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

Programs:

Edge Detection:

#Sobel

#Prewit

#Laplacian

#Canny Edge Detection

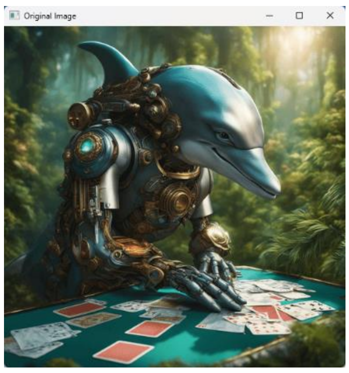
Template Matching (Both for Image and Video)

a.Single Template

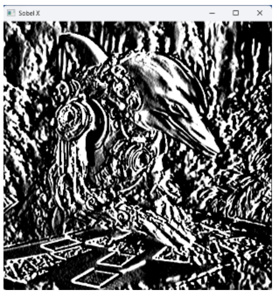
b.Multiple template

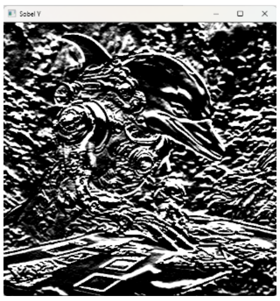
c.Video streaming

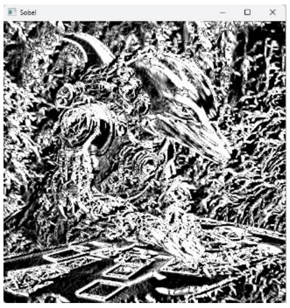
Input: (Original Image)



Output: (Sobel)







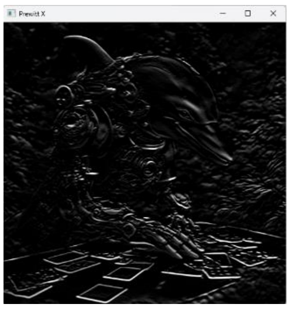
1. **Edge detection**

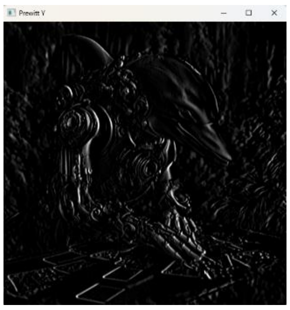
a) **Sobel**

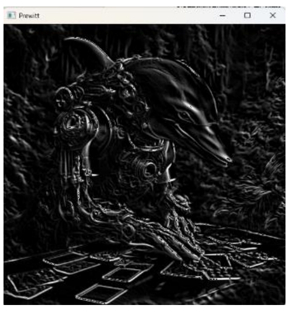
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

input_image = cv.imread(dolphin.jpg')  # Replace with your image path
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 = cv.cvtColor(input_image,cv.COLOR_BGR2GRAY)
    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
# Apply Sobel operator
sobel_x, sobel_y, sobel_mag = sobel_operator(input_image)
# Display the results
plt.figure(figsize=(10, 5))
plt.subplot(1,4,1)
plt.title('Original')
plt.imshow(input_image,cmap ='gray')
plt.subplot(1, 4, 2)
plt.title('Sobel X')
plt.imshow(sobel_x, cmap='gray')
plt.axis('off')
plt.subplot(1, 4, 3)
plt.title('Sobel Y')
plt.imshow(sobel_y, cmap='gray')
plt.axis('off')
plt.subplot(1, 4, 4)
plt.title('Sobel Magnitude')
plt.imshow(sobel_mag, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()
```

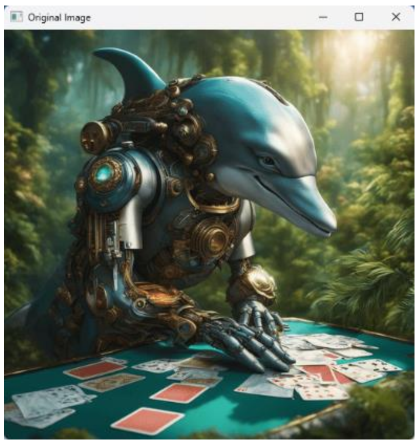
Output: (Prewitt)

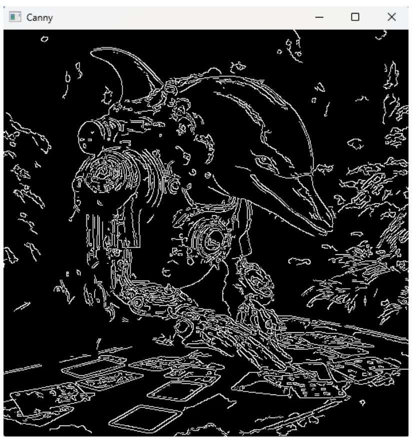






Output:(Canny)





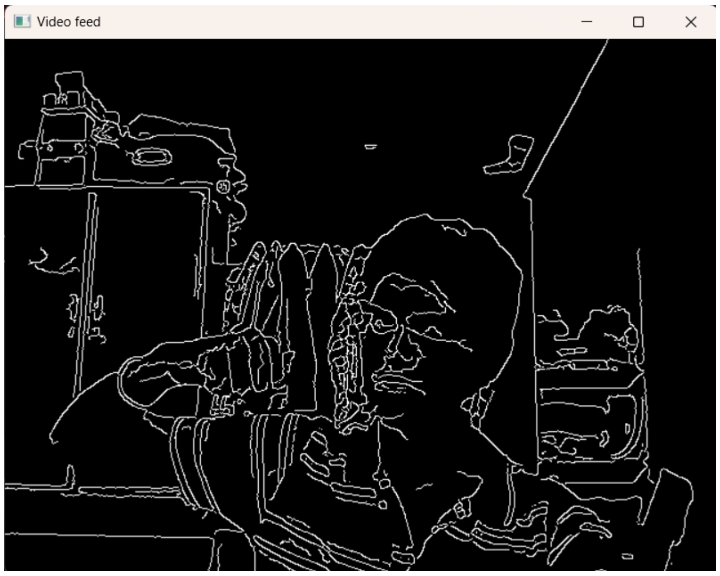
b) Prewitt

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
img = cv2.imread('dolphin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_gaussian = cv2.GaussianBlur(gray,(3,3),0)
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)
plt.figure(figsize=(10, 5))
plt.subplot(1,3,1)
plt.title('Original')
plt.imshow(img,cmap ='gray')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.title("Prewitt X")
plt.imshow(img_prewittx, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.title("Prewitt Y")
plt.imshow(img_prewitty, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()
```

c) Canny

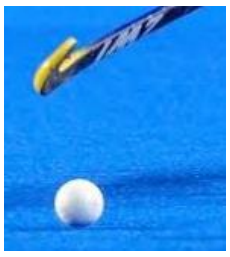
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the image in grayscale
img = cv2.imread(dolphin.jpg', cv2.IMREAD_GRAYSCALE)
# Apply Canny edge detector
edges = cv2.Canny(img, 50, 150)
# Display the results
plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis("off")
plt.subplot(1,2,2)
plt.imshow(edges, cmap='gray')
plt.title('Canny Edges')
plt.axis("off")
plt.show()
```

Output: (Edge detection in video)

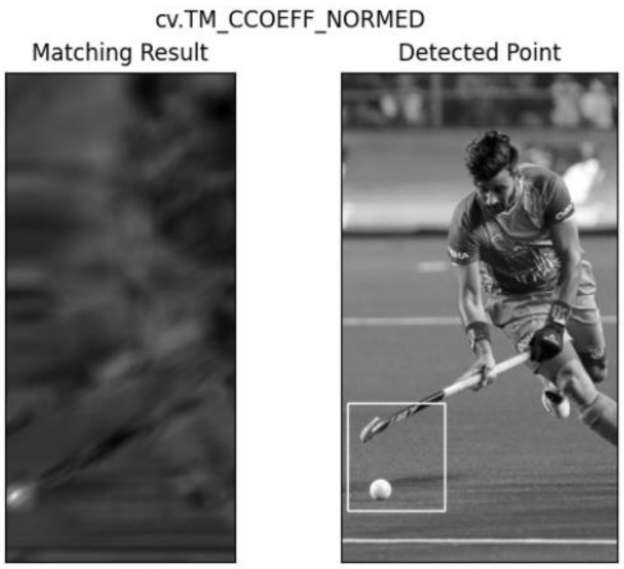


Output: (Single Template)

Template:



Result:



d) Edge detection in Video

```
import cv2
cap = cv2.VideoCapture(0)
while True:
   ret, img = cap.read()
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   blur = cv2.GaussianBlur(gray, (5, 5), 0)
   canny = cv2.Canny(blur, 10, 70)
   ret, mask = cv2.threshold(canny, 70, 255, cv2.THRESH_BINARY)
   cv2.imshow('Video feed', mask)
   if cv2.waitKey(1) == 13:
      break
cap.release()
cv2.destroyAllWindows()
```

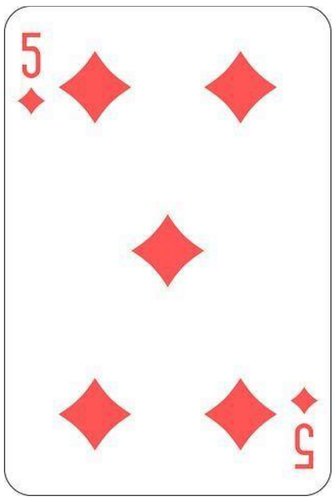
2. Template matching

a) Single

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread(hockey.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
img2 = img.copy()
template = cv.imread('temp.png', cv.IMREAD_GRAYSCALE)
assert template is not None, "file could not be read, check with os.path.exists()"
w, h = template.shape[::-1]
# All the 6 methods for comparison in a list
methods = ['cv.TM_CCOEFF_NORMED']
for meth in methods:
   img = img2.copy()
   method = eval(meth)
   # Apply template Matching
   res = cv.matchTemplate(img,template,method)
   min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
   # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
   if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
      top_left = min_loc
   else:
      top_left = max_loc
   bottom_right = (top_left[0] + w, top_left[1] + h)
   cv.rectangle(img,top_left, bottom_right, 255, 2)
   plt.subplot(1,2,1)
   plt.imshow(res,cmap = 'gray')
   plt.title('Matching Result')
   plt.xticks([])
   plt.yticks([])
   plt.subplot(1,2,2)
   plt.imshow(img,cmap = 'gray')
   plt.title('Detected Point')
   plt.xticks([])
   plt.yticks([])
   plt.suptitle(meth)
   plt.show()
```

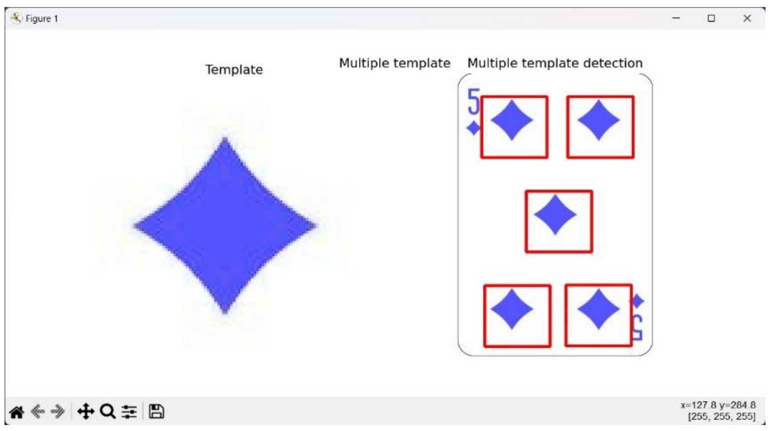
Output: (Multiple Template)

Original Image:



Template:

Result:



b) Multiple template

```
import cv2
import numpy as np
from imutils.object_detection import non_max_suppression
import matplotlib.pyplot as plt
# Reading the image and the template
img = cv2.imread('card.jpg')
temp = cv2.imread('temp.png')
# save the image dimensions
W, H = temp.shape[:2]
# Define a minimum threshold
thresh = 0.4
# Converting them to grayscale
img_gray = cv2.cvtColor(img,
                    cv2.COLOR_BGR2GRAY)
temp_gray = cv2.cvtColor(temp,
                     cv2.COLOR_BGR2GRAY)
# Passing the image to matchTemplate method
match = cv2.matchTemplate(
   image=img_gray, templ=temp_gray,
   method=cv2.TM_CCOEFF_NORMED)
# Select rectangles with
# confidence greater than threshold
(y_points, x_points) = np.where(match >= thresh)
# initialize our list of rectangles
boxes = list()
# loop over the starting (x, y)-coordinates again
for (x, y) in zip(x_points, y_points):
   # update our list of rectangles
   boxes.append((x, y, x + W, y + H))
# apply non-maxima suppression to the rectangles
# this will create a single bounding box
boxes = non_max_suppression(np.array(boxes))
# loop over the final bounding boxes
for (x1, y1, x2, y2) in boxes:
   # draw the bounding box on the image
   cv2.rectangle(img, (x1, y1), (x2, y2),
              (255, 0, 0), 3)
# Show the template and the final output
plt.figure(figsize=(10,5))
plt.title("Multiple template")
plt.axis("off")
plt.subplot(1,2,1)
plt.title("Template")
plt.imshow(temp)
plt.axis("off")
plt.subplot(1,2,2)
plt.title("Multiple template detection")
plt.imshow(img)
plt.axis("off")
plt.show()
```

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.

<u>Post Lab Questions</u>

1. What is the difference between convolution and 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
3. Create a template and change the orientation of the template to different orientations and perform template matching for image of your choice

<u>Answers:</u>

1. Convolution and correlation are both mathematical operations that combine two functions to produce a third function.
   Convolution involves flipping one of the functions (usually the impulse response) and then sliding it along the other function, calculating the product at each position and summing the results. It is used to filter a signal
   Correlation, on the other hand, does not flip the function and instead slides it along the other function, calculating the product at each position and summing the results. It is used to find similarities between two signals.

2. First order derivative:

   Formula: $$\frac{da_{ij}}{dx} \approx \frac{a_{ij}(x+\epsilon)-a_{ij}(x)}{\epsilon}$$

   Since each element is constant, First order derivative will be zero

   Second order derivative:

   Formula: $$\frac{d^2a_{ij}}{dx^2} \approx \frac{a_{ij}(x+\epsilon)-2a_{ij}(x)+a_{ij}(x-\epsilon)}{\epsilon^2}$$

   Since First order derivative is zero, Second order derivative will also be zero.

3.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('card.jpg',0)
img2 = img.copy()
template = cv.imread('temp.png',0)
w, h = template.shape[::-1]
# All the 6 methods for comparison in a list
```

```
methods = [ 'cv.TM_CCOEFF_NORMED']
for meth in methods:
    img = img2.copy()
    method = eval(meth)
    # Apply template Matching
    res = cv.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
    # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
    if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv.rectangle(img,top_left, bottom_right, 255, 2)
    plt.subplot(121),plt.imshow(res,cmap = 'gray')
    plt.title('Matching Result'), plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(img,cmap = 'gray')
    plt.title('Detected Point'), plt.xticks([]), plt.yticks([])
    plt.suptitle(meth)
    plt.show()
```