Lab Report-05

Course title: Digital Image Processing Laboratory

Course code: CSE-406

4th Year 1st Semester Examination 2023

Date of Submission: 29 September 2024

Submitted to-

Dr. Morium Akter
Professor

Dr. Md. Golam Moazzam
Professor

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Sl	Class Roll	Exam Roll	Name
01	408	202220	Sudipta Singha

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

Lab Report Title

Edge Detection Using Isotropic Operation in Python

Introduction

Edge detection is a technique used to identify the boundaries within an image. The isotropic operation applies filters to detect edges in all directions uniformly, using kernels like the Laplacian or Sobel operator. It is effective in detecting regions of rapid intensity change, which correspond to object boundaries.

Python code

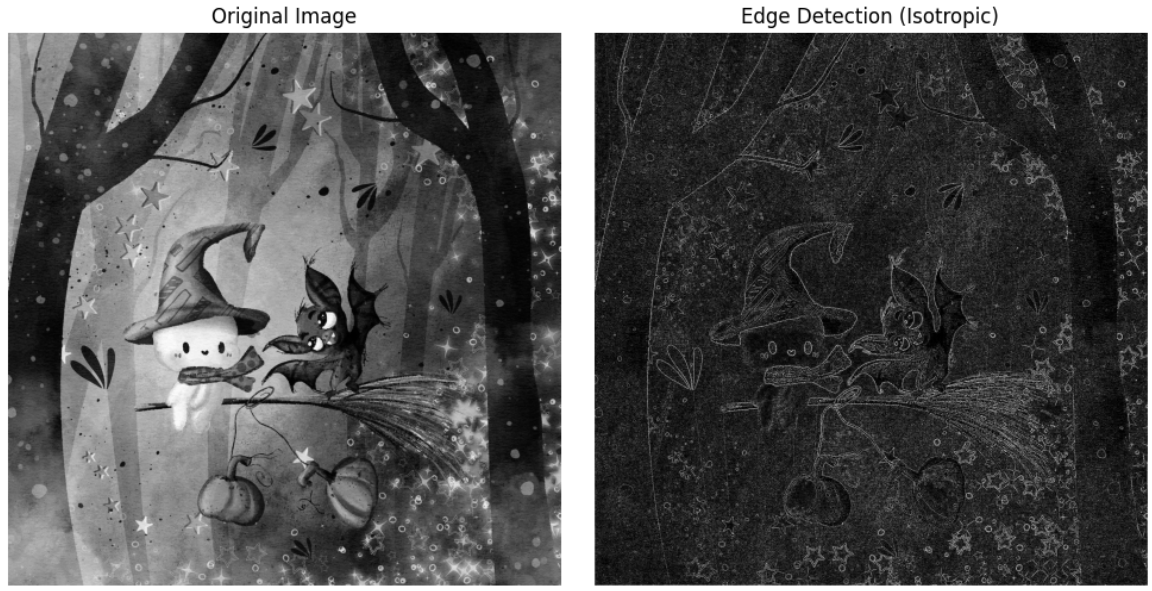
```
# isotropic_edge_detection.py

import numpy as np
from PIL import Image, ImageFilter
import matplotlib.pyplot as plt

def edge_detection_isotropic(input_path, output_path):
    img = Image.open(input_path).convert("L")  # Convert
       ↪ image to grayscale

    # Apply the isotropic edge detection (using a Laplacian
       ↪ filter)
    edge_img = img.filter(ImageFilter.FIND_EDGES)

    # Save the output image
    edge_img.save(output_path)

    # Display original and edge-detected images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap="gray")
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(edge_img, cmap="gray")
    plt.title("Edge Detection (Isotropic)")
    plt.axis("off")

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

```
# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "edge_detected.jpg"
    edge_detection_isotropic(input_path, output_path)
    print(f"Edge-detected image saved as {output_path}")
```

Input

Figure 1:

Output




```
Edge-detected image saved as edge_detected.jpg
```

Figure 2:

Lab Report Title

Edge Detection Using Prewitt Operator in Python

Introduction

The Prewitt Operator is a discrete differentiation operator used in edge detection. It works by convolving the image with two kernels that compute the gradient in the horizontal and vertical directions. The result highlights regions with high intensity changes, indicating edges.

Python code

```
# prewitt_edge_detection.py

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from scipy.ndimage import convolve

def prewitt_operator(img):
    # Define Prewitt kernels for horizontal and vertical
      ↪ edge detection
    kernel_x = np.array([[ -1, 0, 1],
                         [ -1, 0, 1],
                         [ -1, 0, 1]])

    kernel_y = np.array([[ -1, -1, -1],
                         [  0,  0,  0],
                         [  1,  1,  1]])

    # Apply convolution using Prewitt kernels
    grad_x = convolve(img, kernel_x)
    grad_y = convolve(img, kernel_y)

    # Compute the gradient magnitude
    grad_magnitude = np.hypot(grad_x, grad_y)
    grad_magnitude = grad_magnitude / grad_magnitude.max() *
      ↪ 255

    return grad_magnitude

def edge_detection_preitt(input_path, output_path):
    img = Image.open(input_path).convert("L")  # Convert
      ↪ image to grayscale
    img_array = np.array(img)
```

```
    # Apply Prewitt edge detection
    edge_img_array = prewitt_operator(img_array)

    # Convert the edge-detected image back to an image
    edge_img =
       ↪ Image.fromarray(edge_img_array.astype(np.uint8))

    # Save the output image
    edge_img.save(output_path)

    # Display original and edge-detected images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap="gray")
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(edge_img, cmap="gray")
    plt.title("Edge Detection (Prewitt Operator)")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "prewitt_edge_detected.jpg"
    edge_detection_preitt(input_path, output_path)
    print(f"Edge-detected image saved as {output_path}")
```

Input

Figure 3:

Output

Figure 4:

Lab Report Title

Edge Detection Using Robert Operator in Python

Introduction

The Roberts Operator is a simple, quick edge detection technique that uses a pair of 2x2 convolution kernels to calculate the gradient of an image. It highlights the edges by detecting high-frequency components in the image, typically emphasizing the diagonal edges.

Python code

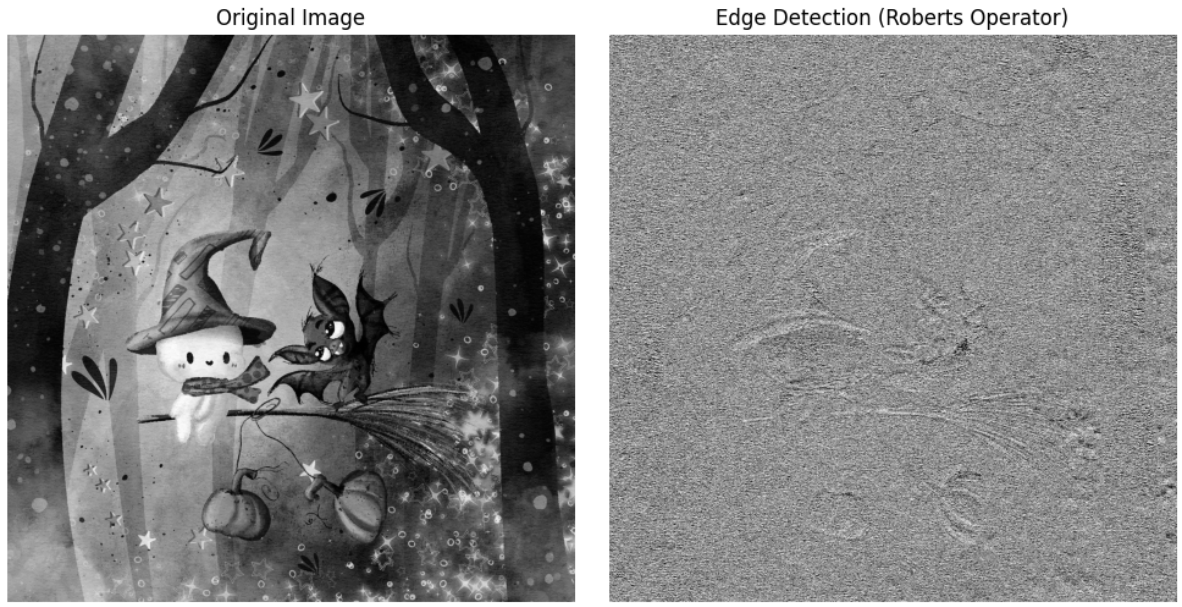
```
# roberts_edge_detection.py

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from scipy.ndimage import convolve

def roberts_operator(img):
    # Define Roberts operator kernels for diagonal edge
    ↪ detection
    kernel_x = np.array([[1, 0],
                         [0, -1]])

    kernel_y = np.array([[0, 1],
                         [-1, 0]])

    # Apply convolution using Roberts kernels
    grad_x = convolve(img, kernel_x)
    grad_y = convolve(img, kernel_y)

    # Compute the gradient magnitude
    grad_magnitude = np.hypot(grad_x, grad_y)
    grad_magnitude = grad_magnitude / grad_magnitude.max() *
    ↪ 255

    return grad_magnitude

def edge_detection_roberts(input_path, output_path):
    img = Image.open(input_path).convert("L")  # Convert
    ↪ image to grayscale
    img_array = np.array(img)

    # Apply Roberts edge detection
```

```
    edge_img_array = roberts_operator(img_array)

    # Convert the edge-detected image back to an image
    edge_img =
       ↪ Image.fromarray(edge_img_array.astype(np.uint8))

    # Save the output image
    edge_img.save(output_path)

    # Display original and edge-detected images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap="gray")
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(edge_img, cmap="gray")
    plt.title("Edge Detection (Roberts Operator)")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "roberts_edge_detected.jpg"
    edge_detection_roberts(input_path, output_path)
    print(f"Edge-detected image saved as {output_path}")
```

Input

Figure 5:

Output




```
Edge-detected image saved as roberts_edge_detected.jpg
```

Figure 6:

Lab Report Title

Edge Detection Using Sobel Operator in Python

Introduction

The Sobel Operator is one of the most widely used edge detection techniques in image processing. It uses two convolution kernels (horizontal and vertical) to compute the gradient of the image intensity at each pixel, emphasizing edges in the horizontal and vertical directions. The result is an image where edges are highlighted.

Python code

```
# sobel_edge_detection.py

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from scipy.ndimage import convolve

def sobel_operator(img):
    # Define Sobel kernels for horizontal and vertical edge
    ↪ detection
    kernel_x = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])

    kernel_y = np.array([[-1, -2, -1],
                         [ 0,  0,  0],
                         [ 1,  2,  1]])

    # Apply convolution using Sobel kernels
    grad_x = convolve(img, kernel_x)
    grad_y = convolve(img, kernel_y)

    # Compute the gradient magnitude
    grad_magnitude = np.hypot(grad_x, grad_y)
    grad_magnitude = grad_magnitude / grad_magnitude.max() *
    ↪ 255

    return grad_magnitude

def edge_detection_sobel(input_path, output_path):
    img = Image.open(input_path).convert("L")  # Convert
    ↪ image to grayscale
    img_array = np.array(img)
```

```
    # Apply Sobel edge detection
    edge_img_array = sobel_operator(img_array)

    # Convert the edge-detected image back to an image
    edge_img =
       ↪ Image.fromarray(edge_img_array.astype(np.uint8))

    # Save the output image
    edge_img.save(output_path)

    # Display original and edge-detected images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap="gray")
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(edge_img, cmap="gray")
    plt.title("Edge Detection (Sobel Operator)")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "sobel_edge_detected.jpg"
    edge_detection_sobel(input_path, output_path)
    print(f"Edge-detected image saved as {output_path}")
```

Input

Figure 7:

Output

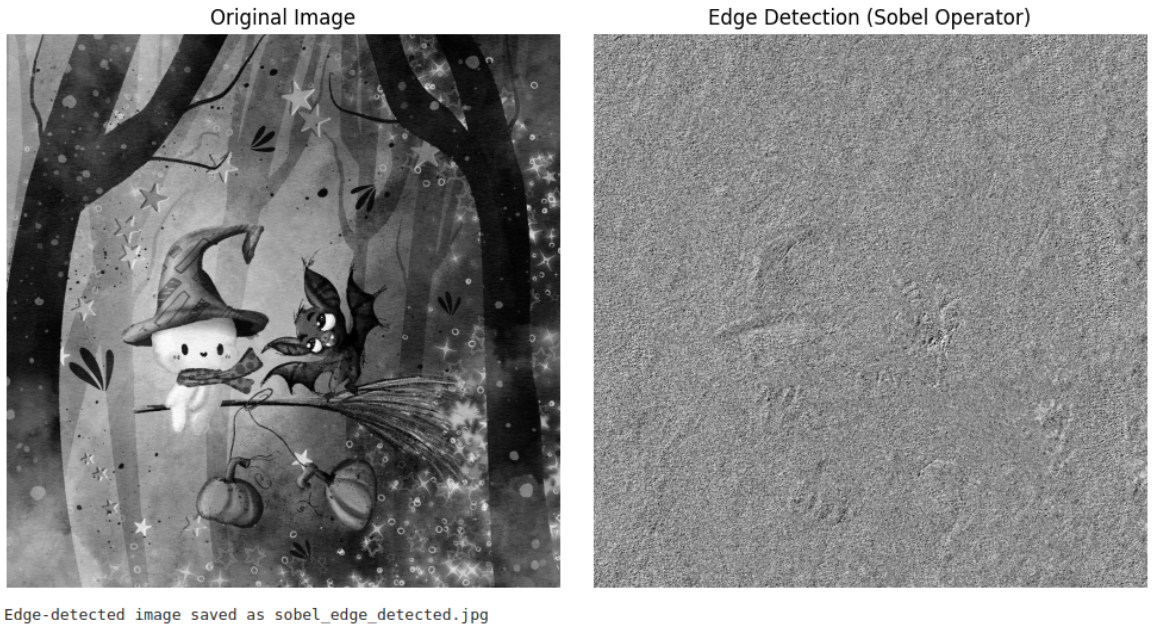


Figure 8: