Academic Year 2024-25



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJS22ITL603**

**COURSE NAME: Image Processing and Computer Vision Laboratory CLASS: T Y B. TECH**

**Name : Abhinav Nair Sap Id : 60003220045**

**Roll No : I011 Batch : I1-1**

**EXPERIMENT NO. 6 CO/LO:** Apply Image Segmentation Techniques.

**AIM / OBJECTIVE:** To apply Canny, Prewitt and Sobel Operators on a given image.

**EXERCISE**

Detect the edges or identify the regions in the image where the brightness of the image changes sharply using Canny, Prewitt, and Sobel edge detection operators using PIL/OpenCV libraries:

1. Read the image.
2. Convert into grayscale if it is colored.
3. Convert into the double format.
4. Define the mask or filter (Prewitt, Scharr and Sobel Operator). 5. Detect the edges Horizontal edges or along the x-axis, 6. Detect the edges Vertical Edges or along the y-axis.
5. Combine the edges detected along the X and Y axes.
6. Display the identified regions in the given images.

**Code**:

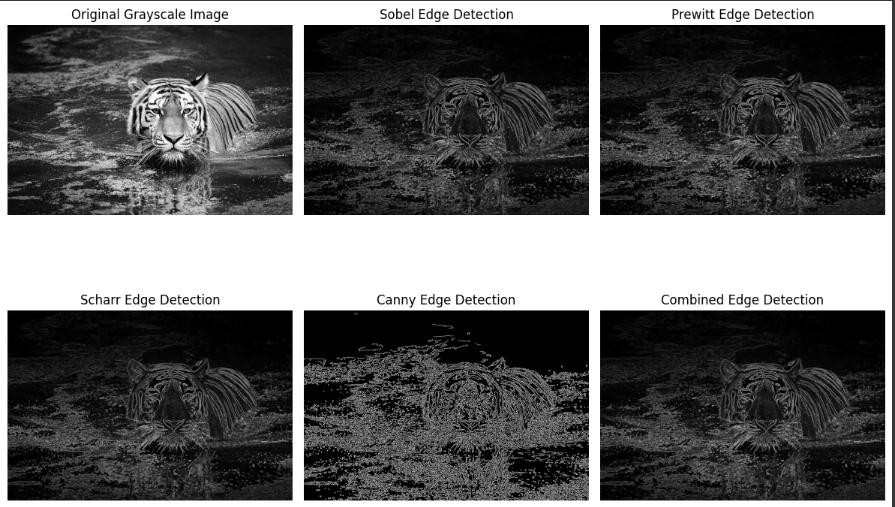
|  |
| --- |
| import cv2 import numpy as np from PIL import Image  import matplotlib.pyplot as plt    # Step 1: Read the image using PIL image\_path = /content/tiger-jpg.jpg # replace with your image path image = Image.open(image\_path)    # Step 2: Convert to grayscale if the image is colored gray\_image = image.convert('L')    # Step 3: Convert the image to a numpy array (double format for precision) gray\_image\_np = np.array(gray\_image, dtype=np.float64) |

|  |
| --- |
| # Step 4: Define the Sobel, Prewitt, and Scharr Operators  # Sobel operator kernels sobel\_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float64) sobel\_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.float64)  # Prewitt operator kernels prewitt\_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], dtype=np.float64) prewitt\_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]], dtype=np.float64)  # Scharr operator kernels (stronger edge response) scharr\_x = np.array([[3, 0, -3], [10, 0, -10], [3, 0, -3]], dtype=np.float64) scharr\_y = np.array([[3, 10, 3], [0, 0, 0], [-3, -10, -3]], dtype=np.float64)  # Step 5: Detect edges using the Sobel operator sobel\_x\_edges = cv2.filter2D(gray\_image\_np, -1, sobel\_x) sobel\_y\_edges = cv2.filter2D(gray\_image\_np, -1, sobel\_y)  # Step 6: Detect edges using the Prewitt operator prewitt\_x\_edges = cv2.filter2D(gray\_image\_np, -1, prewitt\_x) prewitt\_y\_edges = cv2.filter2D(gray\_image\_np, -1, prewitt\_y)  # Step 7: Detect edges using the Scharr operator scharr\_x\_edges = cv2.filter2D(gray\_image\_np, -1, scharr\_x) scharr\_y\_edges = cv2.filter2D(gray\_image\_np, -1, scharr\_y)  # Step 8: Combine the edges detected along X and Y axes sobel\_edges = np.sqrt(sobel\_x\_edges\*\*2 + sobel\_y\_edges\*\*2) prewitt\_edges = np.sqrt(prewitt\_x\_edges\*\*2 + prewitt\_y\_edges\*\*2) scharr\_edges = np.sqrt(scharr\_x\_edges\*\*2 + scharr\_y\_edges\*\*2)  # Canny Edge detection for comparison canny\_edges = cv2.Canny(gray\_image\_np.astype(np.uint8), 100, 200)  # Display the images with detected edges plt.figure(figsize=(12, 12))    # Show original grayscale image plt.subplot(3, 3, 1)  plt.imshow(gray\_image, cmap='gray') plt.title('Original Grayscale Image') plt.axis('off')    # Show Sobel edge detection result plt.subplot(3, 3, 2) plt.imshow(sobel\_edges, cmap='gray') plt.title('Sobel Edge Detection') plt.axis('off')    # Show Prewitt edge detection result plt.subplot(3, 3, 3) plt.imshow(prewitt\_edges, cmap='gray') |
| plt.title('Prewitt Edge Detection') plt.axis('off')    # Show Scharr edge detection result plt.subplot(3, 3, 4) plt.imshow(scharr\_edges, cmap='gray') plt.title('Scharr Edge Detection') plt.axis('off')    # Show Canny edge detection result plt.subplot(3, 3, 5) plt.imshow(canny\_edges, cmap='gray') plt.title('Canny Edge Detection') plt.axis('off')    # Combine all edges into a single image (for comparison) combined\_edges = np.maximum.reduce([sobel\_edges, prewitt\_edges, scharr\_edges, canny\_edges])    # Show combined edges plt.subplot(3, 3, 6) plt.imshow(combined\_edges, cmap='gray') plt.title('Combined Edge Detection') plt.axis('off')    # Adjust layout and show the image plt.tight\_layout() plt.show() |

**Input:**



**Output**:



**QUESTIONS:**

* What are the derivative operators useful in image segmentation? Explain their role in segmentation?
* Comparing Edge Detection Methods?
* Why the sum of all the elements in Edge detection mask is zero?

**REFERENCES:**

**Website References:**

1. Medium “Comparing Edge Detection Methods,” Available: [https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e.](https://medium.com/%40nikatsanka/comparing-edge-detection-methods-638a2919476e)
2. OpenCV, “Image Gradients,” *OpenCV Documentation*. Available: [https://docs.opencv.org/4.x/d5/d0f/tutorial\_py\_gradients.html.](https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html)