**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**

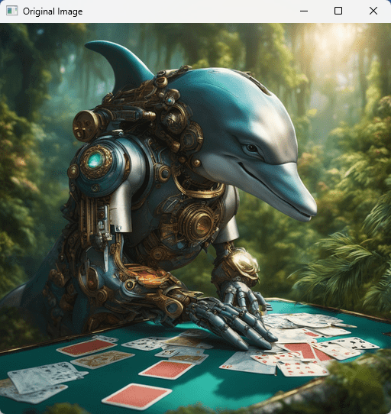
**Template Matching (Both for Image and Video)**

**a.Single Template**

**b.Multiple template**

**c.Video streaming**

**Input: (Original Image)**



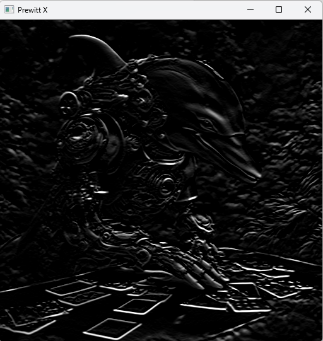
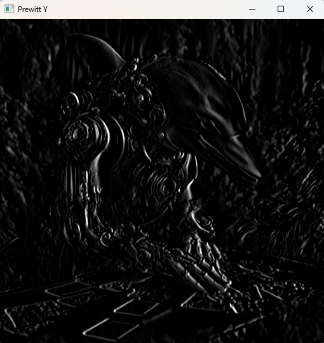
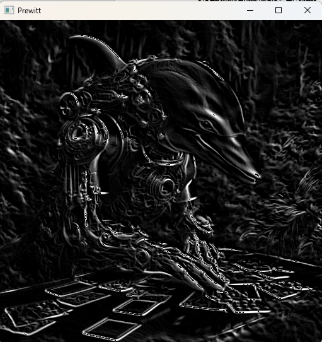
**Output: (Sobel)**



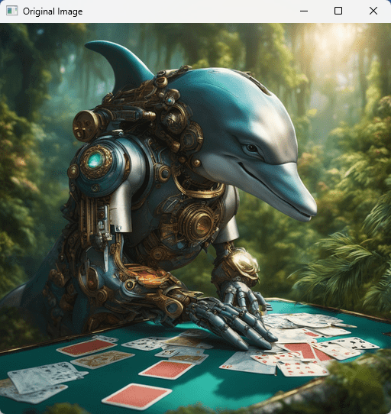
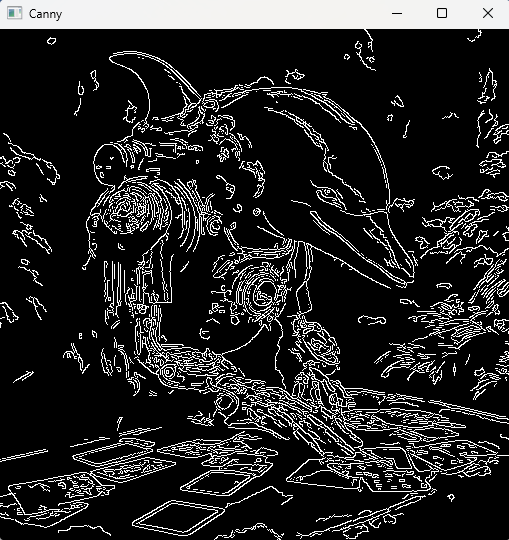
1. **Edge detection**
2. **Sobel**

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)**



1. **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()

1. **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)**

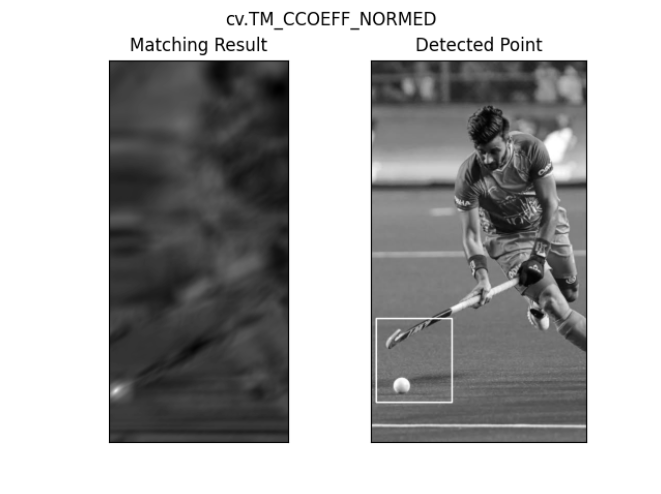
****

**Output: (Single Template)**

**Template:**



**Result:**

****

1. **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()

1. **Template matching**
2. **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([])

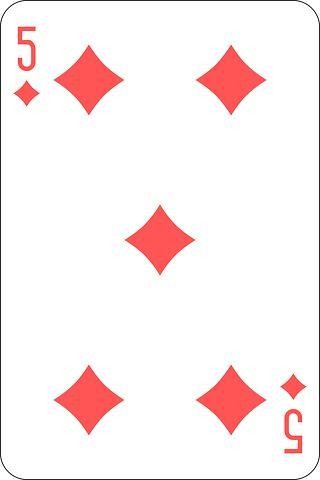
plt.yticks([])

plt.suptitle(meth)

plt.show()

**Output: (Multiple Template)**

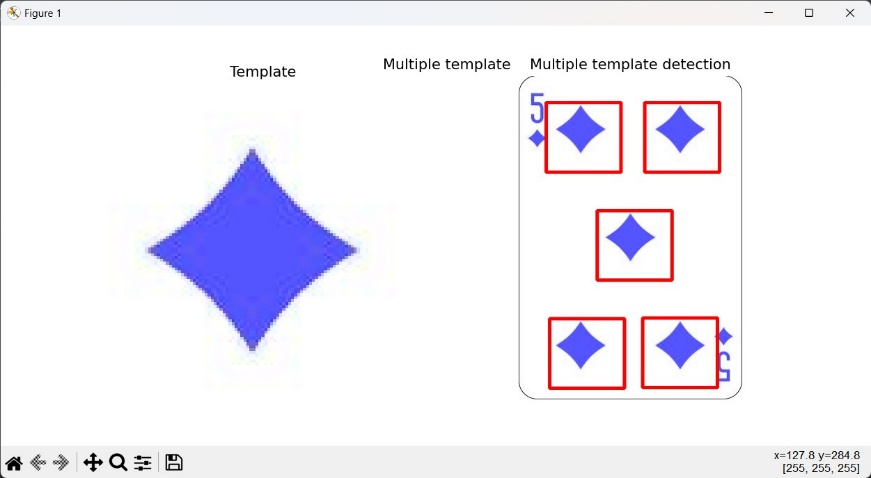
**Original Image:**



**Template:**

****

**Result:**



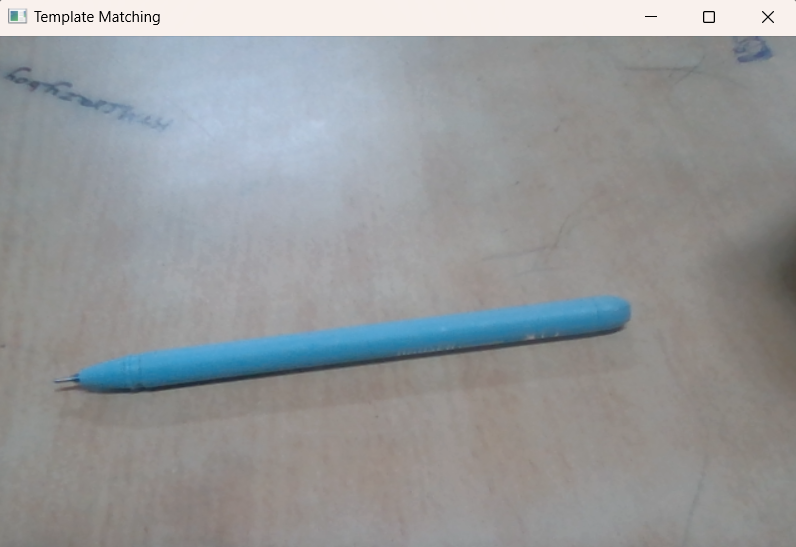
1. **Multiple template**

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()

**Input Image:**



**Output: (Detected Image)**

****

1. **Template Matching Video:**

import cv2

import numpy as np

# Load the video

video\_capture = cv2.VideoCapture(0)

# Load the template image

template = cv2.imread('D:\pen.png', cv2.IMREAD\_COLOR)

# Get the dimensions of the template

template\_height, template\_width, \_ = template.shape

method = cv2.TM\_CCOEFF\_NORMED

while True:

# Read the current frame from the video

ret, frame = video\_capture.read()

if not ret:

break

# Perform template matching

result = cv2.matchTemplate(frame, template, method)

# Get the location of the best match

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

# Draw a rectangle around the matched region

top\_left = max\_loc

bottom\_right = (top\_left[0] + template\_width, top\_left[1] + template\_height)

cv2.rectangle(frame, top\_left, bottom\_right, (0, 255, 0), 2)

# Display the frame with the rectangle

cv2.imshow('Template Matching', frame)

# Break the loop if the 'q' key is pressed

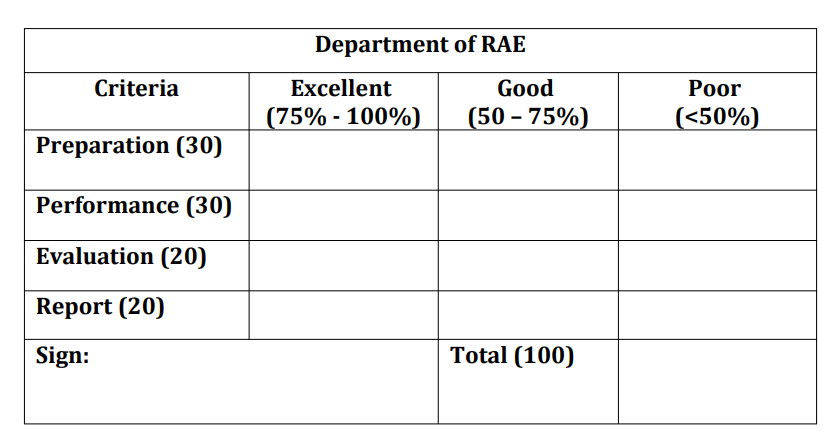
if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the video capture object and close all windows

video\_capture.release()

cv2.destroyAllWindows()



**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
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

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

**Answers:**

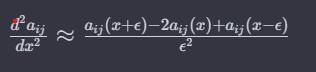
1. Convolution and correlation are both mathematical operations that combine two functions to produce a third function.

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.

1. First order derivative:

Formula:

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

Formula:

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

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()