Experiment 5

Edge Detection and Template Matching

Aim:

To implement Edge detection techniques and template matching in OpenCV.

Software/ Packages Used:

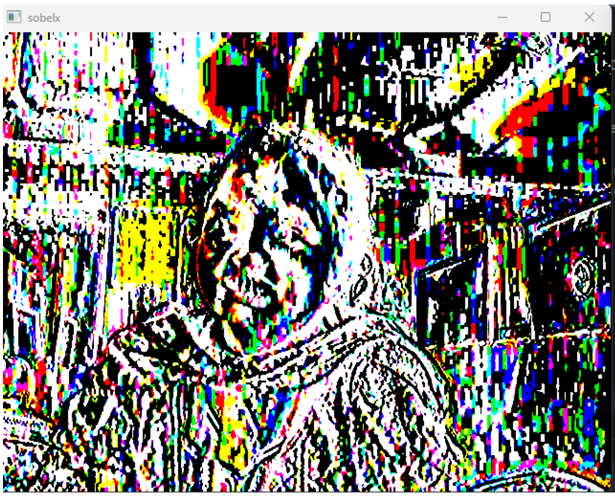
1. Pycharm IDE
2. Libraries used:
   - NumPy
   - opencv-python
   - matplotlib
   - scipy

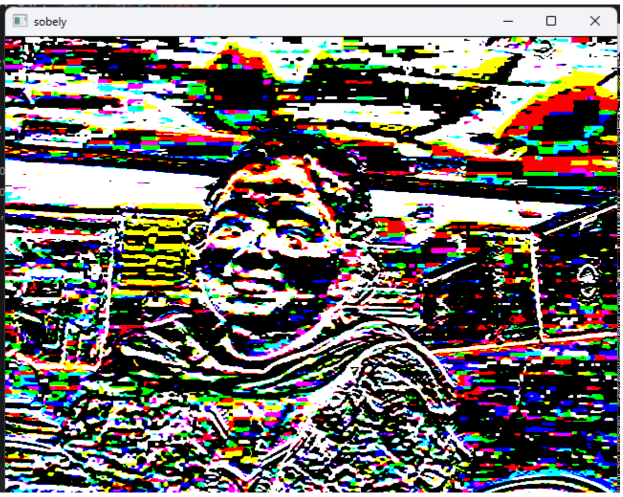
1) SOBEL LIVE CAPTURE:

PROGRAMS:

```
# Python program to Edge detection
# using OpenCV in Python
# using Sobel edge detection
# and laplacian method
import cv2
import numpy as np

# Capture livestream video content from camera 0
cap = cv2.VideoCapture(0)
while (1):
    # Take each frame
    _, frame = cap.read()
    # Convert to HSV for simpler calculations
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Calculation of Sobelx
    sobelx = cv2.Sobel(frame, cv2.CV_64F, 1, 0, ksize=5)
    # Calculation of Sobely
    sobely = cv2.Sobel(frame, cv2.CV_64F, 0, 1, ksize=5)
    cv2.imshow('sobelx', sobelx)
    cv2.imshow('sobely', sobely)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
# release the frame
cap.release()
```

OUTPUT:



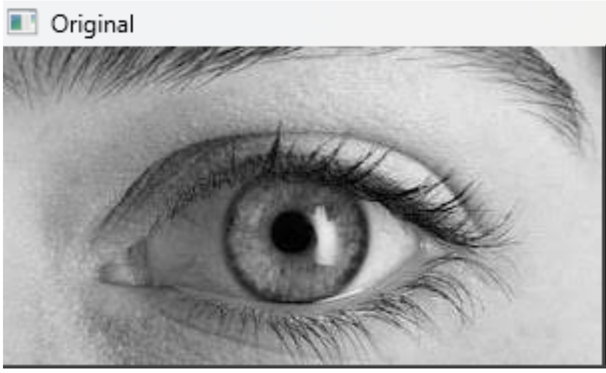


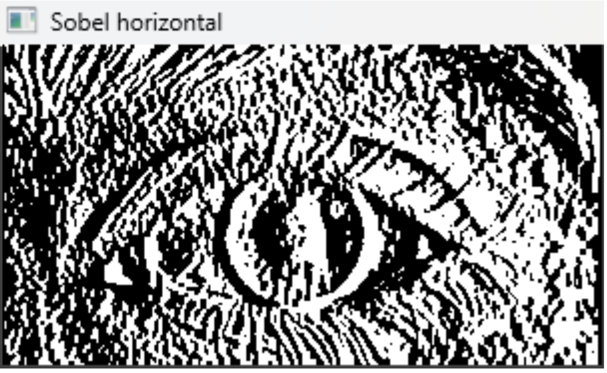
2)SOBEL IMAGE:

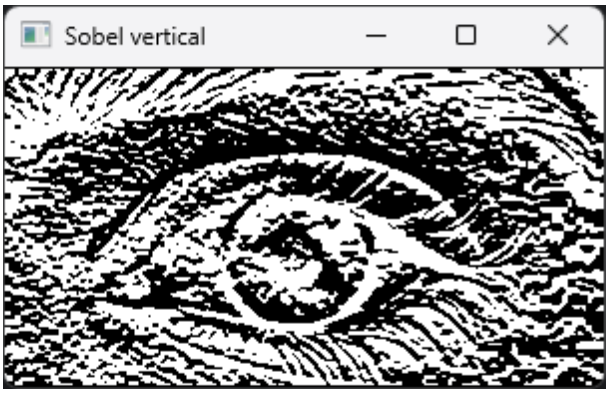
PROGRAM:

```
import cv2
import numpy as np

img = cv2.imread('images.jpg', cv2.IMREAD_GRAYSCALE)
rows, cols = img.shape

sobel_horizontal = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
sobel_vertical = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)

cv2.imshow('Original', img)
cv2.imshow('Sobel horizontal', sobel_horizontal)
cv2.imshow('Sobel vertical', sobel_vertical)

cv2.waitKey(0)
```

OUTPUT:





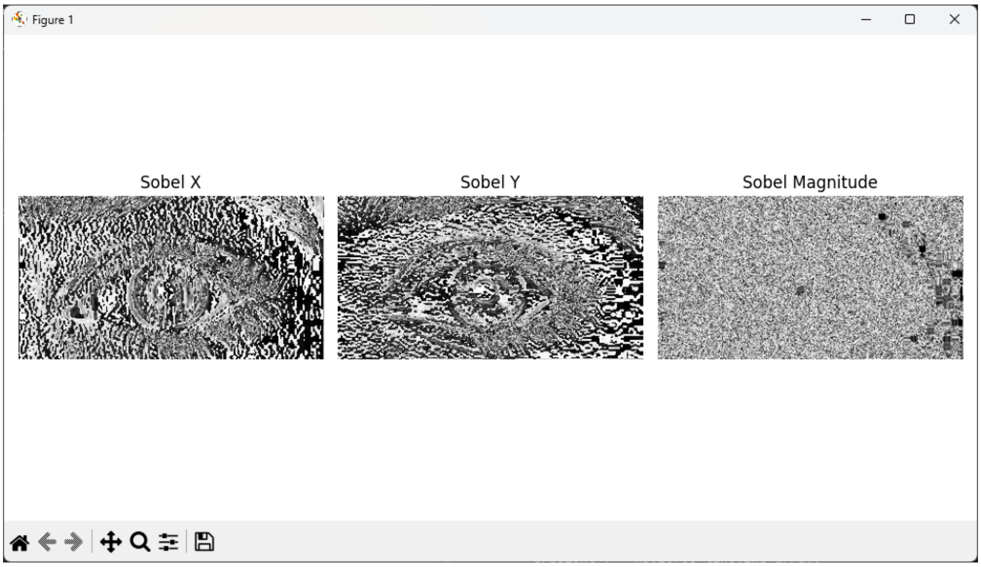


3)SOBEL MATRIX:

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
def sobel_operator(image):
    # Sobel kernels for gradient calculation
    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]])

    # Convert the image to grayscale
    grayscale_image = image.convert("L")
    # Convert the image to a NumPy array
    img_array = np.array(grayscale_image)
    # Pad the image to handle boundaries
    padded_image = np.pad(img_array, pad_width=1, mode='constant', constant_values=0)
    # Initialize empty arrays for gradient values
    gradient_x = np.zeros_like(img_array)
    gradient_y = np.zeros_like(img_array)
    # Convolve the image with Sobel kernels
    for i in range(img_array.shape[0]):
        for j in range(img_array.shape[1]):
            gradient_x[i, j] = np.sum(kernel_x * padded_image[i:i + 3, j:j + 3])
            gradient_y[i, j] = np.sum(kernel_y * padded_image[i:i + 3, j:j + 3])
    # Combine gradient magnitudes in both x and y directions
    gradient_magnitude = np.sqrt(gradient_x ** 2 + gradient_y ** 2)
    return gradient_x, gradient_y, gradient_magnitude
# Load an image
input_image = Image.open("images.jpg")  # Replace with your image path
# Apply Sobel operator
sobel_x, sobel_y, sobel_mag = sobel_operator(input_image)
# Display the results
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.title('Sobel X')
plt.imshow(sobel_x, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.title('Sobel Y')
plt.imshow(sobel_y, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.title('Sobel Magnitude')
plt.imshow(sobel_mag, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()
```

OUTPUT:

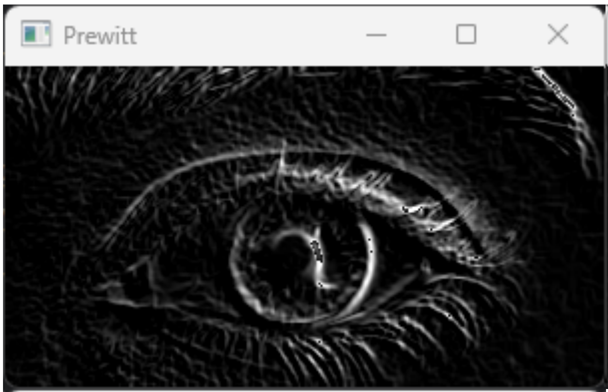


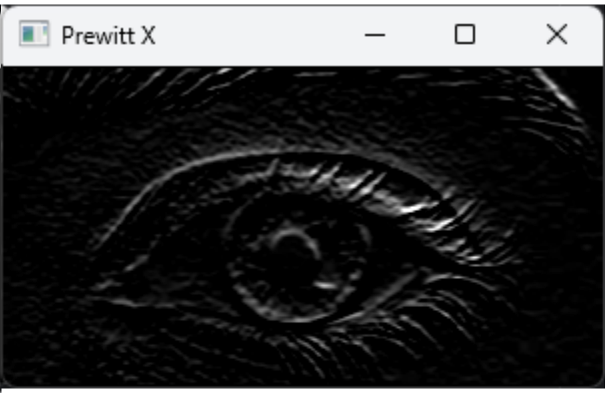
4)PREWITT WITH INBUILT FUNCTION:

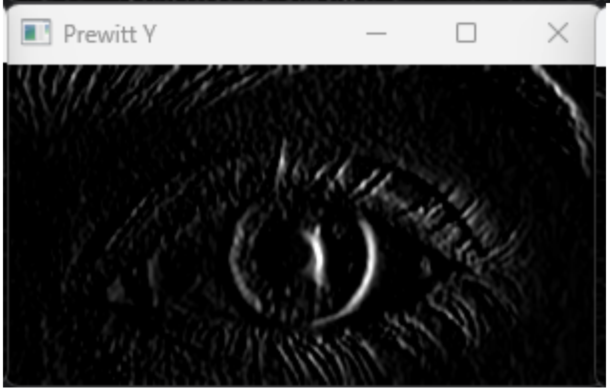
PROGRAM:

```
import cv2
import numpy as np
img = cv2.imread('images.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_gaussian = cv2.GaussianBlur(gray,(3,3),0)
#prewitt
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
img_prewittx = cv2.filter2D(img_gaussian, -1, kernelx)
img_prewitty = cv2.filter2D(img_gaussian, -1, kernely)
cv2.imshow("Original Image", img)
cv2.imshow("Prewitt X", img_prewittx)
cv2.imshow("Prewitt Y", img_prewitty)
cv2.imshow("Prewitt", img_prewittx + img_prewitty)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:





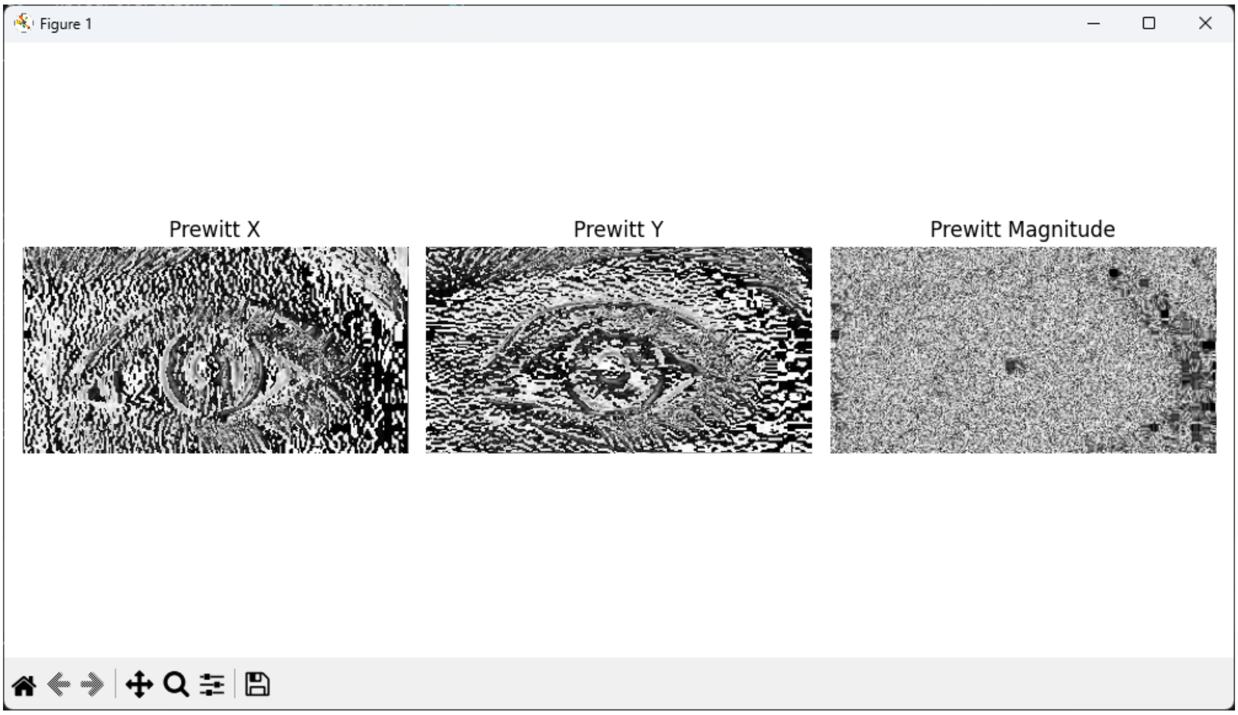


5) **PREWITT WITHOUT INBUILT:**

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
def prewitt_operator(image):
   # Prewitt kernels for gradient calculation
   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]])

   # Convert the image to grayscale
   grayscale_image = image.convert("L")
   # Convert the image to a NumPy array
   img_array = np.array(grayscale_image)
   # Pad the image to handle boundaries
   padded_image = np.pad(img_array, pad_width=1, mode='constant', constant_values=0)
   # Initialize empty arrays for gradient values
   gradient_x = np.zeros_like(img_array)
   gradient_y = np.zeros_like(img_array)
   # Convolve the image with Prewitt kernels
   for i in range(img_array.shape[0]):
      for j in range(img_array.shape[1]):
            gradient_x[i, j] = np.sum(kernel_x * padded_image[i:i + 3, j:j + 3])
            gradient_y[i, j] = np.sum(kernel_y * padded_image[i:i + 3, j:j + 3])
    # Combine gradient magnitudes in both x and y directions
    gradient_magnitude = np.sqrt(gradient_x ** 2 + gradient_y ** 2)
    return gradient_x, gradient_y, gradient_magnitude
# Load an image
input_image = Image.open("images.jpg")  # Replace with your image path
# Apply Prewitt operator
prewitt_x, prewitt_y, prewitt_mag = prewitt_operator(input_image)
# Display the results
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.title('Prewitt X')
plt.imshow(prewitt_x, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.title('Prewitt Y')
plt.imshow(prewitt_y, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.title('Prewitt Magnitude')
plt.imshow(prewitt_mag, cmap='gray')
plt.axis('off')
plt.tight_layout()
```

OUTPUT:

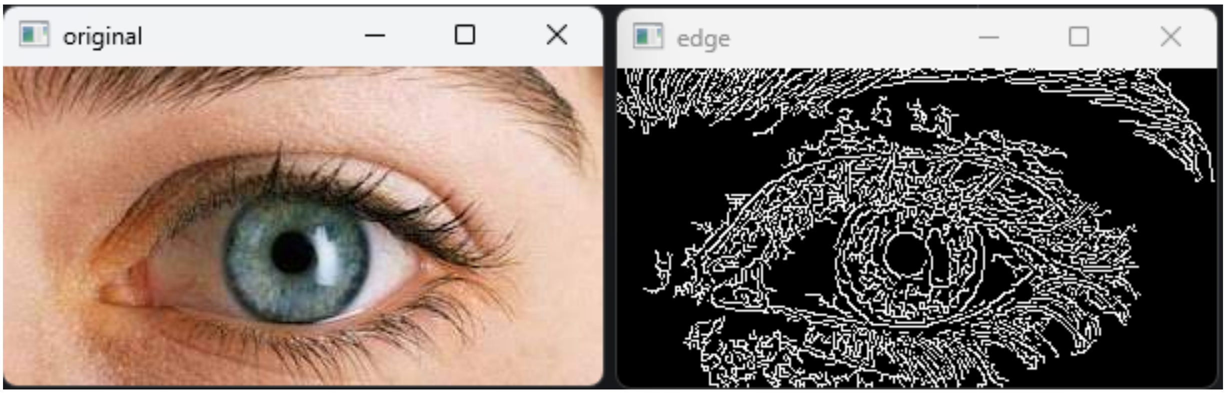


6) CANNY IMAGE:

PROGRAM:

```
import cv2
img = cv2.imread("images.jpg") # Read image
# Setting parameter values
t_lower = 50 # Lower Threshold
t_upper = 150 # Upper threshold
# Applying the Canny Edge filter
edge = cv2.Canny(img, t_lower, t_upper)
cv2.imshow('original', img)
cv2.imshow('edge', edge)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

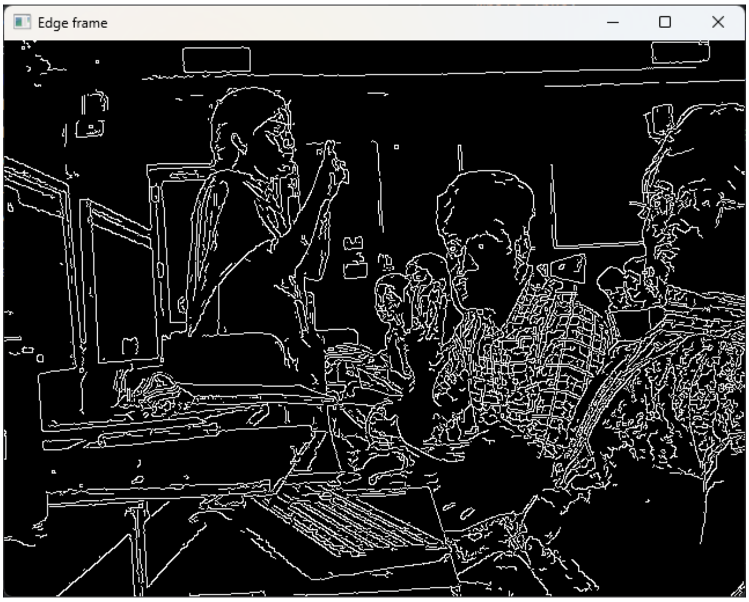


7) CANNY VIDEO:

PROGRAM:

```
import cv2
vcapture = cv2.VideoCapture(0)
while True:
    ret, frame = vcapture.read()
    if ret == True:
        grayscale = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        edge = cv2.Canny(grayscale, 75, 125)
        cv2.imshow('Edge frame', edge)
        if cv2.waitKey(20) == ord('q'):
            break
vcapture.release()
```

OUTPUT:

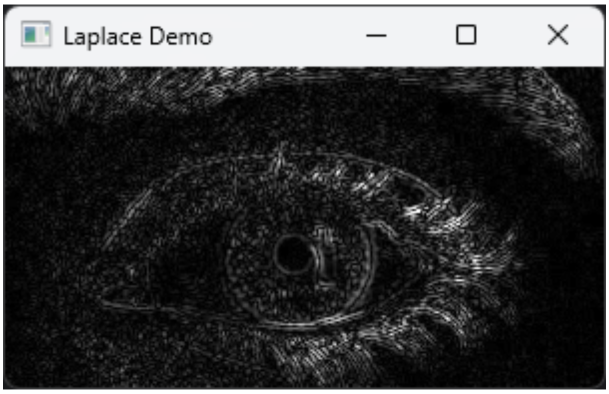


8)LAPLACE IMAGE:

PROGRAM:

```
import sys
import cv2 as cv
def main(argv):
    ddepth = cv.CV_16S
    kernel_size = 3
    window_name = "Laplace Demo"
    imageName = argv[0] if len(argv) > 0 else 'images.jpg'
    src = cv.imread(cv.samples.findFile(imageName), cv.IMREAD_COLOR) # Load an image
    if src is None:
        print ('Error opening image')
        return -1
    src = cv.GaussianBlur(src, (3, 3), 0)
    src_gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
    cv.namedWindow(window_name, cv.WINDOW_AUTOSIZE)
    dst = cv.Laplacian(src_gray, ddepth, ksize=kernel_size)
    abs_dst = cv.convertScaleAbs(dst)
    cv.imshow(window_name, abs_dst)
    cv.waitKey(0)
    return 0
if __name__ == "__main__":
    main(sys.argv[1:])
```

OUTPUT:

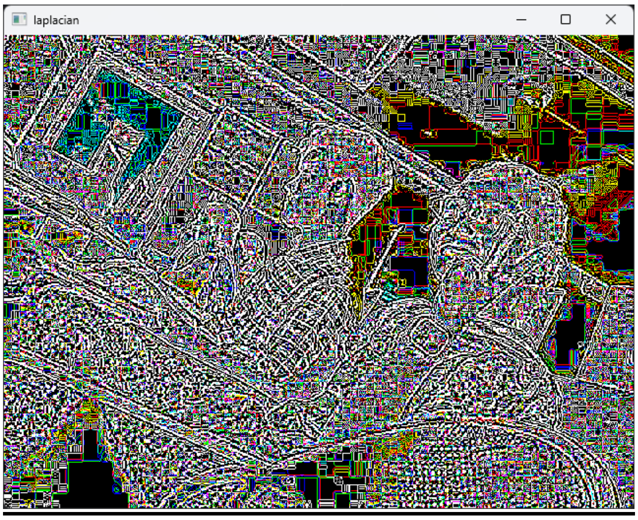


9)LAPLACE VIDEO:

PROGRAM:

```
import cv2
import numpy as np
# Capture livestream video content from camera 0
cap = cv2.VideoCapture(0)
while (1):
    # Take each frame
    _, frame = cap.read()
    # Convert to HSV for simpler calculations
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Calculation of Laplacian
    laplacian = cv2.Laplacian(frame, cv2.CV_64F)
    cv2.imshow('laplacian', laplacian)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
# release the frame
cap.release()
```

OUTPUT:



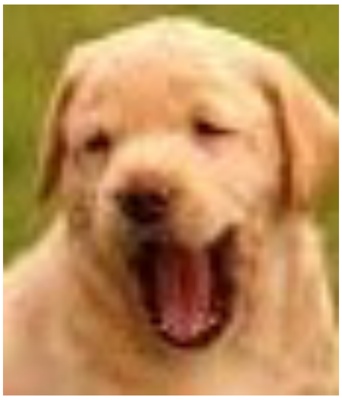
10)TEMPLATE IMAGE:

PROGRAM:

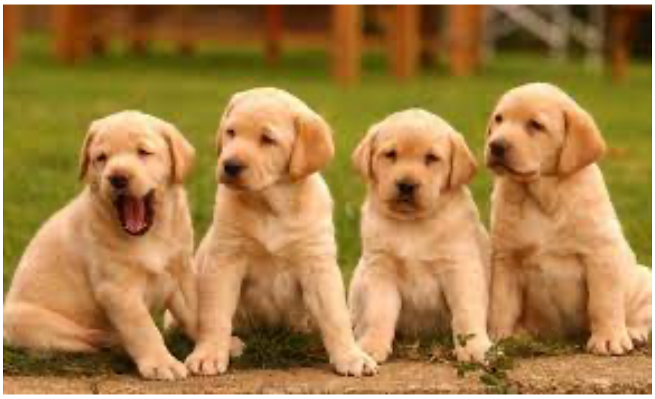
```
# Python program to illustrate
# template matching
import cv2
import numpy as np
# Read the main image
img_rgb = cv2.imread('DOG.jpg')
# Convert it to grayscale
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
# Read the template
template = cv2.imread('DOG1.jpg', 0)
# Store width and height of template in w and h
w, h = template.shape[::-1]
# Perform match operations.
res = cv2.matchTemplate(img_gray, template, cv2.TM_CCOEFF_NORMED)
# Specify a threshold
threshold = 0.8
# Store the coordinates of matched area in a numpy array
loc = np.where(res >= threshold)
# Draw a rectangle around the matched region.
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0, 255, 255), 2)
# Show the final image with the matched area.
cv2.imshow('Detected', img_rgb)
cv2.waitKey(0)
```

OUTPUT:

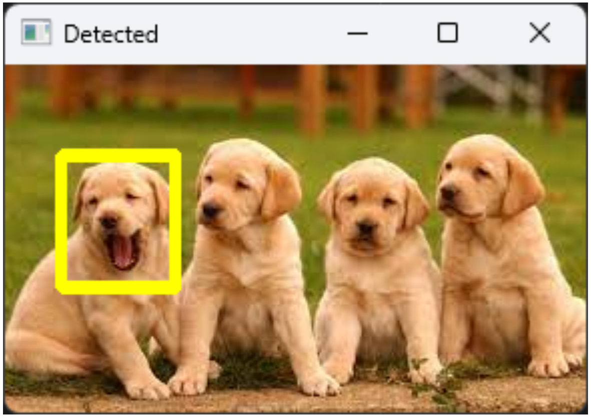
TEMPLATE



TEST IMAGE



DETECTED:



Department of RAE			
Criteria	Excellent (75% - 100%)	Good (50 – 75%)	Poor (<50%)
Preparation (30)			
Performance (30)			
Evaluation (20)			
Report (20)			
Sign:		Total (100)	

Result:

Thus the Edge Detection and Template Matching Techniques were learnt using OpenCV.

Post Lab Questions

1. **What is the difference between convolution and correlation?**

Convolution and correlation are two mathematical operations that are frequently used in signal processing, image processing, and various other fields. While they share many similarities, the key difference lies in the way they handle the filter (kernel) during the operation.

1. Convolution:

- In convolution, the filter is flipped before the operation. Mathematically, if f is the input signal and g is the filter, the convolution $f*g$ is given by: $(f*g)(t)=\sum a\sum bf(a)\cdot g(b)$
- The filter g is typically flipped both horizontally and vertically before sliding it over the input signal f.

2. Correlation:

- In correlation, the filter is not flipped. Mathematically, if f is the input signal and g is the filter, the correlation $f\star g$ is given by: $(f\star g)(t)=\sum a\sum bf(a)\cdot g(b)$
- Unlike convolution, there is no flipping of the filter in correlation.

2. **180 160 160 140 120**
110 110 120 140 120
110 140 120 120 140
120 160 160 170 170
170 120 110 140 110

For all the rows perform first order and second order derivative

PROGRAM:

```
import numpy as np
# Define the input matrix
matrix = np.array([
   [180, 160, 160, 140, 120],
   [110, 110, 120, 140, 120],
   [110, 140, 120, 120, 140],
   [120, 160, 160, 170, 170],
   [170, 120, 110, 140, 110]
], dtype=np.float64)
# Compute first-order derivatives
dx = np.gradient(matrix, axis=1)  # First-order derivative in the x-direction
dy = np.gradient(matrix, axis=0)  # First-order derivative in the y-direction
# Compute second-order derivatives
```

```
dxx = np.gradient(dx, axis=1)  # Second-order derivative in the x-direction
dyy = np.gradient(dy, axis=0)  # Second-order derivative in the y-direction
# Display the results
print("Original Matrix:")
print(matrix)
print("\nFirst Order Derivative (dx):")
print(dx)
print("\nFirst Order Derivative (dy):")
print(dy)
print("\nSecond Order Derivative (dxx):")
print(dxx)
print("\nSecond Order Derivative (dyy):")
print(dyy)
```

OUTPUT:

```
[[180, 160, 160, 140, 120],
 [110, 110, 120, 140, 120],
[110, 140, 120, 120, 140],
 [120, 160, 160, 170, 170],
[170, 120, 110, 140, 110]]

Original Matrix:
[[180. 160. 160. 140. 120.]
 [110. 110. 120. 140. 120.]
[110. 140. 120. 120. 140.]
 [120. 160. 160. 170. 170.]
 [170. 120. 110. 140. 110.]]

First Order Derivative (dx):
[[ -20.    0.    0.  -20.  -20.]
 [   0.   10.   10.   20.  -20.]
 [  30.  -20.    0.    0.   20.]
[  40.   20.    0.   10.    0.]
[ -50.   10.  -10.   30.  -30.]]

First Order Derivative (dy):
[[-70. -50. -40. -20. -40.]
 [  0.  30. -10.  20.   0.]
[ 30.  20.   0. -20.  20.]
[ 10.  40.   0.  10.   0.]
```

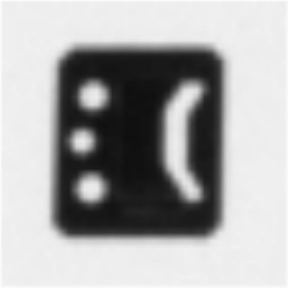
```
 [ 50. -50. -10.  30. -30.]]

Second Order Derivative (dxx):
[[ 20.   0.   0.   0.   0.]
 [ 10.   0.   0.  10. -40.]
[-50.  20.  20. -20.  20.]
 [ 10. -20. -20.  10. -10.]
 [ 60. -60.   0.  20. -60.]]

Second Order Derivative (dyy):
[[-70. -50. -40. -20. -40.]
 [ 30.  50. -20.  30. -20.]
 [ 20.   0.   0. -20.  20.]
 [ 20.  20. -10. -10.  10.]
 [ 40. -90. -10.  40. -50.]]
```

3. **Create a template and change the orientation of the template to different orientations and perform template matching for image of your choice.**

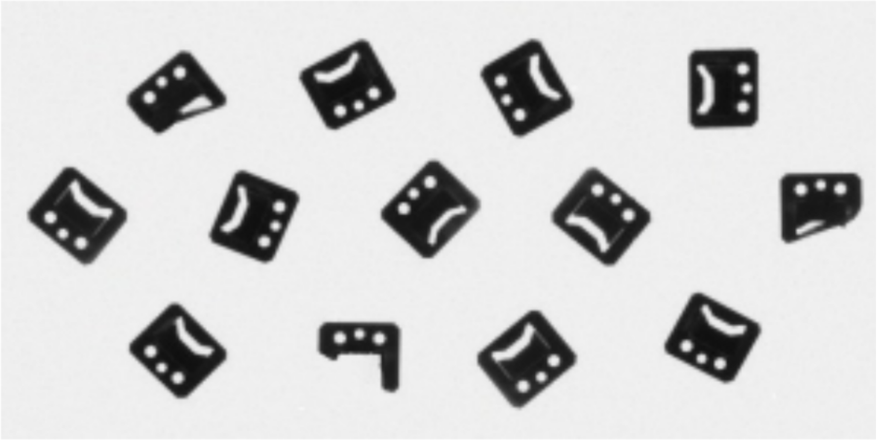
```
import cv2
import numpy as np
# Load the larger image
larger_image = cv2.imread('path/to/larger_image.jpg', cv2.IMREAD_GRAYSCALE)
# Load the template
template = cv2.imread('path/to/template.jpg', cv2.IMREAD_GRAYSCALE)
# Function to perform template matching
def perform_template_matching(image, templ):
   result = cv2.matchTemplate(image, templ, cv2.TM_CCOEFF_NORMED)
   min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
   return max_loc, max_val
# Perform template matching for the original template orientation
loc, val = perform_template_matching(larger_image, template)
print("Original Template Matching Result:")
print("Location:", loc)
print("Match Value:", val)
# Rotate the template (e.g., 90 degrees) and perform template matching
rotated_template = cv2.rotate(template, cv2.ROTATE_90_CLOCKWISE)
loc_rotated, val_rotated = perform_template_matching(larger_image, rotated_template)
print("\nTemplate Matching Result (Rotated 90 degrees):")
print("Location:", loc_rotated)
print("Match Value:", val_rotated)
```

OUTPUT:

TEMPLATE IMAGE



INPUT IMAGE:



RESULTS: