# CSE 573: Introduction to Computer Vision and Image Processing

Instructor: Dr. Junsong Yuan
*University at Buffalo*

# Project 1 Report

Submitted By:
**Shreyas Narasimha** (UB Person Number: 50289736)

# Task 1 – Edge Detection

**Objective:** To manually write a Sobel operator to detect horizontal and vertical edges in python.
**Developed Code:** (Using Python 3)

```
#Name: Shreyas Narasimha
#UBID: sn58
#UB Person Number: 50289736

import cv2
import numpy as np
import math

w = 0
h = 0

#dimensions are known by using img.shape()
img = cv2.imread("task1.png", 0)
for i in img:
    h+=1

for i in img[0]:
    w+=1

filtered = []
nlist = [[0 for j in range(w)] for i in range(h)]
xtemplist = [[0 for j in range(w)] for i in range(h)]
ytemplist = [[0 for j in range(w)] for i in range(h)]
maggi = [[0 for j in range(w)] for i in range(h)]
oreo =  [[0 for j in range(w)] for i in range(h)]

def display(name,img):
    cv2.namedWindow(name, cv2.WINDOW_NORMAL)
    cv2.imshow(name, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


def norm(edge,w,h):
    m = 0.0
    for i in range(h):
        for j in range(w):
            if(abs(edge[i][j]) > m):
                m=abs(edge[i][j])
    edge = [[abs(item)/m for item in subl] for subl in edge]
    return np.asarray(edge, dtype = np.float32)



for i in range(h):
    for j in range(w):
        nlist[i][j] = img[i][j]

edge_x = [[0 for j in range(w)] for i in range(h)]
edge_y = [[0 for j in range(w)] for i in range(h)]

# Computing vertical edges
#We manually write the equivalent of te above
#Aim is to write the code above manually without using OpenCV for vertical edges
for i in range(1,h-1):
    for j in range(1,w-1):
        xtemplist[i][j] = nlist[i-1][j+1] + 2*nlist[i][j+1] + nlist[i+1][j+1] -
nlist[i+1][j-1] - 2*nlist[i][j-1] - nlist[i+1][j-1]
```

```
        ytemplist[i][j] = nlist[i+1][j-1] + 2*nlist[i+1][j] + nlist[i+1][j+1] -
nlist[i-1][j-1] - 2*nlist[i-1][j] - nlist[i-1][j+1]
        #maggi[i][j] = math.sqrt(xtemplist[i][j]*xtemplist[i][j] +
ytemplist[i][j]*ytemplist[i][j])
        #oreo[i][j] = math.atan2(ytemplist[i][j] , xtemplist[i][j] + 1e-3)
#Computations
edge_x = np.asarray(xtemplist, dtype = np.float32) #Vertical response
edge_y = np.asarray(ytemplist, dtype = np.float32) #Horizontal response
#Display0 - original image
cv2.imwrite('edge_x.jpg',edge_x)
cv2.imwrite('norm_x.jpg',norm(edge_x,w,h))
cv2.imwrite('edge_y.jpg',edge_y)
cv2.imwrite('norm_y.jpg',norm(edge_y,w,h))

display('original',img)
display('edge_x',edge_x)
display('norm_x',norm(edge_x,w,h))
display('edge_y',edge_y)
display('norm_y',norm(edge_y,w,h))
```

**Input:**


**Fig.1** Original Image – 900x600

**Output:**

The Sobel operator is applied on the original image after converting it to its grayscale equivalent.
The Sobel operator used is a 3x3 matrix kernel:-

[[-1 0 1],
[-2 0 2], To detect vertical edges - Gx
[-1 0 1]]

[[1 2 1],
[0 0 0], To detect horizontal edges - Gy
[-1 -2 -1]]

The matrix is flipped and convolved with each pixel to detect the edges.
The displayed output is a follows:-

**Fig.1.1 -** shows all the edges detected along the y axis, i.e the horizontal edges.

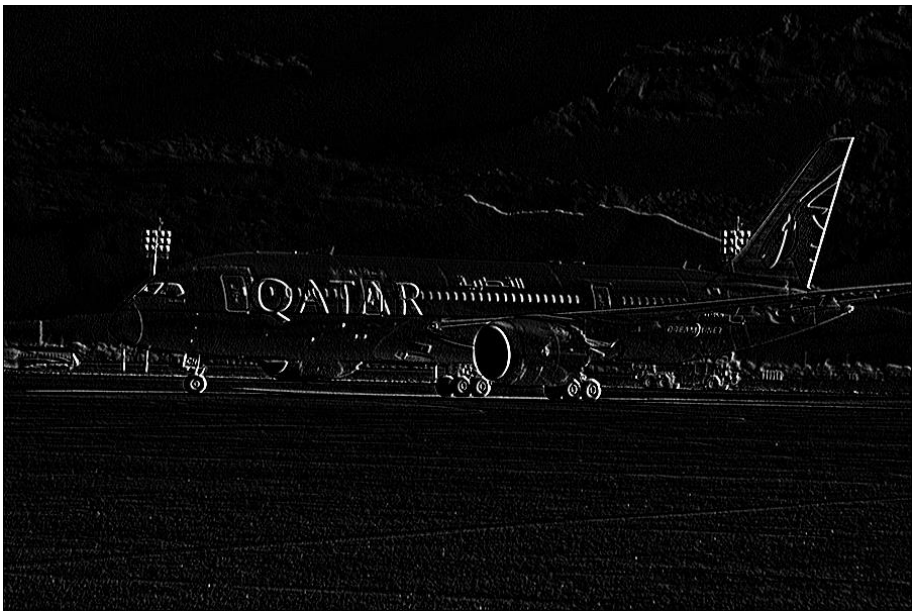

**Fig.1.2 -** shows all the edges detected along the y axis, i.e the horizontal edges.

# Task 2 – Keypoint Detection

**Objective:** To detect keypoints in a given image using Scale Invariant Feature Transform(SIFT).

**Steps:**

1) Successively blur the given image in grayscale by convolution with a 7x7 gaussian kernel to generate 5 images, each using different sigma values.
2) The above constitutes 1 octave. Generate 3 more octaves using the scaled down version of the original image.
3) Using the generated images with Gaussian blurs, generate 4 laplacians (DoG) for each octave.
4) Display pixels in the DoG which are the minima or maxima and then display the combined keypoints.

Gaussian kernel is calculated using:

$$G(x) = (1/2*PI*(sigma^2) )*(exp(-(x^2 + y^2)/2*sigma^2))$$

| Octave | Sigma 1 | Sigma 2 | Sigma 3 | Sigma 4 | Sigma 5 |
|---|---|---|---|---|---|
| 1 | 1/√2 | 1 | 1√2 | 2 | 2√2 |
| 2 | √2 | 2 | 2√2 | 4 | 4√2 |
| 3 | 2√2 | 4 | 4√2 | 8 | 8√2 |
| 4 | 4√2 | 8 | 8√2 | 16 | 16√2 |

**Table 1:** Sigma Values used for each octave

**Developed code:** (Using python 3)

```
#Name: Shreyas Narasimha
#UBID: sn58
#UB Person Number: 50289736

import cv2
import numpy as np
import math
import time
s = time.time()
#declarations
#convert numpy array to a nested list

h = 0
w = 0
#dimensions are known by using img.shape()
img = cv2.imread("task2.jpg", 0)
for i in img:
    h+=1

for i in img[0]:
    w+=1

nlist = [[0 for j in range(w+6)] for i in range(h+6)]
for i in range(h):
    for j in range(w):
        nlist[i+3][j+3] = img[i][j]


#code to create a list
```

```
def padlist(img,w,h):
    w = int(w/2)
    h = int(h/2)
    temp = [[0 for i in range(w+6)] for j in range(h+6)]
    for i in range(h):
        for j in range(w):
            temp[i+3][j+3] = img[i][j]
    return temp
#display function
def display(name,img):
    cv2.namedWindow(name, cv2.WINDOW_NORMAL)
    cv2.imshow(name, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def downsize(img,h,w,octno):
    hn = int(h/2)
    wn = int(w/2)
    print(hn,wn)
    temp =[[0 for j in range(wn)] for i in range(hn)]
    for i in range(hn):
        for j in range(wn):
            temp[i][j] = img[i*octno][j*octno]
    return np.asarray(temp, dtype = np.uint8)

#function to calculate g(x,y)
def transform(a,sigma,w,h):
    w = int(w)
    h = int(h)
    t = 0.0
    xtemplist = [[0 for j in range(w)] for i in range(h)]
    b = [[0 for j in range(7)] for i in range(7)]
    for i in range(-3,4):
        for j in range(-3,4):
            b[i+3][j+3] = (1/(2*math.pi*(sigma**2)))*(math.exp(-(i**2 +
j**2)/(2*sigma**2)))
            t+=b[i+3][j+3]
    b = [[item/t for item in subl]for subl in b]
    for i in range(h):
        for j in range(w):
            xtemplist[i][j] = (a[i][j] * b[0][0]) + (a[i][j+1] * b[0][1]) +
(a[i][j+2] * b[0][2]) + (a[i][j+3] * b[0][3]) + (a[i][j+4] * b[0][4]) + (a[i][j+5]
* b[0][5]) + (a[i][j+6] * b[0][6]) + (a[i+1][j] * b[1][0]) + (a[i+1][j+1] *
b[1][1]) + (a[i+1][j+2] * b[1][2]) + (a[i+1][j+3] * b[1][3]) + (a[i+1][j+4] *
b[1][4]) + (a[i+1][j+5] * b[1][5]) + (a[i+1][j+6] * b[1][6]) + (a[i+2][j] *
b[2][0]) + (a[i+2][j+1] * b[2][1]) + (a[i+2][j+2] * b[2][2]) + (a[i+2][j+3] *
b[2][3]) + (a[i+2][j+4] * b[2][4]) + (a[i+2][j+5] * b[2][5]) + (a[i+2][j+6] *
b[2][6]) + (a[i+3][j] * b[3][0]) + (a[i+3][j+1] * b[3][1]) + (a[i+3][j+2] *
b[3][2]) + (a[i+3][j+3] * b[3][3]) + (a[i+3][j+4] * b[3][4]) + (a[i+3][j+5] *
b[3][5]) + (a[i+3][j+6] * b[3][6]) + (a[i+4][j] * b[4][0]) + (a[i+4][j+1] *
b[4][1]) + (a[i+4][j+2] * b[4][2]) + (a[i+4][j+3] * b[4][3]) + (a[i+4][j+4] *
b[4][4]) + (a[i+4][j+5] * b[4][5]) + (a[i+4][j+6] * b[4][6]) + (a[i+5][j] *
b[5][0]) + (a[i+5][j+1] * b[5][1]) + (a[i+5][j+2] * b[5][2]) + (a[i+5][j+3] *
b[5][3]) + (a[i+5][j+4] * b[5][4]) + (a[i+5][j+5] * b[5][5]) + (a[i+5][j+6] *
b[5][6]) + (a[i+6][j] * b[6][0]) + (a[i+6][j+1] * b[6][1]) + (a[i+6][j+2] *
b[6][2]) + (a[i+6][j+3] * b[6][3]) + (a[i+6][j+4] * b[6][4]) + (a[i+6][j+5] *
b[6][5]) + (a[i+6][j+6] * b[6][6])
    return np.asarray(xtemplist, dtype = np.float32)
    #return xtemplist
```

```
def laplace(img1,img2,w,h):
    w = int(w)
    h = int(h)
    temp3 = [[0 for i in range(w)] for j in range(h)]
    for i in range(h):
        for j in range(w):
            temp3[i][j] = (img1[i][j] - img2[i][j])
    temp3 = np.asarray(temp3, dtype = np.float32)
    return temp3

def keypoints(l1,l2,l3,l4,w,h,n):
    w = int(w)
    h = int(h)
    temp1 = [[0 for i in range(w)] for j in range(h)]
    temp2 = [[0 for i in range(w)] for j in range(h)]
    for i in range(1,h-1):
        for j in range(1,w-1):
            if((l2[i][j] > max(l1[i-1][j-1],l1[i-1][j],l1[i-1][j+1],l1[i][j-
1],l1[i][j],l1[i][j+1],l1[i+1][j-1],l1[i+1][j],l1[i+1][j+1],l2[i-1][j-1],l2[i-
1][j],l2[i-1][j+1],l2[i][j-1],l2[i][j+1],l2[i+1][j-1],l2[i+1][j],l2[i+1][j+1],l3[i-
1][j-1],l3[i-1][j],l3[i-1][j+1],l3[i][j-1],l3[i][j],l3[i][j+1],l3[i+1][j-
1],l3[i+1][j],l3[i+1][j+1])) or (l2[i][j] < min(l1[i-1][j-1],l1[i-1][j],l1[i-
1][j+1],l1[i][j-1],l1[i][j],l1[i][j+1],l1[i+1][j-1],l1[i+1][j],l1[i+1][j+1],l2[i-
1][j-1],l2[i-1][j],l2[i-1][j+1],l2[i][j-1],l2[i][j+1],l2[i+1][j-
1],l2[i+1][j],l2[i+1][j+1],l3[i-1][j-1],l3[i-1][j],l3[i-1][j+1],l3[i][j-
1],l3[i][j],l3[i][j+1],l3[i+1][j-1],l3[i+1][j],l3[i+1][j+1] ))):
                if(l2[i][j]>n):
                    temp1[i][j] = 255
                else:
                    temp1[i][j] = 0
    for i in range(1,h-1):
        for j in range(1,w-1):
            if((l3[i][j] > max(l2[i-1][j-1],l2[i-1][j],l2[i-1][j+1],l2[i][j-
1],l2[i][j],l2[i][j+1],l2[i+1][j-1],l2[i+1][j],l2[i+1][j+1],l3[i-1][j-1],l3[i-
1][j],l3[i-1][j+1],l3[i][j-1],l3[i][j+1],l3[i+1][j-1],l3[i+1][j],l3[i+1][j+1],l4[i-
1][j-1],l4[i-1][j],l4[i-1][j+1],l4[i][j-1],l4[i][j],l4[i][j+1],l4[i+1][j-
1],l4[i+1][j],l4[i+1][j+1])) or (l3[i][j] < min(l2[i-1][j-1],l2[i-1][j],l2[i-
1][j+1],l2[i][j-1],l2[i][j],l2[i][j+1],l2[i+1][j-1],l2[i+1][j],l2[i+1][j+1],l3[i-
1][j-1],l3[i-1][j],l3[i-1][j+1],l3[i][j-1],l3[i][j+1],l3[i+1][j-
1],l3[i+1][j],l3[i+1][j+1],l4[i-1][j-1],l4[i-1][j],l4[i-1][j+1],l4[i][j-
1],l4[i][j],l4[i][j+1],l4[i+1][j-1],l4[i+1][j],l4[i+1][j+1] ))):
                if(l3[i][j]>n):
                    temp2[i][j] = 255
                else:
                    temp2[i][j] = 0
    return (np.asarray(temp1, dtype=np.uint8), np.asarray(temp2, dtype=np.uint8))

def overlay(img,key1,key2,w,h):
    w = int(w)
    h = int(h)
    temp = [[0 for i in range(w)] for j in range(h)]
    for i in range(h):
        for j in range(w):
            if((key1[i][j] == 255) or (key2[i][j] == 255) ):
                temp[i][j] = 255
            else:
                temp[i][j] = img[i][j]
    return np.asarray(temp, dtype = np.uint8)

def finalkey(key,b,c,d,w,h):
```

```
    w1 = int(w/2)
    h1 = int(h/2)
    w2 = int(w1/2)
    h2 = int(h1/2)
    w3 = int(w2/2)
    h3 = int(h2/2)
    for i in range(h1):
        for j in range(w1):
            if(b[i][j] == 255):
                key[i*2][j*2] = 255
    for i in range(h2):
        for j in range(w2):
            if(b[i][j] == 255):
                key[i*4][j*4] = 255
    for i in range(h3):
        for j in range(w3):
            if(b[i][j] == 255):
                key[i*8][j*8] = 255
    return np.asarray(key, dtype= np.uint8)



#1st oct
a1 = transform(nlist,(1/math.sqrt(2)),w,h)
a2 = transform(nlist,1,w,h)
a3 = transform(nlist,(math.sqrt(2)),w,h)
a4 = transform(nlist,2,w,h)
a5 = transform(nlist,(2*math.sqrt(2)),w,h)
al1 = laplace(a1,a2,w,h)
al2 = laplace(a2,a3,w,h)
al3 = laplace(a3,a4,w,h)
al4 = laplace(a4,a5,w,h)
(ak1,ak2)  = keypoints(al1,al2,al3,al4,w,h,5)
af = overlay(img, ak1,ak2,w,h)

#2nd Oct
b   = downsize(img,h,w,2)
bp  = padlist(b,w,h)
b1  = transform(bp,math.sqrt(2),w/2,h/2)
b2  = transform(bp,2,w/2,h/2)
b3  = transform(bp,(2*math.sqrt(2)),w/2,h/2)
b4  = transform(bp,4,w/2,h/2)
b5  = transform(bp,(4*math.sqrt(2)),w/2,h/2)
bl1 = laplace(b1,b2,w/2,h/2)
bl2 = laplace(b2,b3,w/2,h/2)
bl3 = laplace(b3,b4,w/2,h/2)
bl4 = laplace(b4,b5,w/2,h/2)
(bk1,bk2)  = keypoints(bl1,bl2,bl3,bl4,w/2,h/2,0.2)
bf = overlay(b, bk1,bk2,w/2,h/2)

#3rd oct

c   = downsize(img,h/2,w/2,4)
cp  = padlist(c,w/2,h/2)
c1  = transform(cp,2*math.sqrt(2),w/4,h/4)
c2  = transform(cp,4,w/4,h/4)
c3  = transform(cp,4*math.sqrt(2),w/4,h/4)
c4  = transform(cp,8,w/4,h/4)
c5  = transform(cp,8*math.sqrt(2),w/4,h/4)
```

```
cl1 = laplace(c1,c2,w/4,h/4)
cl2 = laplace(c2,c3,w/4,h/4)
cl3 = laplace(c3,c4,w/4,h/4)
cl4 = laplace(c4,c5,w/4,h/4)
(ck1,ck2)  = keypoints(cl1,cl2,cl3,cl4,w/4,h/4,0)
cf = overlay(c, ck1,ck2,w/4,h/4)


#4th Octave
d   = downsize(img,h/4,w/4,8)
dp  = padlist(d,w/4,h/4)
d1  = transform(dp,4*math.sqrt(2),w/8,h/8)
d2  = transform(dp,8,w/8,h/8)
d3  = transform(dp,8*math.sqrt(2),w/8,h/8)
d4  = transform(dp,16,w/8,h/8)
d5  = transform(dp,16*math.sqrt(2),w/8,h/8)
dl1 = laplace(d1,d2,w/8,h/8)
dl2 = laplace(d2,d3,w/8,h/8)
dl3 = laplace(d3,d4,w/8,h/8)
dl4 = laplace(d4,d5,w/8,h/8)
(dk1,dk2)  = keypoints(dl1,dl2,dl3,dl4,w/8,h/8,0)
df = overlay(d, dk1,dk2,w/8,h/8)

#Calculating final keypoints by overlaying the downsized keypoints on the original
image
keypointImage = finalkey(af,bf,cf,df,w,h)
count = 1
#to print left most point positions
for i in range(h):
    for j in range(10):
            if(keypointImage[i][j] == 255):
                print(i,j)
                count = count+1
t = time.time()
print(t-s)
#Writing data for report

display('final keypoints',keypointImage)
```

**Input:**

**Fig.2 –** given image sized 750x458

**Output:**

Images of the second and third octave after convolution with the Gaussian matrix:-

**Size** – 375x229

**Fig.2.1 –** Octave 2 Image(Not blurred) **Size** – 375x229

**Fig.2.2 –** Octave 2 Image 1 **Size** – 375x229

**Fig.2.3 –** Octave 2 Image 2 **Size** – 375x229

**Fig.2.4 –** Octave 2 Image 3 **Size** – 375x229

**Fig.2.5 –** Octave 2 Image 4 **Size** – 375x229

**Fig.2.6–** Octave 2 Image 4 **Size** – 375x229

**Fig.2.7–** Octave 3 Image(Not blurred) **Size** – 187x114

**Fig.2.8–** Octave 3 Image 1**Size** – 187x114

**Fig.2.9–** Octave 3 Image 2 **Size** – 187x114

**Fig.2.10–** Octave 3 Image 3 **Size** – 187x114

**Fig.2.11–** Octave 3 Image 4 **Size** – 187x114

**Fig.2.12–** Octave 3 Image 5 **Size** – 187x114

Difference of Gaussian(DoG) of the 2nd and 3rd Octave:-

2nd Octave:-

**Fig.2.13.a** **Fig.2.13.b**

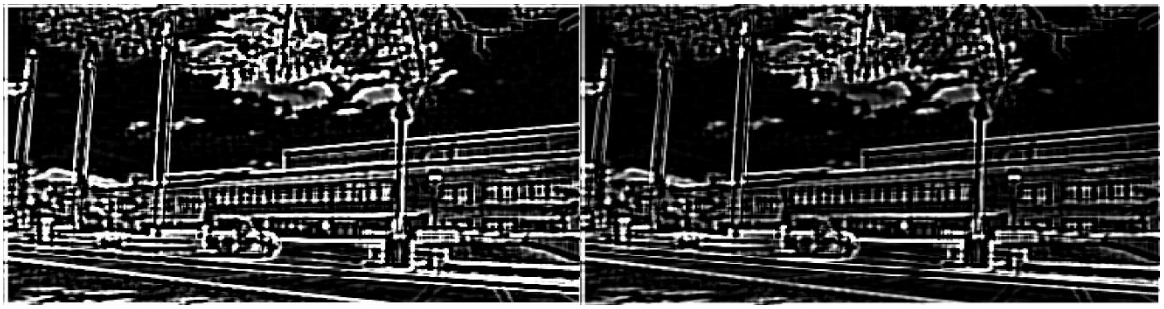

**Fig.2.13.c** **Fig.2.13.d**

**Fig.2.13** – DoG Images of Octave 2

3rd Octave:-

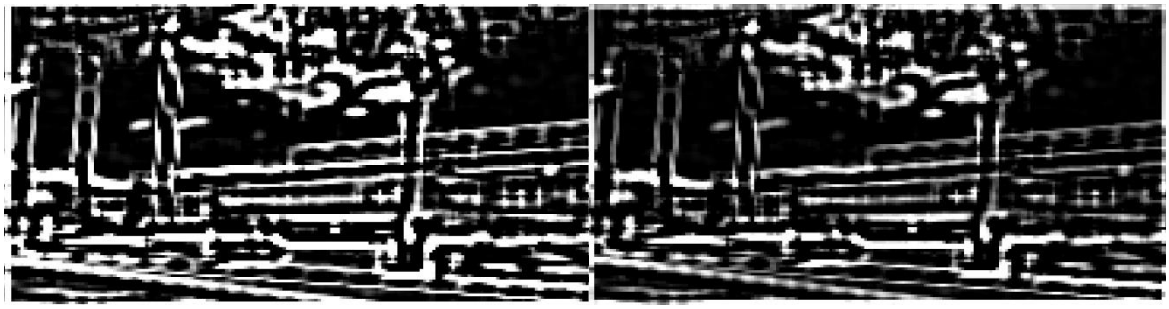

**Fig.2.14.a** **Fig.2.14.b**

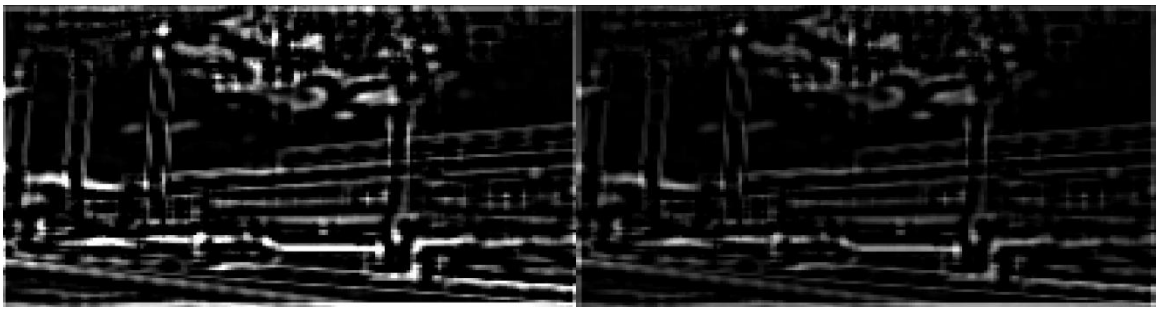

**Fig.2.14.c** **Fig.2.14.d**

**Fig.2.14** – DoG Images of Octave 3

The final output image with all the keypoints displayed as white dots:-

**Fig.2.15** – All the detected keypoints displayed on the original image.

Left most detected points:

| x | y |
|---|---|
| 0 | 231 |
| 0 | 266 |
| 0 | 324 |
| 0 | 351 |
| 0 | 386 |
| 1 | 274 |
| 1 | 333 |

# Task 3- Template Matching

**Objective:** Given a set of images, detect the location of a given template in the images.

**Steps:**

**1)** Apply a Gaussian blur on the image to smoothen the edges so only important features remain. (Gaussian blurring has been explained in Task 2)

**2)** Find the laplacian of the image in grayscale as well as the template in grayscale.(Laplacian of an image is approximated by the difference in Gaussian over successive values of an octave and preserves only the important edges.)

**3)** Match template using a suitable method, (here, we use Normalized Cross Correlation) which gave the best performance after testing among other methods.

- Methods tested were:-

**a)** Square difference –It calculates the summation of squared for the product of pixels subtraction between two images.

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

**b)** Normalized Square Difference – Uses the above after normalizing it by dividing with the sum of all square differences.

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

**c)** Normalized Cross Correlation - NCC determines the matching point between template and image by searching the location of maximum value in the image matrices.

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

where u and v are shifts along the x and y axis.

**d)** Cross Correlation

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

**e)** Normalized Cross Coefficient

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

*[Source: opencv.org]*

**4)** Set a threshold value for the coefficient of the template matched. If the threshold is more than a certain amount consider the template matched. Here, we found the optimal threshold value to be between 0.55 and 0.7

**5)** Iterate over all the given images and display it.

**More about the Method used(Normalized cross correlation):-**

Normalized Cross Correlation - NCC determines the matching point between template and image by searching the location of maximum value in the image matrices.

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

where u and v are shifts along the x and y axis.

Normalized Cross Correlation (NCC) is always chosen as similarity measure due to its robustness . Its computation is more complex if we compare to the other methods as it involves numerous multiplication, division and square root operations.

*[Source: Template Matching Using Sum of Squared Difference and Normalized Cross Correlation - M. B. Hisham, Shahrul Nizam Yaakob, Raof R. A. A, A.B A. Nazren, N.M.Wafi]*

**Templates:** 12 templates were created for testing purposes.

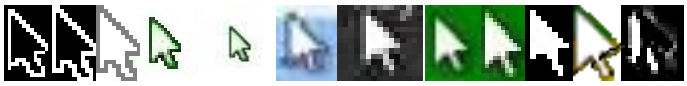

**Fig.3.1** – Cursor Templates

After testing all the templates the one shown below was used in the final code.

**Fig.3.2** – Selected Cursor Template

For the bonus problem which matches different types of cursors as well, the below templates were used.

**Fig.3.3**– Types of Cursor Templates

**Developed code:** (Python Code)

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
def display(name,img):
    cv2.namedWindow(name, cv2.WINDOW_NORMAL)
    cv2.imshow(name, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
template1 = cv2.imread('a1.png',0)
template1 = cv2.Laplacian(template1,cv2.CV_8U,5)
template1 = cv2.convertScaleAbs(template1)
template2 = cv2.imread('a2.png',0)
template2 = cv2.Laplacian(template2,cv2.CV_8U,5)
template2 = cv2.convertScaleAbs(template2)
template3 = cv2.imread('a3.png',0)
template3 = cv2.Laplacian(template3,cv2.CV_8U,5)
template3 = cv2.convertScaleAbs(template3)
template4 = cv2.imread('a4.png',0)
template4 = cv2.Laplacian(template4,cv2.CV_8U,5)
template4 = cv2.convertScaleAbs(template4)
template5 = cv2.imread('a5.png',0)
template5 = cv2.Laplacian(template5,cv2.CV_8U,5)
template5 = cv2.convertScaleAbs(template5)
template6 = cv2.imread('a6.png',0)
template6 = cv2.Laplacian(template6,cv2.CV_8U,5)
template6 = cv2.convertScaleAbs(template6)
template7 = cv2.imread('a7.png',0)
template7 = cv2.Laplacian(template7,cv2.CV_8U,5)
template7 = cv2.convertScaleAbs(template7)

w = 25
h = 25
res = []



threshold = 0.6
for i in range(1,4):
    for j in range(1,7):
        img = cv2.imread('t'+str(i)+'_'+str(j)+'.jpg')
        img1 = img.copy()
        img = cv2.GaussianBlur(img,(3,3),0)
        img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        img = cv2.Laplacian(img,cv2.CV_8U,5)
        img = cv2.convertScaleAbs(img)
        res = cv2.matchTemplate(img,template1,cv2.TM_CCORR_NORMED)
        '''
        loc = np.where( res >= threshold)
        for pt in zip(*loc[::-1]):
            cv2.rectangle(img1, pt, (pt[0] + w, pt[1] + h), (0,0,255 ), 2)
        display('res.png',img1)
        '''
        res1 = cv2.matchTemplate(img,template1,cv2.TM_CCORR_NORMED)
        loc1 = np.where( res1 >= threshold)
        for pt in zip(*loc1[::-1]):
            cv2.rectangle(img1, pt, (pt[0] + 23, pt[1] + 23), (255,0,0 ), 2)
        #display('res.png',img1)

        res2 = cv2.matchTemplate(img,template2,cv2.TM_CCORR_NORMED)
        loc2 = np.where( res2 >= threshold)
        for pt in zip(*loc2[::-1]):
            cv2.rectangle(img1, pt, (pt[0] + 21, pt[1] + 21), (255,0,0 ), 2)
        #display('res.png',img1)

        res3 = cv2.matchTemplate(img,template3,cv2.TM_CCORR_NORMED)
        loc3 = np.where( res3 >= threshold)
        for pt in zip(*loc3[::-1]):
            cv2.rectangle(img1, pt, (pt[0] + 25, pt[1] + 25), (255,0,0 ), 2)
        #display('res.png',img1)
```

```
 res4 = cv2.matchTemplate(img,template4,cv2.TM_CCORR_NORMED)
 loc4 = np.where( res4 >= threshold)
 for pt in zip(*loc4[::-1]):
     cv2.rectangle(img1, pt, (pt[0] + 16, pt[1] + 22), (255,0,0 ), 2)
# display('res.png',img1)


 res5 = cv2.matchTemplate(img,template5,cv2.TM_CCORR_NORMED)
 loc5 = np.where( res5 >= 0.6)
 for pt in zip(*loc5[::-1]):
     cv2.rectangle(img1, pt, (pt[0] + 16, pt[1] + 16), (255,0,0 ), 2)
 #display('res.png',img1)


 res6 = cv2.matchTemplate(img,template6,cv2.TM_CCORR_NORMED)
 loc6 = np.where( res6 >= threshold)
 for pt in zip(*loc6[::-1]):
     cv2.rectangle(img1, pt, (pt[0] + 17, pt[1] + 17 ), (255,0,0 ), 2)

 res7 = cv2.matchTemplate(img,template7,cv2.TM_CCORR_NORMED)
 loc7 = np.where( res7 >= threshold)
 for pt in zip(*loc7[::-1]):
     cv2.rectangle(img1, pt, (pt[0] + 17, pt[1] + 17 ), (255,0,0 ), 2)

 display('res.png',img1)
```

**Output:** The code was checked on certain images, some of the images with the templates detected are shown below.

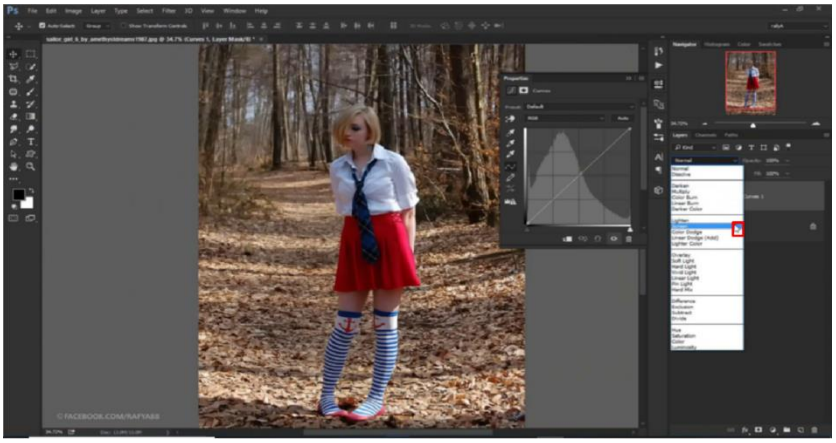

**Fig.3.4** – Cursor Detected(Arrow)

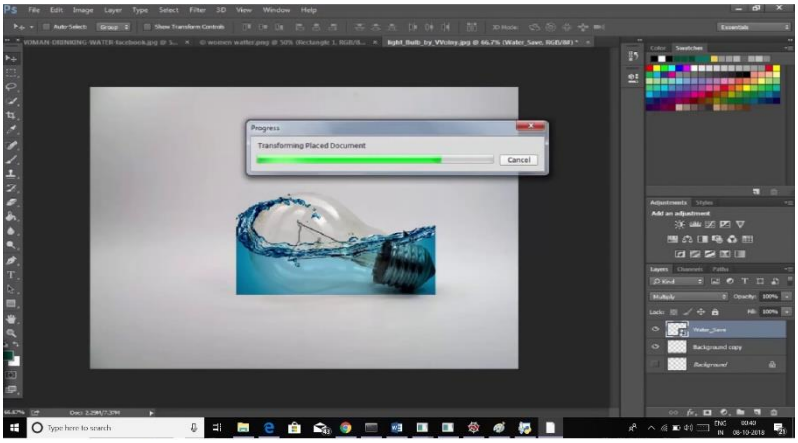


**Fig 3.5** – Cursor not detected(Arrow)

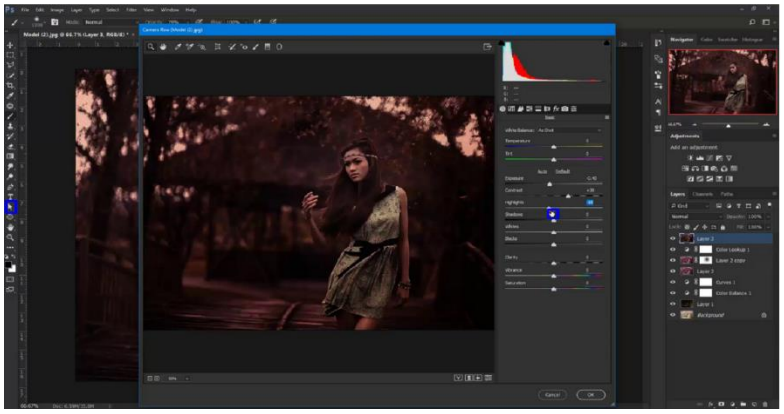

**Fig.3.6** – Cursor Detected(Hand And Arrow)

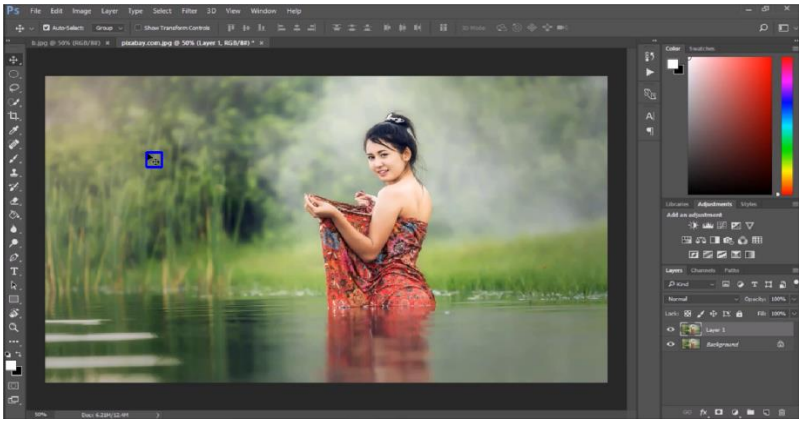

**Fig.3.7** – Cursor Detected(Black Arrow)

**Conclusion:**

**Task 1** – We learned how to detect edges in a given image by manually applying the Sobel matrix on the pixels of a given image.

**Task2** – We learned how Scale Invariant Feature Transform Works to detect keypoints in a given image.

**Task3** – We learned how to detect certain features over multiple images by matching a template.

# References


1) docs.opencv.org
2) programcreek.com
3) Template Matching Using Sum of Squared Difference and Normalized Cross Correlation - M. B. Hisham, Shahrul Nizam Yaakob, Raof R. A. A, A.B A. Nazren, N.M.Wafi
4) Youtube.com Channel: Cmputerphile - https://www.youtube.com/user/Computerphile
5) Aishack - aishack.in