Lab Report: 05
Title: Edge Detection
Course title: Digital Image Processing Laboratory
Course code: CSE-406
4th Year 1st Semester Examination 2023

Date of Submission: 29/09/2024

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

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka, Bangladesh

Experiment No: 01

Experiment Name: Edge Detection of image using Sobel Operator

Objectives:

1. Edge Detection
2. Gradient Approximation.
3. Noise Resistance

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('nature.jpeg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
sobel_edge = np.sqrt(sobelx**2 + sobely**2)
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(sobel_edge, cmap='gray')
plt.title('Edge Detection using Sobel Operator')
plt.axis('off')
plt.show()
```

Output:

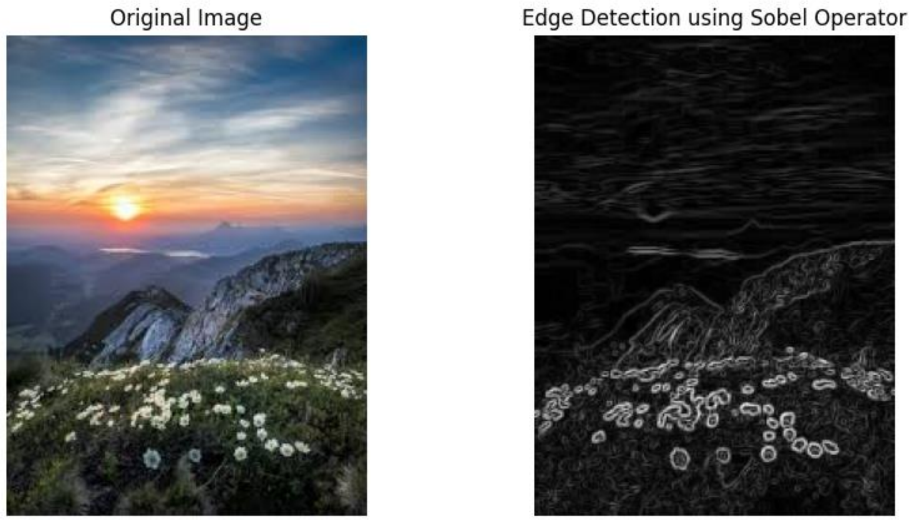

Figure 1.1: Showing the image detection using Sobel operator in python

Explanation:

1. Read the Input Image: The image is read using cv2.imread() from OpenCV.
2. Convert to Grayscale: The image is converted to grayscale using cv2.cvtColor() since edge detection is commonly applied to grayscale images.
3. Sobel Operator:
4. cv2.Sobel() is used to compute the gradient in the x and y directions. The ksize=3 parameter defines the kernel size.
5. The sobelx variable holds the horizontal edge gradients, and sobely holds the vertical edge gradients.
6. Calculate the Gradient Magnitude: The edge magnitude is calculated using the formula Gx2+Gy2\sqrt{G_x^2 + G_y^2}Gx2+Gy2, where sobelx and sobely are the gradients in the x and y directions.
7. Display the Images: matplotlib.pyplot is used to display the original and edge-detected images side by side.

Code-02: MATLAB

```
I = imread('nature.jpeg');
I_gray = rgb2gray(I);
Gx = [-1 0 1; -2 0 2; -1 0 1];
Gy = [-1 -2 -1; 0 0 0; 1 2 1];
Ix = imfilter(double(I_gray), Gx);
Iy = imfilter(double(I_gray), Gy);
SobelEdge = sqrt(Ix.^2 + Iy.^2);
figure;
subplot(1, 2, 1), imshow(I), title('Original Image');
subplot(1, 2, 2), imshow(uint8(SobelEdge)), title('Edge Detection using Sobel Operator');
```

Output:

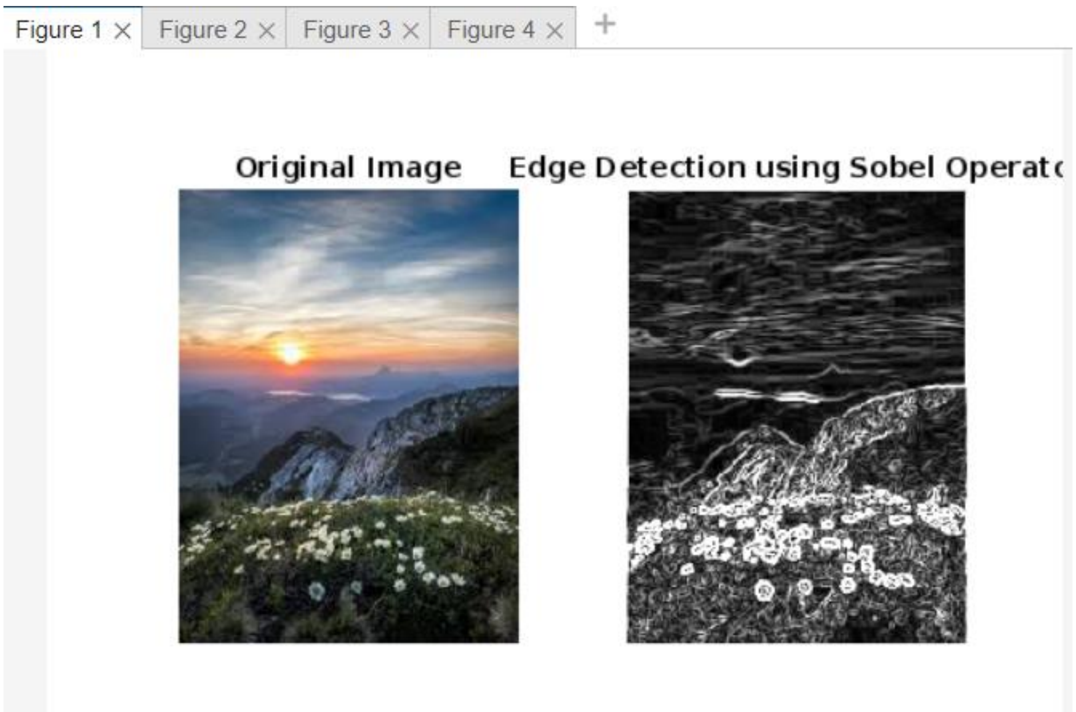

Figure 1.2: Edge detection using Sobel operator in MATLAB

Explanation:

1. **Read the Input Image:** The image is loaded using the imread() function..
2. **Convert to Grayscale:** The image is converted to grayscale using rgb2gray() to simplify the edge detection process, as edge detection usually works better in grayscale images.
3. **Sobel Operators:** Two Sobel kernels, GxG_xGx and GyG_yGy, are used to compute the gradient in the x and y directions.
4. **Gradient Magnitude**: The magnitude of the gradient is calculated as SobelEdge = sqrt(Ix.^2 + Iy.^2), giving the edge intensity.
5. **Display the Results:** The original image and the result of the Sobel edge detection are displayed side by side using subplot().

Experiment: 02

Experiment Name: Edge Detection of image using Prewitt Operator

Objectives:

1. Edge Detection
2. Gradient Approximation
3. Simplicity and Efficiency

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('nature.jpeg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
prewittx = np.array([[ -1,  0,  1],
                  [ -1,  0,  1],
                  [ -1,  0,  1]])

prewitty = np.array([[  1,  1,  1],
                  [  0,  0,  0],
                  [ -1, -1, -1]])
edge_x = cv2.filter2D(gray, -1, prewittx)
edge_y = cv2.filter2D(gray, -1, prewitty)
prewitt_edge = np.sqrt(edge_x**2 + edge_y**2)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
```

```
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(prewitt_edge, cmap='gray')
plt.title('Edge Detection using Prewitt Operator')
plt.axis('off')
plt.show()
```

Output:

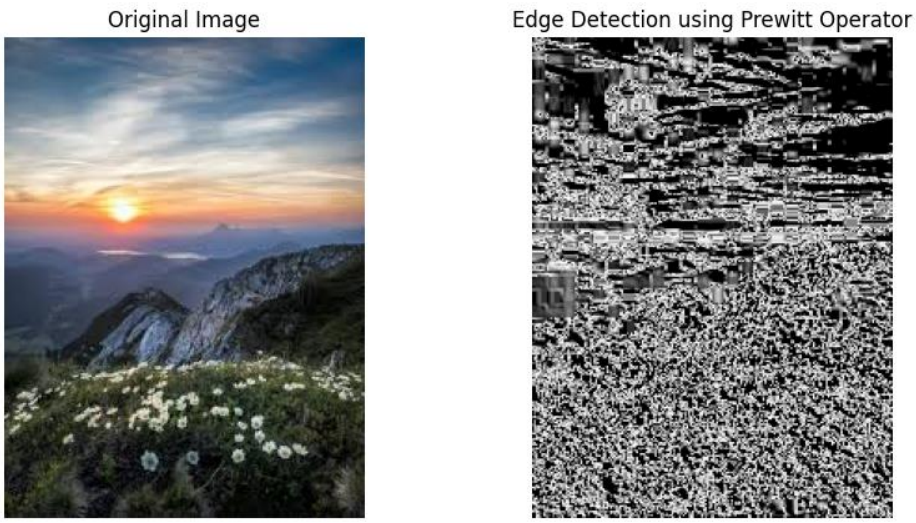

Figure 2.1: Edge detection using Prewitt operator in python code

Explanation:

1. **Read the Input Image:** The image is loaded using cv2.imread().
2. **Convert to Grayscale:** The image is converted to grayscale using cv2.cvtColor() since edge detection is more effective in grayscale.
3. **Prewitt Operator:**
   The Prewitt X and Y kernels are defined manually in prewittx and prewitty respectively. cv2.filter2D() is used to apply the Prewitt filter, which performs convolution with the Prewitt kernels on the grayscale image.
4. **Gradient Magnitude**: The edge magnitude is computed using Gx2+Gy2\sqrt{G_x^2 + G_y^2}Gx2+Gy2, where edge_x and edge_y represent the gradients in the x and y directions.
5. **Display the Images:** The original image and the Prewitt edge detection result are displayed side by side using matplotlib.pyplot.

Code-02: MATLAB

```
I = imread('nature.jpeg');
I_gray = rgb2gray(I);
Gx = [-1 0 1; -1 0 1; -1 0 1];
Gy = [-1 -1 -1; 0 0 0; 1 1 1];
Ix = imfilter(double(I_gray), Gx);
Iy = imfilter(double(I_gray), Gy);
PrewittEdge = sqrt(Ix.^2 + Iy.^2);
figure;
subplot(1, 2, 1), imshow(I), title('Original Image');
subplot(1, 2, 2), imshow(uint8(PrewittEdge)), title('Edge Detection using Prewitt Operator');
```

Output:

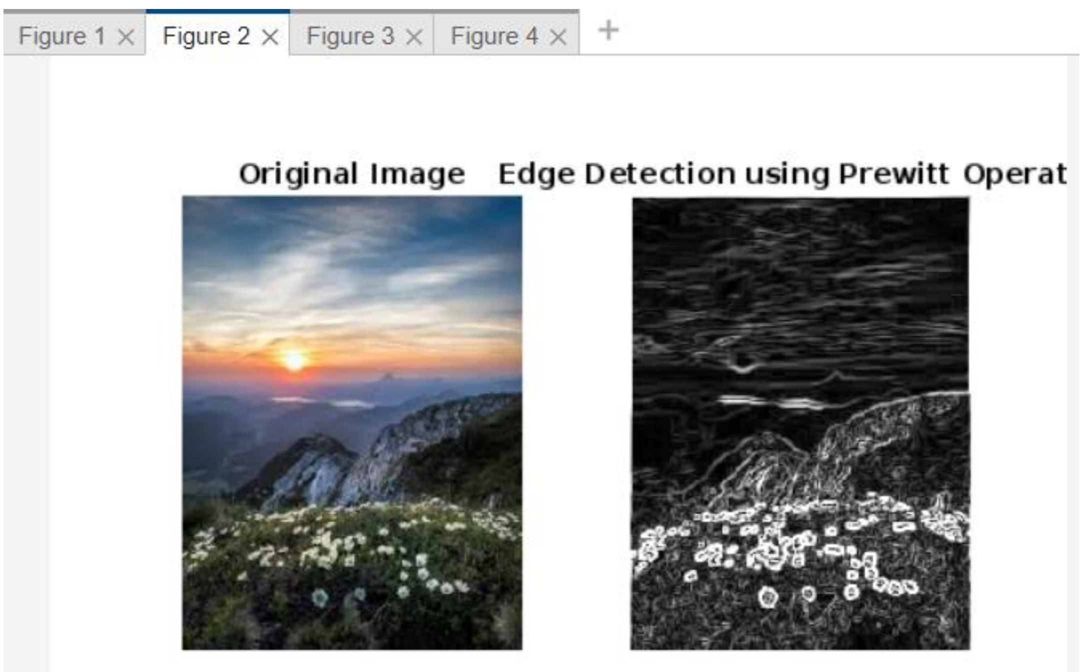

Figure 2.2: Edge detection using Prewitt operator in MATLAB

Explanation:

1. **Read the Input Image**: The image is loaded using the imread() function.
2. **Convert to Grayscale**: The image is converted to grayscale using rgb2gray() to simplify the edge detection process.

3. **Prewitt Operator**: Two Prewitt kernels, GxG_xGx and GyG_yGy, are used to compute the gradient in the x and y directions. The histogram of the input image is plotted using imhist.
4. **Gradient Magnitude**: The magnitude of the gradient is calculated as Ix2+Iy2\sqrt{I_x^2 + I_y^2}Ix2+Iy2, providing the edge intensity.
5. **Display the Results**: The original image and the result of the Prewitt edge detection are displayed side by side using subplot().

Experiment No: 03

Experiment Name: Edge Detection of image using Isotropic Operator

Objectives:

1. Edge Detection
2. Gradient Approximation
3. Simplicity and Efficiency

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('nature.jpeg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
isotropic_x = np.array([[-1, 0, 1],
                        [-np.sqrt(2), 0, np.sqrt(2)],
                        [-1, 0, 1]])

isotropic_y = np.array([[-1, -np.sqrt(2), -1],
                        [ 0, 0, 0],
                        [ 1, np.sqrt(2), 1]])
edge_x = cv2.filter2D(gray, -1, isotropic_x)
edge_y = cv2.filter2D(gray, -1, isotropic_y)
isotropic_edge = np.sqrt(edge_x**2 + edge_y**2)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
```

```
plt.imshow(isotropic_edge, cmap='gray')
plt.title('Edge Detection using Isotropic Operator')
plt.axis('off')
plt.show()
```

Output:

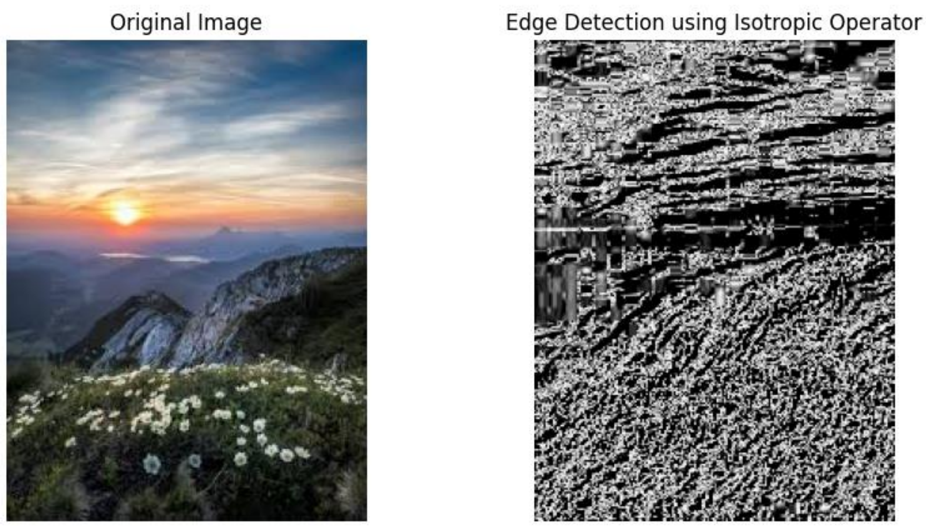

Figure 3.1: Edge detection using Isotropic operator in python code

Explanation:

1. **Read the Input Image**: The image is loaded using cv2.imread().
2. **Convert to Grayscale**: The image is converted to grayscale using cv2.cvtColor() since edge detection is commonly performed on grayscale images.
3. **Isotropic Operator**:
   The Isotropic operator kernels (isotropic_x and isotropic_y) are manually defined to approximate gradients in all directions (circular symmetry).
   cv2.filter2D() is used to apply these filters to the grayscale image.
4. **Gradient Magnitude**: The magnitude of the edge is calculated as Gx2+Gy2\sqrt{G_x^2 + G_y^2}Gx2+Gy2, where edge_x and edge_y represent the gradient in the x and y directions, respectively.
5. **Display the Results**: The original image and the result of the Isotropic edge detection are displayed side by side using matplotlib.pyplot.

Code-02: MATLAB

```
I = imread('nature.jpeg');
I_gray = rgb2gray(I);
Gx = [-1 0 1; -sqrt(2) 0 sqrt(2); -1 0 1];
Gy = [-1 -sqrt(2) -1; 0 0 0; 1 sqrt(2) 1];
Ix = imfilter(double(I_gray), Gx);
Iy = imfilter(double(I_gray), Gy);
IsotropicEdge = sqrt(Ix.^2 + Iy.^2);
figure;
subplot(1, 2, 1), imshow(I), title('Original Image');
subplot(1, 2, 2), imshow(uint8(IsotropicEdge)), title('Edge Detection using Isotropic Operator');
```

Output:

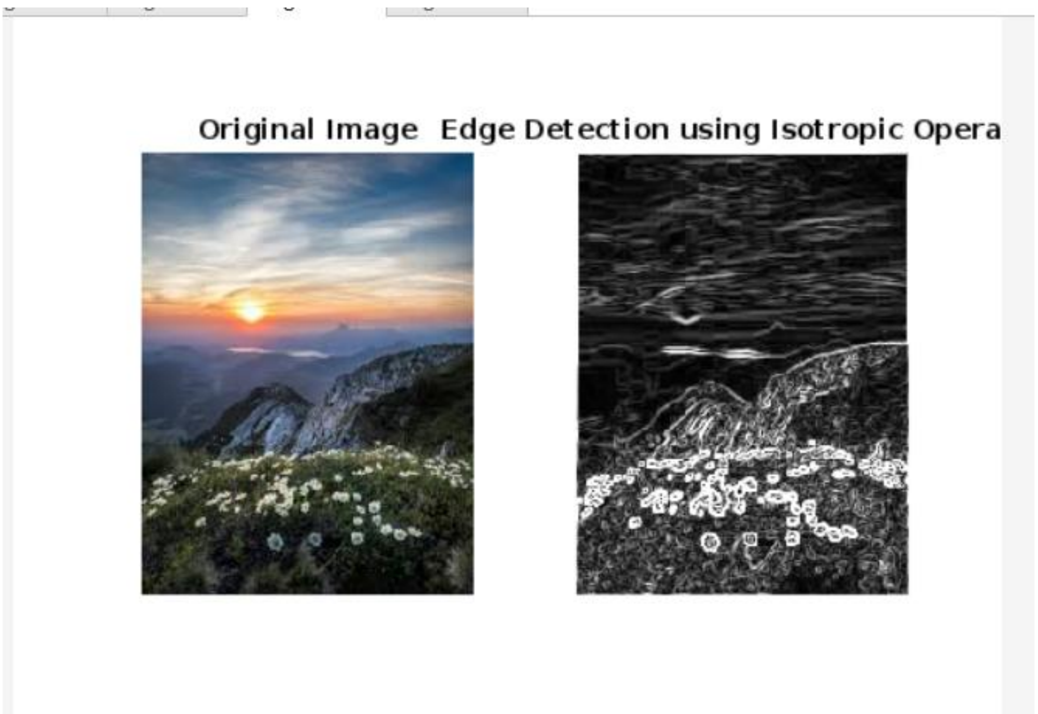

Figure 3.2: Edge detection using Isotropic operator in MATLAB

Explanation:

1. **Read the Input Image**: The image is loaded using the imread() function.
2. **Convert to Grayscale**: The image is converted to grayscale using rgb2gray() since edge detection works well on grayscale images.
3. **Isotropic Operator**: Two kernels, GxG_xGx and GyG_yGy, approximate edge detection in all directions by considering circular symmetry (approximating isotropic gradients).

4. **Gradient Magnitude**: The magnitude of the gradient is computed as Ix2+Iy2\sqrt{I_x^2 + I_y^2}Ix2+Iy2, giving the edge intensity in the image.
5. **Display the Results**: The original image and the result of the Isotropic edge detection are displayed side by side using subplot().

Experiment No: 04

Experiment Name: Edge Detection of image using Robert Operator

Objectives:

1. Edge Detection
2. Gradient Approximation
3. Simplicity and Efficiency

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('nature.jpeg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
roberts_x = np.array([[1, 0],
                      [0, -1]])
roberts_y = np.array([[0, 1],
                      [-1, 0]])
edge_x = cv2.filter2D(gray, -1, roberts_x)
edge_y = cv2.filter2D(gray, -1, roberts_y)
roberts_edge = np.sqrt(edge_x**2 + edge_y**2)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(roberts_edge, cmap='gray')
plt.title('Edge Detection using Roberts Operator')
plt.axis('off')
plt.show()
```

Output:

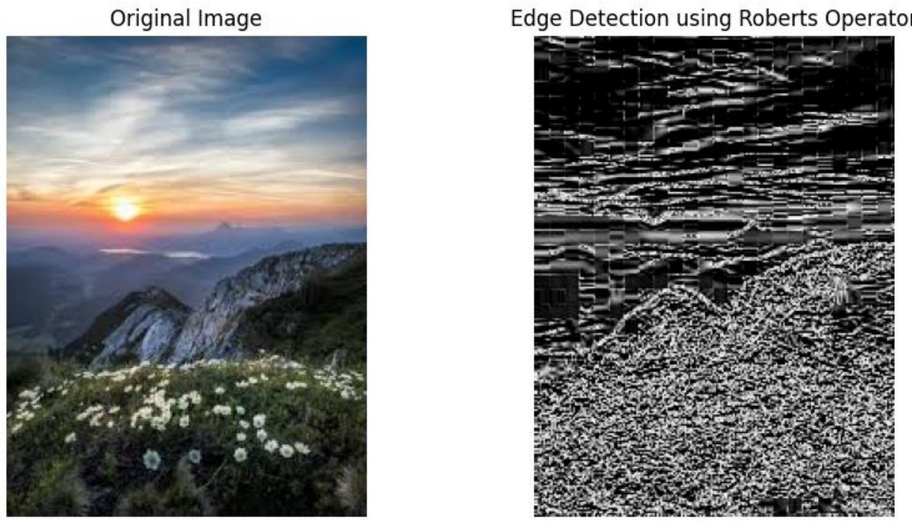

Figure 4.1: Edge detection using Robert operator in python code

Explanation:

1. **Read the Input Image**: The image is loaded using cv2.imread().
2. **Convert to Grayscale**: The image is converted to grayscale using cv2.cvtColor(), as edge detection is commonly applied on grayscale images.
3. **Roberts Operator**:
   The Roberts operator uses 2x2 kernels for diagonal edge detection. These kernels are manually defined (roberts_x and roberts_y).
   cv2.filter2D() is used to apply the Roberts filter on the grayscale image.
4. **Gradient Magnitude**: The gradient magnitude is calculated as Gx2+Gy2\sqrt{G_x^2 + G_y^2}Gx2+Gy2, where edge_x and edge_y represent the gradient in the x and y directions, respectively.
5. **Display the Results**: The original image and the edge-detected result using the Roberts operator are displayed side by side using matplotlib.pyplot.

Code-02: MATLAB

```
I = imread('nature.jpeg');
I_gray = rgb2gray(I);
Gx = [1 0; 0 -1];
Gy = [0 1; -1 0];
Ix = imfilter(double(I_gray), Gx);
Iy = imfilter(double(I_gray), Gy);
RobertsEdge = sqrt(Ix.^2 + Iy.^2);
figure;
subplot(1, 2, 1), imshow(I), title('Original Image');
subplot(1, 2, 2), imshow(uint8(RobertsEdge)), title('Edge Detection using Roberts Operator');
```

Output:

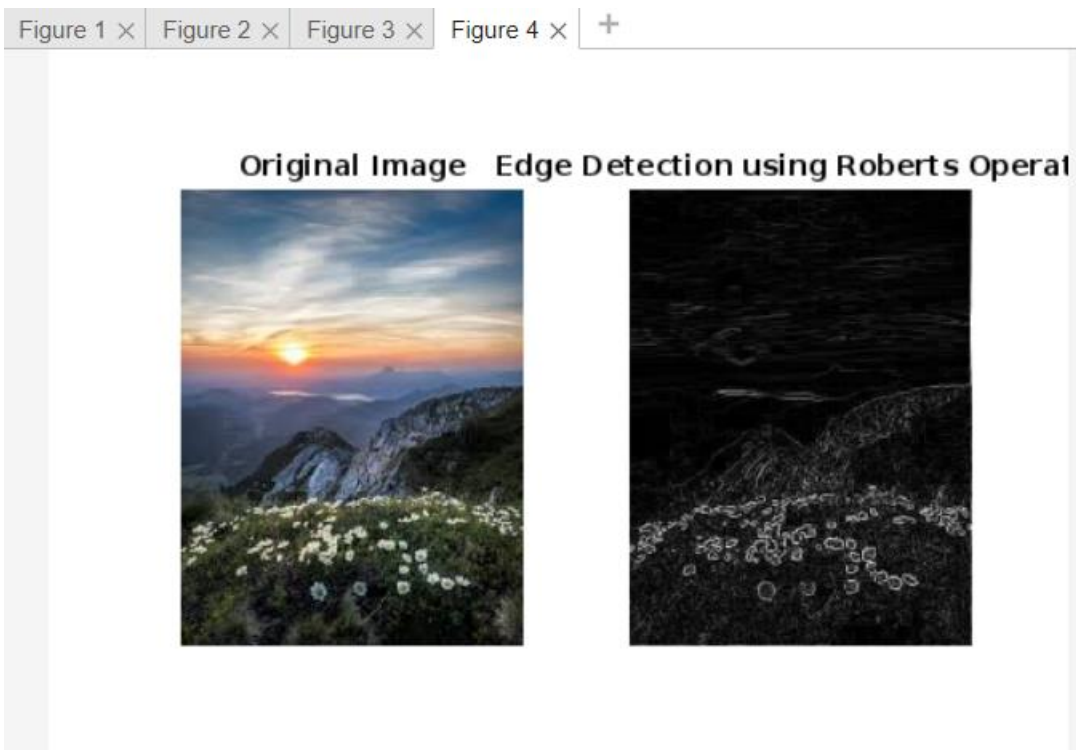

Figure 4.2: Edge detection using Robert operator in MATLAB

Explanation:

1. **Read the Input Image**: The image is loaded using the imread() function.
2. **Convert to Grayscale**: The image is converted to grayscale using rgb2gray(), as edge detection is typically performed on grayscale images.
3. **Roberts Operator**: Two Roberts kernels, GxG_xGx and GyG_yGy, are used to compute the gradient in diagonal directions. This operator is particularly sensitive to diagonal edges.
4. **Gradient Magnitude**: The magnitude of the gradient is calculated as Ix2+Iy2\sqrt{I_x^2 + I_y^2}Ix2+Iy2, which gives the edge intensity.
5. **Display the Results**: The original image and the result of the Roberts edge detection are displayed side by side using subplot().