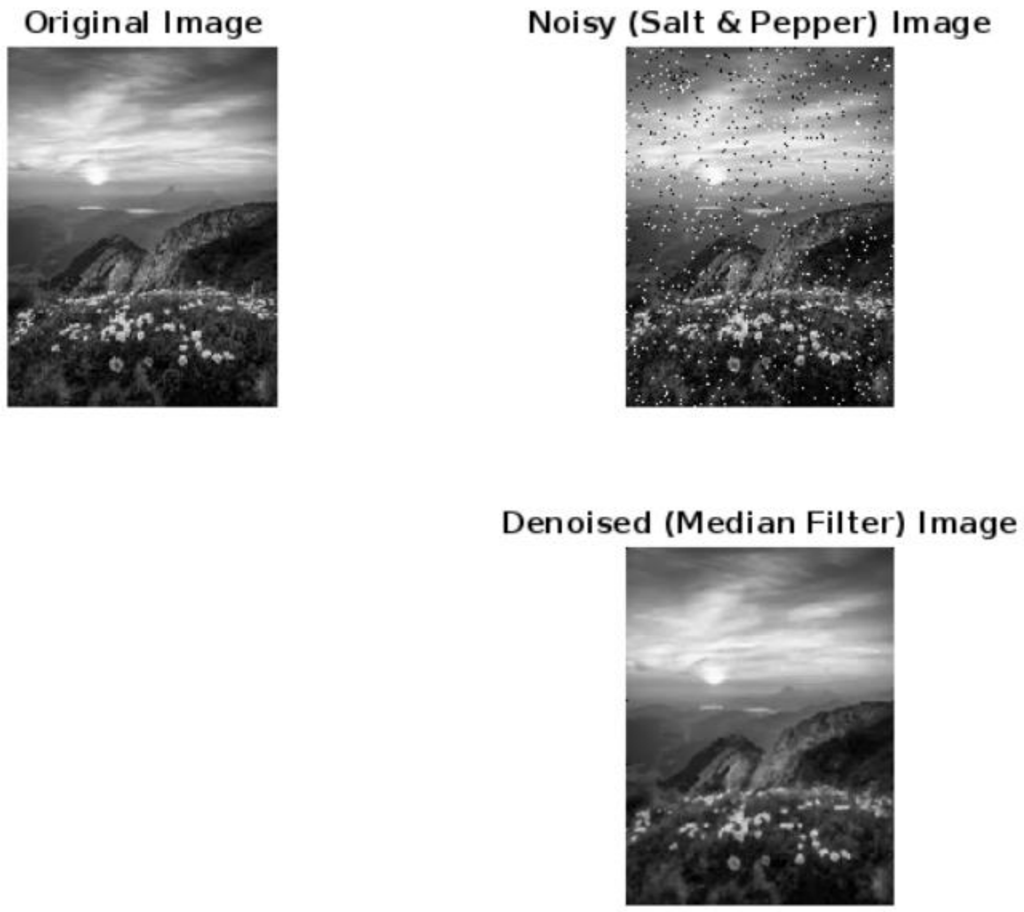

Figure 5.2: Noise Simulation (Salt and Pepper Noise Adding and Removing) using MATLAB

Explanation:

1. Image Reading and Grayscale Conversion.
2. `noisy_img = imnoise(img, 'salt & pepper', 0.05);` adds salt-and-pepper noise to the grayscale image with a noise density of 0.05. This means 5% of the pixels are randomly set to either black (pepper) or white (salt), simulating noise that resembles dust or scratches on a photo.
3. denoised_img = medfilt2(noisy_img, [3 3]); applies a median filter of size 3x3 to the noisy image. The median filter is effective in reducing salt-and-pepper noise, as it replaces each pixel's intensity with the median of the intensities in its local 3x3 neighborhood, removing isolated black and white spots.