**EXPERIMENT 6**

|  |  |
| --- | --- |
| **Name: Shivam Bhosle** | **Roll No: 2** |
| **Subject: Computer Vision (CV)** | **Class/Batch: B1** |
| **Date of Performance:** | **Date of Submission:** |

|  |
| --- |
| **AIM** |

# **To perform Edge detection on an image using opencv**

|  |
| --- |
| **Theory** |

**What is Edge detection?**

Edge detection is a technique used in image processing and computer vision to identify and locate sharp discontinuities in an image, which often correspond to significant changes in intensity or colour. These discontinuities typically indicate the boundaries of objects within the image.

The primary goal of edge detection is to simplify the image representation while preserving important structural features. Common methods for edge detection include:

1. **Sobel Operator** :

* Uses two 3x3 kernels (one for detecting horizontal edges and one for vertical).
* Computes the gradient magnitude, which highlights regions with high intensity changes.
* Simple and effective for many applications but sensitive to noise.

1. **Canny Edge Detector**:
   * A multi-step process that includes:
     + **Smoothing**: Reduces noise using a Gaussian filter.
     + **Finding Gradients**: Calculates the gradient magnitude and direction.
     + **Non-Maximum Suppression**: Thin out the edges by keeping only local maxima.
     + **Double Thresholding**: Uses two thresholds to identify strong and weak edges.
     + **Edge Tracking by Hysteresis**: Connects weak edges to strong edges based on connectivity.
     + Known for its accuracy and is widely used in practice.
2. **Laplacian of Gaussian (LoG)**:

* Combines Gaussian smoothing and Laplacian operator to find edges.
* Highlights areas of rapid intensity change by looking for zero-crossings in the Laplacian.
* Sensitive to noise, so Gaussian smoothing is crucial.

1. **Prewitt Operator**: Similar to Sobel, this method also uses convolution masks to detect edges but focuses on simpler gradient approximations.
2. **Roberts Cross Operator**: Uses diagonal kernels to compute gradients, emphasizing edge directionality.

Edge detection is crucial for various applications, such as object recognition, image segmentation, and feature extraction.

**Importance of Edge Detection**

1. **Object Identification**: Edges often delineate the boundaries of objects, making them essential for recognizing shapes and forms.
2. **Image Segmentation**: By identifying edges, images can be segmented into meaningful regions for further analysis.
3. **Feature Extraction**: Edges are key features that can be used for tasks like matching, tracking, and classification.

**Key Concepts**

* **Gradient**: The gradient measures the change in intensity at a point in the image. It helps identify where edges occur. Strong gradients usually indicate edges.
* **Noise Reduction**: Before edge detection, images are often smoothed to reduce noise, which can lead to false edges. Techniques like Gaussian blurring are commonly used.

**Applications**

* **Medical Imaging**: Helps in identifying structures like tumors in MRI or CT scans.
* **Autonomous Vehicles**: Critical for obstacle detection and lane tracking.
* **Facial Recognition**: Assists in identifying facial features by outlining edges.
* **Image Compression**: Edges can be used to reduce the amount of data needed for images while maintaining important visual information.

**Challenges**

* **Noise**: Edges can be lost in noisy images or incorrectly identified due to variations in lighting.
* **Edge Thickness**: Some methods may produce thick edges, making precise identification challenging.
* **Multiple Edges**: In complex scenes, overlapping edges can lead to confusion in edge detection.

**Procedure**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image

image\_path = r"C:\\Users\\Shivam 007\\Downloads\\Sports Law.jpeg" # Replace with your image path

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Function to display images

def display\_images(images, titles):

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

for i in range(len(images)):

plt.subplot(2, len(images) // 2, i + 1)

plt.imshow(images[i], cmap='gray')

plt.title(titles[i])

plt.axis('off')

plt.show()

# Enhanced Sobel Edge Detection

def enhanced\_sobel(image):

# Apply Gaussian Blur to reduce noise before edge detection

blurred = cv2.GaussianBlur(image, (5, 5), 1.5)

# Apply Sobel filters

sobel\_x = cv2.Sobel(blurred, cv2.CV\_64F, 1, 0, ksize=5)

sobel\_y = cv2.Sobel(blurred, cv2.CV\_64F, 0, 1, ksize=5)

# Compute magnitude and angle

sobel\_magnitude = cv2.magnitude(sobel\_x, sobel\_y)

sobel\_angle = cv2.phase(sobel\_x, sobel\_y, angleInDegrees=True)

# Normalize to the range 0-255

sobel\_normalized = cv2.normalize(sobel\_magnitude, None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX)

# Adaptive Thresholding to enhance edge visibility

sobel\_edges = cv2.adaptiveThreshold(np.uint8(sobel\_normalized), 255,

cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,

cv2.THRESH\_BINARY, 11, 2)

return sobel\_edges

# Perform enhanced Sobel edge detection

sobel\_edges = enhanced\_sobel(image)

# 2. Canny Edge Detection for comparison

canny\_edges = cv2.Canny(image, 200, 100)

# 5. Fuzzy Logic Edge Detection

def fuzzy\_edge\_detection(image):

blurred = cv2.GaussianBlur(image, (5, 5), 1.5)

edges = cv2.Laplacian(blurred, cv2.CV\_64F)

fuzzy\_edges = np.where(edges > np.mean(edges), 255, 0).astype(np.uint8)

return fuzzy\_edges

fuzzy\_edges = fuzzy\_edge\_detection(image)

# Display results

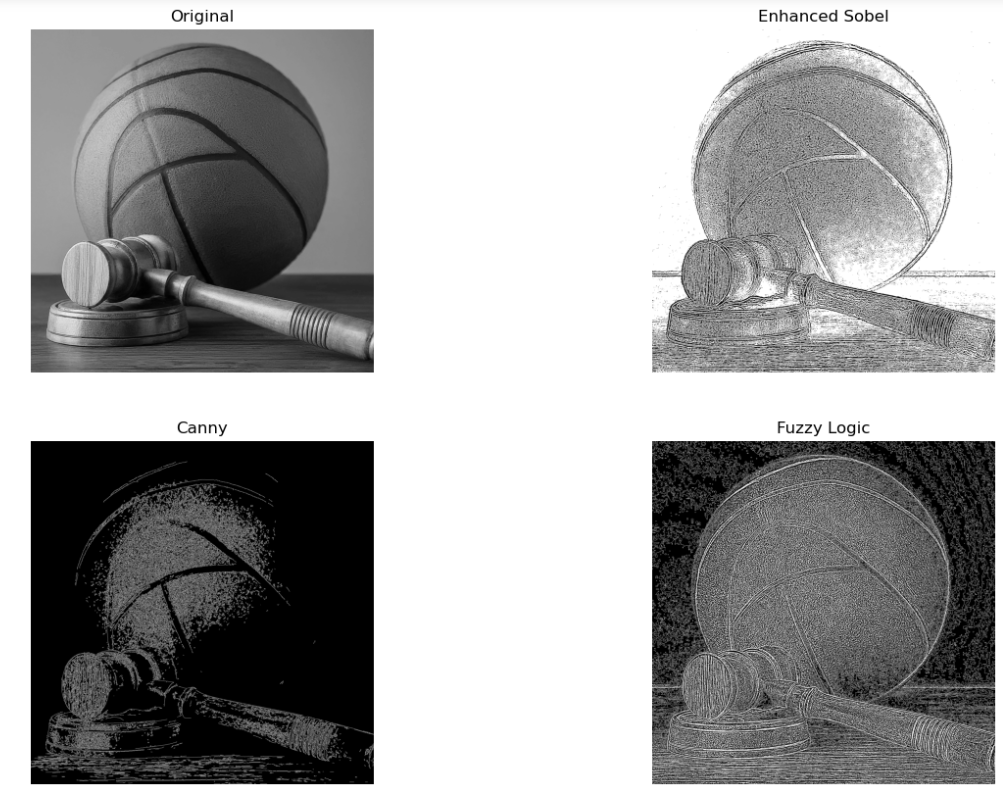
images = [image, sobel\_edges, canny\_edges, fuzzy\_edges]

titles = ['Original', 'Enhanced Sobel', 'Canny', 'Fuzzy Logic']

display\_images(images, titles)

**Output**

4



|  |
| --- |
| **Conclusion** |

# With this experiment we have successfully learned to implement how to perform Edge detection using opencv

|  |
| --- |
| **Assessment** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Timely Submission**  **(7)** | **Presentation**  **(06)** | **Understanding**  **(12)** | **Total**  **(25)** | **Sign** |
|  |  |  |  |  |