CSE 573 Computer Vision & Image Processing

Project 1 Report

UBitName : abhave | personNumber : 50289049 | Date : October 8, 2018

**1 Edge Detection**

Source Code:

import cv2

import numpy as np

img = cv2.imread("C:/Users/abhis/Desktop/us/Courses/CVIP 573/homework and project/task1.png", 0)

height = img.shape[0]

width = img.shape[1]

#padding

outimg=[]

for i in range(height+2):

rowList=[]

for j in range(width+2):

rowList.append(0)

outimg.append(rowList)

for i in range(0,height):

for j in range(0,width):

outimg[i+1][j+1]=img[i][j]

x=np.array(outimg,dtype=np.uint8)

#flipped sobel operator

Gx=np.array([[-1,0,1],[-2,0,2],[-1,0,1]],dtype=np.float)

#flipped sobel operator

Gy=np.array([[-1,-2,-1],[0,0,0],[1,2,1]],dtype=np.float)

#convolution function

def convolve(X,Gx,Gy):

rows=X.shape[0]

columns=X.shape[1]

outx=[]

outy=[]

for i in range(rows):

rowList=[]

for j in range(columns):

rowList.append(0)

outx.append(rowList)

for i in range(rows):

rowList=[]

for j in range(columns):

rowList.append(0)

outy.append(rowList)

for i in range(1,rows-1):

for j in range(1,columns-1):

s=Gx[0][0]\*X[i-1][j-1]+Gx[0][1]\*X[i-1][j]+Gx[0][2]\*X[i-1][j+1]+\

Gx[1][0]\*X[i][j-1]+Gx[1][1]\*X[i][j]+Gx[1][2]\*X[i][j+1]+\

Gx[2][0]\*X[i+1][j-1]+Gx[2][1]\*X[i+1][j]+Gx[2][2]\*X[i+1][j+1]

p=Gy[0][0]\*X[i-1][j-1]+Gy[0][1]\*X[i-1][j]+Gy[0][2]\*X[i-1][j+1]+\

Gy[1][0]\*X[i][j-1]+Gy[1][1]\*X[i][j]+Gy[1][2]\*X[i][j+1]+\

Gy[2][0]\*X[i+1][j-1]+Gy[2][1]\*X[i+1][j]+Gy[2][2]\*X[i+1][j+1]

outx[i-1][j-1]=s

outy[i-1][j-1]=p

#eliminate zero values

x\_min=outx[0][0]

x\_max=outx[0][0]

for i in range(0,len(outx)):

for j in range(0,i):

if(x\_min>outx[i][j]):

x\_min=outx[i][j]

if(x\_max<outx[i][j]):

x\_max=outx[i][j]

y\_min=outy[0][0]

y\_max=outy[0][0]

for i in range(0,len(outy)):

for j in range(0,i):

if(y\_min>outy[i][j]):

y\_min=outy[i][j]

if(y\_max<outy[i][j]):

y\_max=outy[i][j]

pos\_edge\_x = (outx- x\_min) / (x\_max - x\_min)

pos\_edge\_y = (outy- y\_min) / (y\_max - y\_min)

return pos\_edge\_x,pos\_edge\_y

imgx,imgy=convolve(x,Gx,Gy)

cv2.imshow('hori.png',imgx)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imshow('verti.png',imgy)

cv2.waitKey(0)

cv2.destroyAllWindows()

Output:

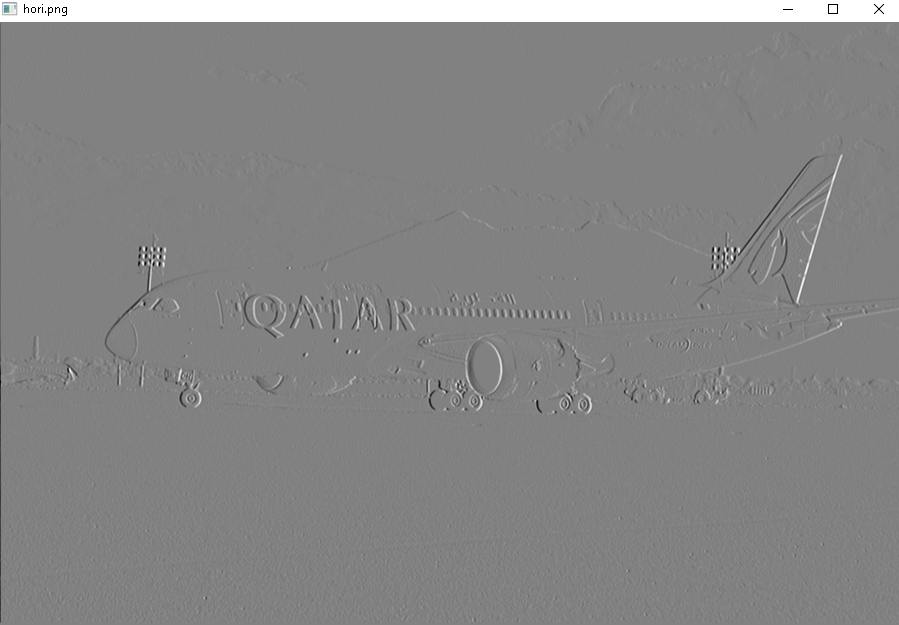


Figure 1: Image for showing edges along x direction

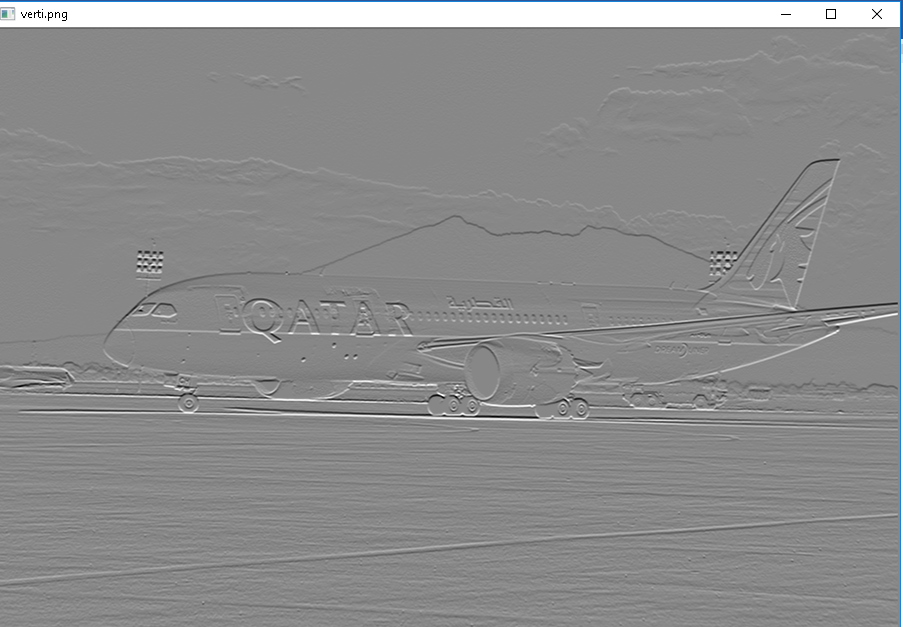


Figure 2: Image for showing edges along y direction

**2 Keypoint Detection**

Source Code [1]:

import cv2

import numpy as np

import math

img=cv2.imread("C:/Users/abhis/Desktop/us/Courses/CVIP 573/homework and project/task2.jpg", 0)

height2 = img.shape[0]

width2 = img.shape[1]

e=0

dog\_matrix=[]

count=0

#taking initial value of each octave

s=[0.5,math.sqrt(2),2\*math.sqrt(2),4\*math.sqrt(2)]

octave=[]

for i in range(0,4):

row=[]

newoct=s[i]\*math.sqrt(2)

for j in range(0,5):

row.append(newoct)

newoct\*=math.sqrt(2)

octave.append(row)

#initialization

new =[]

for i in range(height2):

rowList = []

for j in range(width2):

rowList.append(0)

new.append(rowList)

#calculates each value of kernel

def calcgaussiankernel(sigma):

out=[]

sum=0

for i in range(0,7):

row=[]

for j in range(0,7):

lower\_part=(((j-3)\*\*2 +(3-i)\*\*2)/(2\*sigma\*sigma))

a=float((1/(2\*math.pi\*sigma\*sigma))\*math.exp(-1\*lower\_part))

sum+=a

row.append(a)

out.append(row)

x=np.array(out,dtype=np.float)/sum

return x

#gaussian blur

def calcimg(blur,X,c,d):

height=X.shape[0]

width=X.shape[1]

outx=[]

for i in range(3,height-3):

row=[]

for j in range(3,width-3):

s= (blur[0][0]\*X[i-3][j-3]+blur[0][1]\*X[i-3][j-2]+blur[0][2]\*X[i-3][j-1]+blur[0][3]\*X[i-3][j]+blur[0][4]\*X[i-3][j+1]+blur[0][5]\*X[i-3][j+2]+blur[0][6]\*X[i-3][j+3]+\

blur[1][0]\*X[i-3][j-2]+blur[1][1]\*X[i-2][j-2]+blur[1][2]\*X[i-2][j-1]+blur[1][3]\*X[i-2][j]+blur[1][4]\*X[i-2][j+1]+blur[1][5]\*X[i-2][j+2]+blur[1][6]\*X[i-2][j+3]+\

blur[2][0]\*X[i-3][j-1]+blur[2][1]\*X[i-1][j-2]+blur[2][2]\*X[i-1][j-1]+blur[2][3]\*X[i-1][j]+blur[2][4]\*X[i-1][j+1]+blur[2][5]\*X[i-1][j+2]+blur[2][6]\*X[i-1][j+3]+\

blur[3][0]\*X[i-3][j]+blur[3][1]\*X[i][j-2]+blur[3][2]\*X[i][j-1]+blur[3][3]\*X[i][j]+blur[3][4]\*X[i][j+1]+blur[3][5]\*X[i][j+2]+blur[3][6]\*X[i][j+3]+\

blur[4][0]\*X[i-3][j+1]+blur[4][1]\*X[i+1][j-2]+blur[4][2]\*X[i+1][j-1]+blur[4][3]\*X[i+1][j]+blur[4][4]\*X[i+1][j+1]+blur[4][5]\*X[i+1][j+2]+blur[4][6]\*X[i+1][j+3]+\

blur[5][0]\*X[i-3][j+2]+blur[5][1]\*X[i+2][j-2]+blur[5][2]\*X[i+2][j-1]+blur[5][3]\*X[i+2][j]+blur[5][4]\*X[i+2][j+1]+blur[5][5]\*X[i+2][j+2]+blur[5][6]\*X[i+2][j+3]+\

blur[6][0]\*X[i-3][j+3]+blur[6][1]\*X[i+3][j-2]+blur[6][2]\*X[i+3][j-1]+blur[6][3]\*X[i+3][j]+blur[6][4]\*X[i+3][j+1]+blur[6][5]\*X[i+3][j+2]+blur[6][6]\*X[i+3][j+3])

row.append(s)

outx.append(row)

a=np.array(outx,dtype=np.uint8)

count=str(c)+str(d)

cv2.imwrite('Blur'+str(count)+'.jpg',a)

#selects octave value

def calcsigma(octno,octlength):

return octave[octno][octlength]]

#generates dog

def dog(img2,img1,c,d):

m=img2.shape[0]

n=img2.shape[1]

dog=[]

for i in range(0,m):

row=[]

for j in range(0,n):

x=int(img2[i][j])-int(img1[i][j])

row.append(x)

dog.append(row)

dog1=np.array(dog,dtype=np.uint8)

count=str(c)+str(d)

cv2.imwrite('DoG'+str(count)+'.jpg',dog1)

return dog1

#resizes image by 1/2 each time

def resize(img,m,n,a):

resize=[]

a+=1

for i in range(0,m):

row=[]

for j in range(0,n):

if(j%(2\*\*a)==0 and i%(2\*\*a)==0):

row.append(img[i][j])

else:

pass

resize.append(row)

temp=resize[::2\*\*a]

newimg=np.array(temp,dtype=np.uint8)

return newimg

#function to generate keypoints and 3 images

def genkeypoints():

for a in range(0,4):

for b in range(1,3):

c = int(b-1)

d = 'DoG' + str(a) + str(c) + '.jpg'

prev = cv2.imread(d,0)

e = 'DoG' + str(a) + str(b) + '.jpg'

img = cv2.imread(e,0)

f = int(b+1)

g = 'DoG' + str(a) + str(f) + '.jpg'

next1 = cv2.imread(g,0)

keypoint(prev, img, next1, a)

#displaying final image of each octave

octaveimg = np.array(blank, dtype=np.uint8)

d = 'Octave.jpg'

cv2.imshow(d,octaveimg)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imwrite(d,octaveimg)

def keypoint(prev, img, next1, a):

m = img.shape[0]

n = img.shape[1]

for i in range(1,m-1):

for j in range(1,n-1):

temp = img[i][j]

#counter to check if all conditions are satisfied

min = False

max = False

#comparing with its own 8 values

if temp < img[i-1][j-1] and temp < img[i-1][j] and temp < img[i-1][j+1] and temp < img[i][j-1] and temp < img[i][j+1] and temp < img[i+1][j-1] and temp < img[i+1][j] and temp < img[i+1][j+1]:

min = True

if temp > img[i-1][j-1] and temp > img[i-1][j] and temp > img[i-1][j+1] and temp > img[i][j-1] and temp > img[i][j+1] and temp > img[i+1][j-1] and temp > img[i+1][j] and temp > img[i+1][j+1]:

max = True

#comparing with values of prev image

if min == True or max == True:

if min == True:

if temp < prev[i-1][j-1] and temp < prev[i-1][j] and temp < prev[i-1][j+1] and temp < prev[i][j-1] and temp < prev[i][j+1] and temp < prev[i+1][j-1] and temp < prev[i+1][j] and temp < prev[i+1][j+1]:

min= True

else:

min = False

if max == True:

if temp > prev[i-1][j-1] and temp > prev[i-1][j] and temp > prev[i-1][j+1] and temp > prev[i][j-1] and temp > prev[i][j+1] and temp > prev[i+1][j-1] and temp > prev[i+1][j] and temp > prev[i+1][j+1]:

max == True

else:

max = False

#comparing with values of next image

if min == True or max == True:

if min == True:

if temp < next1[i-1][j-1] and temp < next1[i-1][j] and temp < next1[i-1][j+1] and temp < next1[i][j-1] and temp < next1[i][j+1] and temp < next1[i+1][j-1] and temp < next1[i+1][j] and temp < next1[i+1][j+1]:

min= True

else:

min = False

if max == True:

if temp > next1[i-1][j-1] and temp > next1[i-1][j] and temp > next1[i-1][j+1] and temp > next1[i][j-1] and temp > next1[i][j+1] and temp > next1[i+1][j-1] and temp > next1[i+1][j] and temp > next1[i+1][j+1]:

max == True

else:

max = False

#plotting the point in the empty image

if min == True or max == True:

I = i\*2\*\*a

J = j\*2\*\*a

blank[I][J] = 255

# all fuctions call to create kernel,convolve it and resize

for x in range(0,4):

for y in range(0,5):

sigma=calcsigma(x,y)

filter=calcgaussiankernel(sigma)

calcimg(filter,img,x,y)

img=cv2.imread("C:/Users/abhis/Desktop/us/Courses/CVIP 573/homework and project/task2.jpg", 0)

height2 = img.shape[0]

width2 = img.shape[1]

img=resize(img,height2,width2,x)

#print('calc part done')

#generates a numpy array of dog

for x in range(0,4):

dog\_rows=[]

for y in range(0,4):

e=int(y+1)

c='Blur'+str(x)+str(e)+'.jpg'

img2=cv2.imread(c,0)

d='Blur'+str(x)+str(y)+'.jpg'

img1=cv2.imread(d,0)

new=dog(img2,img1,x,y)

dog\_rows.append(new)

dog\_matrix.append(dog\_rows)

#print('Dog done')

blank =[]

for i in range(height2):

rowList = []

for j in range(width2):

rowList.append(0)

blank.append(rowList)

#function called to calculate maxima and minima

genkeypoints()

c= 0

list = []

for i in range(0,height2):

for j in range(0,width2):

if blank[i][j] == 255:

c += 1

list.append((i,j))

break

if c == 5:

break

print(list)

Output:

Figure 3-7: Images for Octave 2 (369 x 223 x 3)

Figure 8-12: Images for Octave 3 (182 x 108 x 3)

Figure 13-16: Images for DoG 2

Figure 17-20: Images for DoG 3

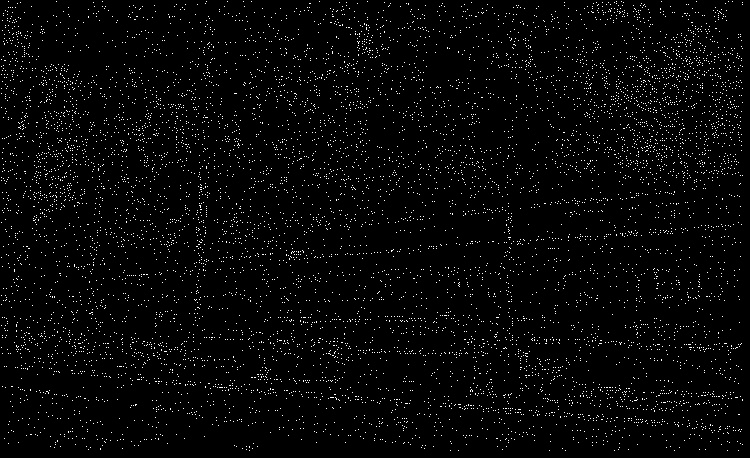


Figure 21: Keypoint Detected on Original Image

Coordinates of the 5 left-most detected key points:

(1, 8), (2, 20), (3, 17), (7, 8), (10, 5)

**3 Cursor Detection**

Source Code [2]:

Set A

import cv2

import numpy as np

def cursordetection():

img=cv2.imread('C:/Users/abhis/Desktop/us/Courses/CVIP 573/homework and project/task3/pos\_7.jpg')

source=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

template = cv2.imread('template1.png',0)

w=template.shape[1]

h = template.shape[0]

blur\_image=cv2.GaussianBlur(source,(3,3),0)

laplacian\_image = cv2.Laplacian(blur\_image,cv2.CV\_64F)

laplacian\_template = cv2.Laplacian(template,cv2.CV\_64F)

new=np.asarray(laplacian\_image,dtype=np.float32)

new1=np.asarray(laplacian\_template,dtype=np.float32)

ssd = cv2.matchTemplate(new,new1,cv2.TM\_CCOEFF\_NORMED)

threshold=0.48

loc=np.where(ssd>=threshold)

for pt in zip(\*loc[::-1]):

cv2.rectangle(img,pt,(pt[0]+w,pt[1]+h),(0,0,255),2 )

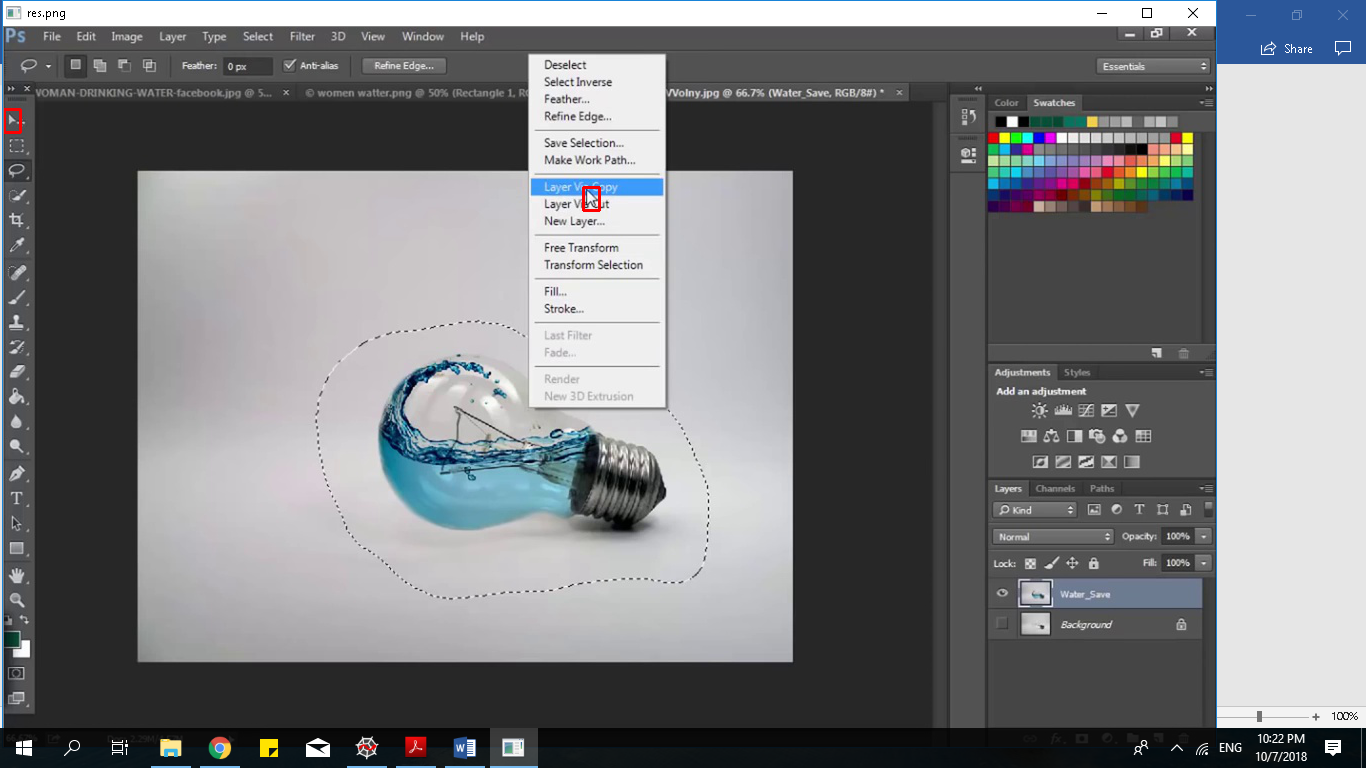
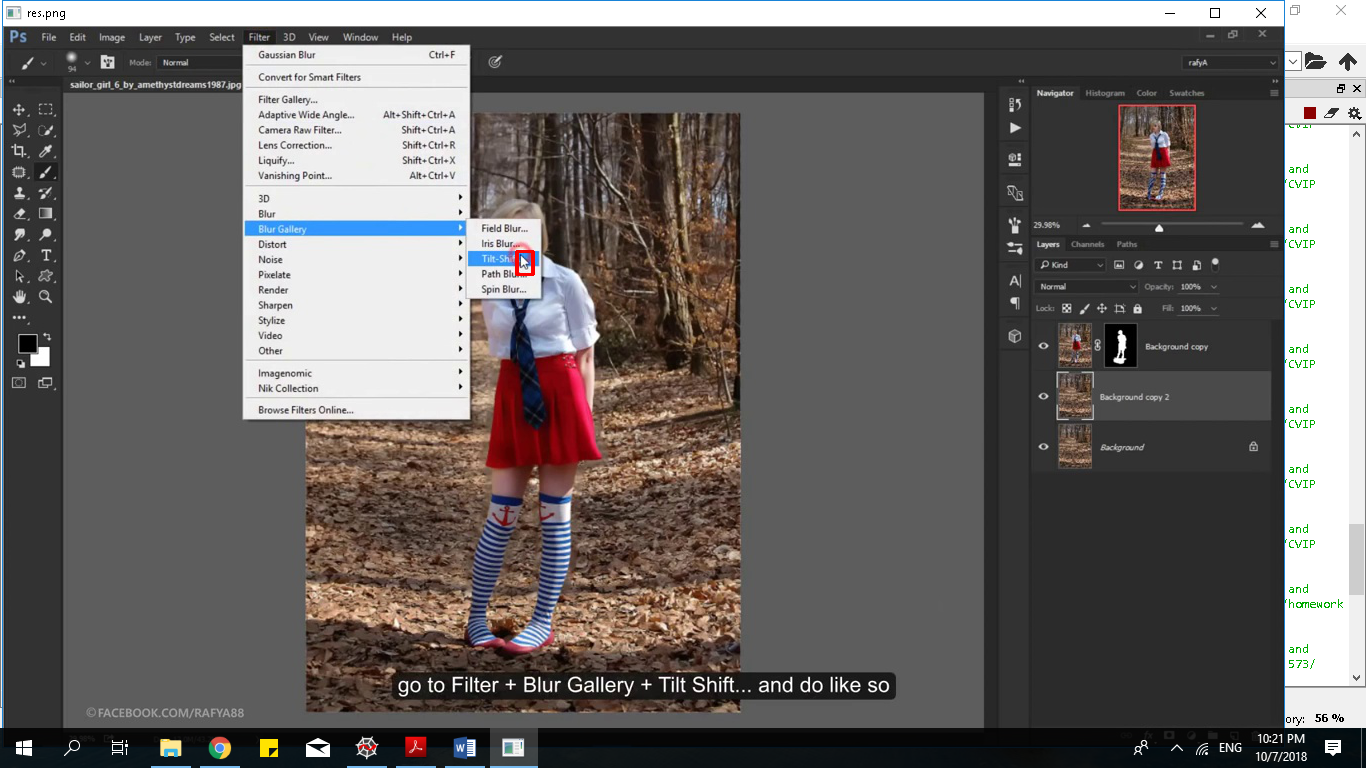
cv2.imshow('new.jpg',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

cursordetection()

Output:



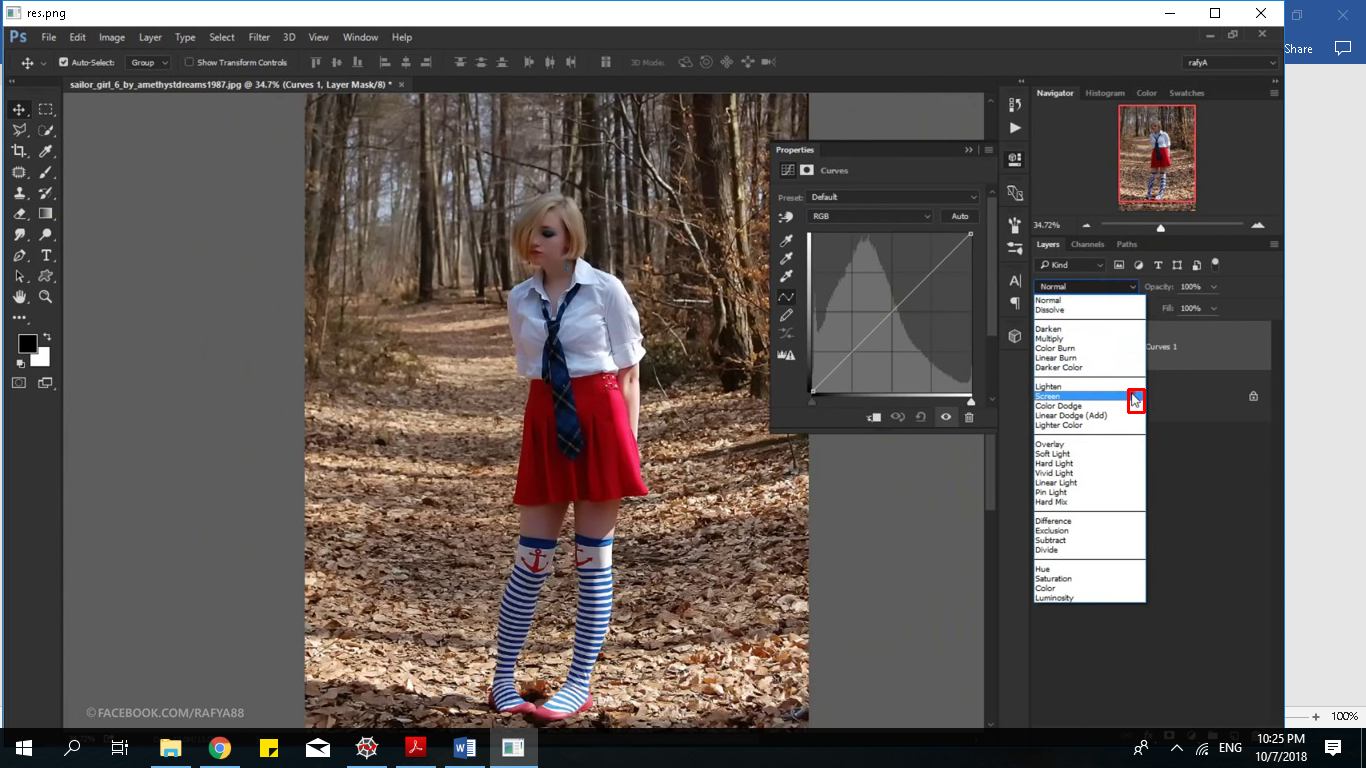
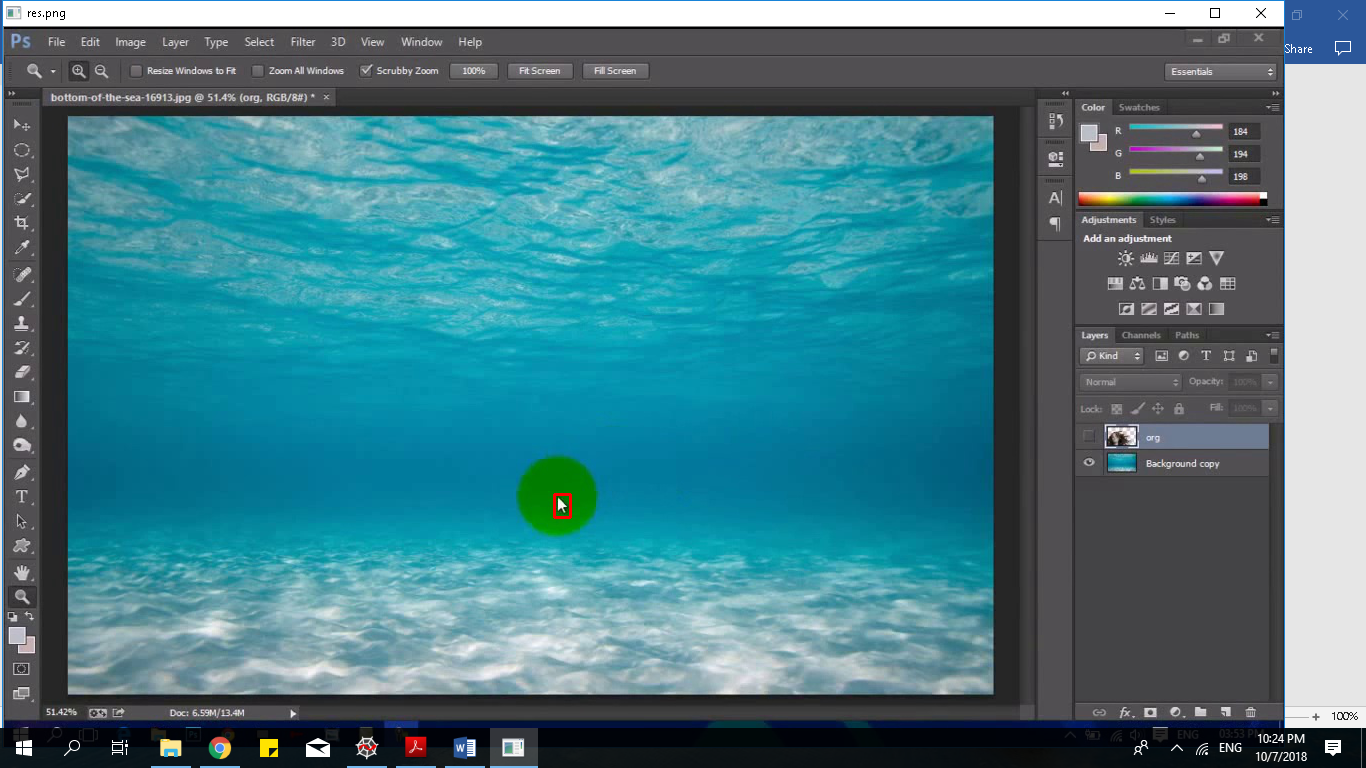


Figure 1-4: Cursor Detection Set A

Explanation:

I have used a custom template for the cursor which is generated from the given set. I have included the image file in the submission.

1. Read the image and template using cv2.imread()
2. Convert the image to grey by using cv2.COLOR\_BGR2GRAY
3. Convolute the image with a 3x3 Gaussian Kernel by using cv2.GaussianBlur()
4. Use Laplacian transform on both the image and the template by cv2.Laplacian() and store the value in an element and convert it to float32 using np.asrray()
5. Template matching done by cv2.matchTemplate(img\_name, template\_name ,cv2.TM\_CCOEFF\_NORMED)
6. Vary the threshold and compare.

All values above the threshold are to be marked using cv2.rectamgle() on the original image

All other values are neglected.

1. Display result image using cv2,imshow()

Set B

import cv2

import numpy as np

def cursordetection():

img=cv2.imread('C:/Users/abhis/Desktop/us/Courses/CVIP 573/homework and project/task3/set b/t2\_4.jpg')

img\_grey=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

template = cv2.imread('temp1.png',0)

template2 = cv2.imread('temp2.png',0)

template3 = cv2.imread('temp3.png',0)

w=template.shape[1]

h = template.shape[0]

w2=template2.shape[1]

h2= template2.shape[0]

w3=template3.shape[1]

h3= template3.shape[0]

blur\_image=cv2.GaussianBlur(img\_grey,(3,3),0)

laplacian\_blur\_image = cv2.Laplacian(blur\_image,cv2.CV\_64F)

laplacian\_blur\_template = cv2.Laplacian(template,cv2.CV\_64F)

laplacian\_blur\_template2 = cv2.Laplacian(template2,cv2.CV\_64F)

laplacian\_blur\_template3 = cv2.Laplacian(template3,cv2.CV\_64F)

new=np.asarray(laplacian\_blur\_image,dtype=np.float32)

new1=np.asarray(laplacian\_blur\_template,dtype=np.float32)

new2=np.asarray(laplacian\_blur\_template2,dtype=np.float32)

new3=np.asarray(laplacian\_blur\_template3,dtype=np.float32)

ssd = cv2.matchTemplate(new, new1,cv2.TM\_CCOEFF\_NORMED)

ssd2 = cv2.matchTemplate(new,new2,cv2.TM\_CCOEFF\_NORMED)

ssd3 = cv2.matchTemplate(new,new3,cv2.TM\_CCOEFF\_NORMED)

threshold=0.45

loc=np.where(ssd>=threshold)

for pt in zip(\*loc[::-1]):

cv2.rectangle(img,pt,(pt[0]+w,pt[1]+h),(0,0,255),2 )

threshold2=0.33

loc=np.where(ssd2>=threshold2)

for pt in zip(\*loc[::-1]):

cv2.rectangle(img,pt,(pt[0]+w2,pt[1]+h2),(0,0,255),2 )

threshold3=0.58

loc=np.where(ssd3>=threshold3)

for pt in zip(\*loc[::-1]):

cv2.rectangle(img,pt,(pt[0]+w3,pt[1]+h3),(0,0,255),2 )

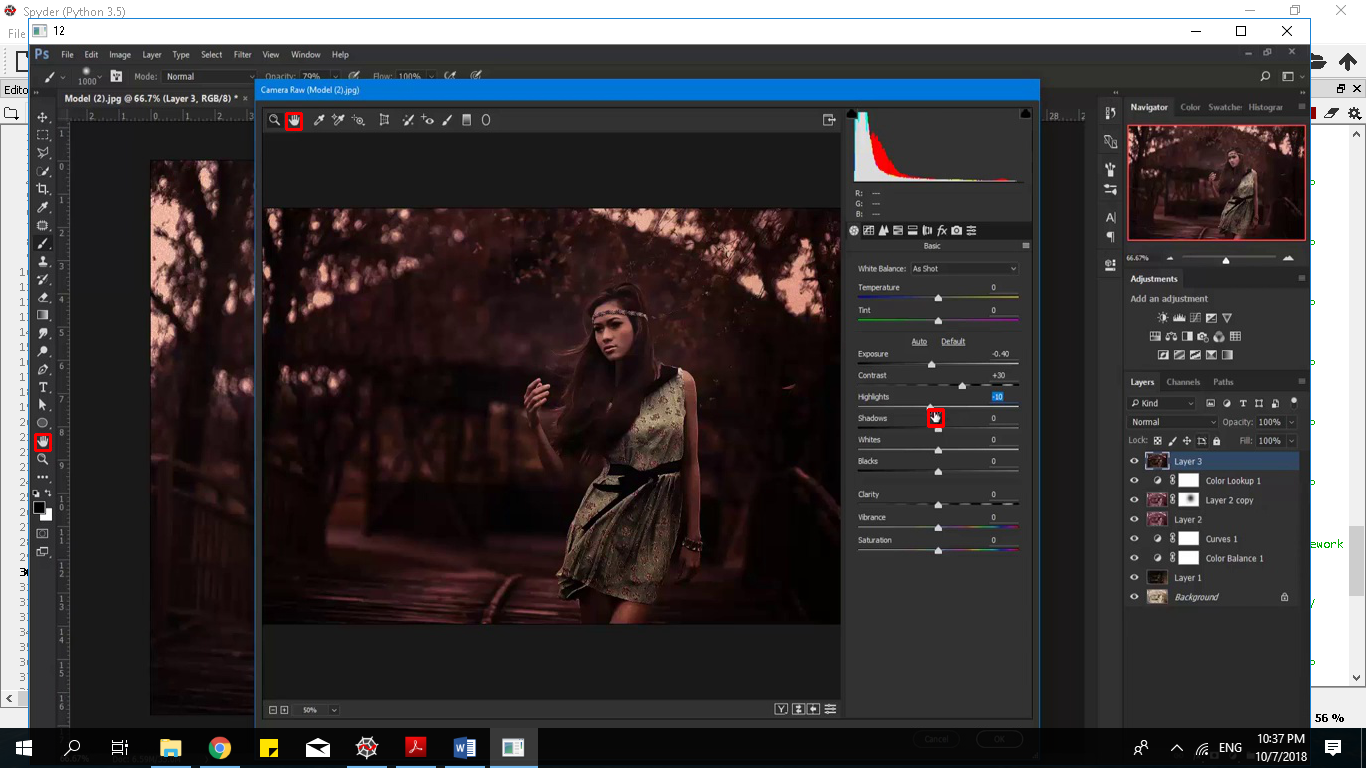
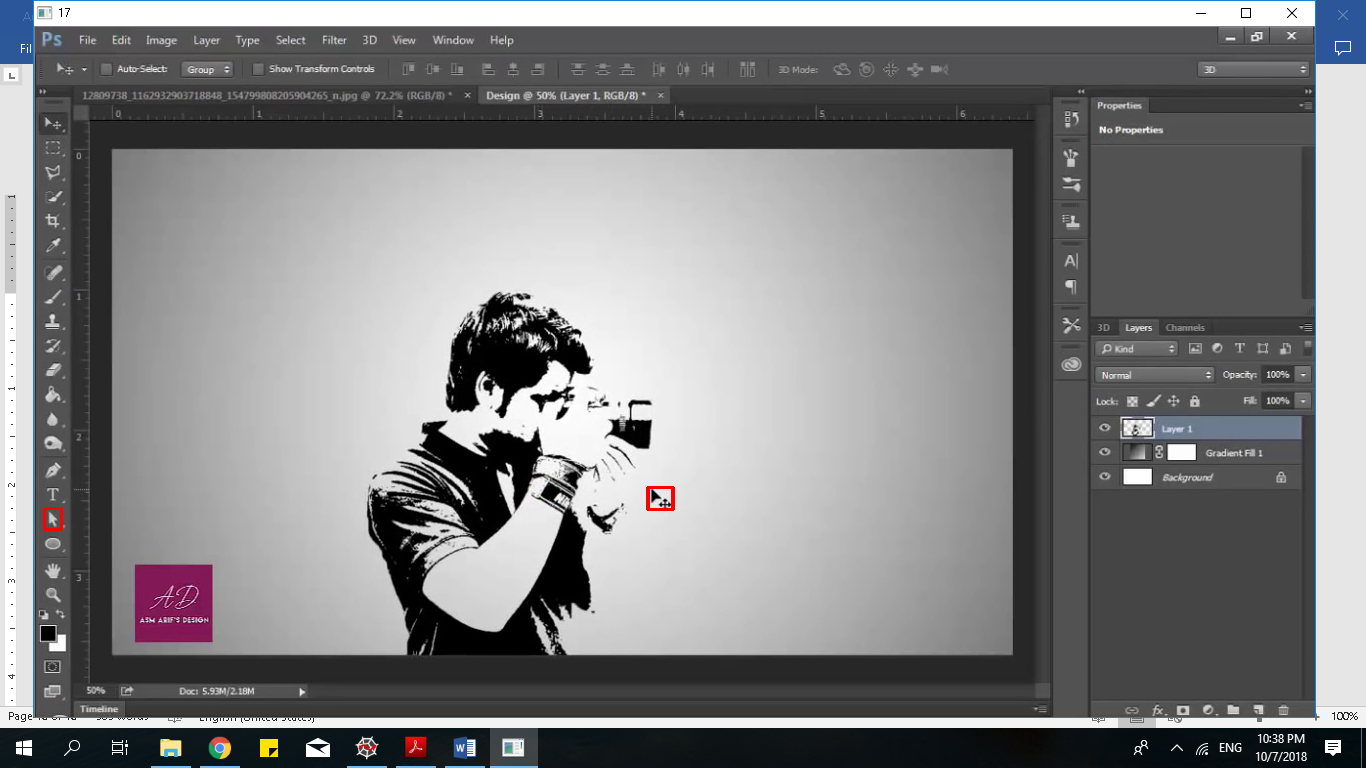
cv2.imshow('new.jpg',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

cursordetection()

Output:

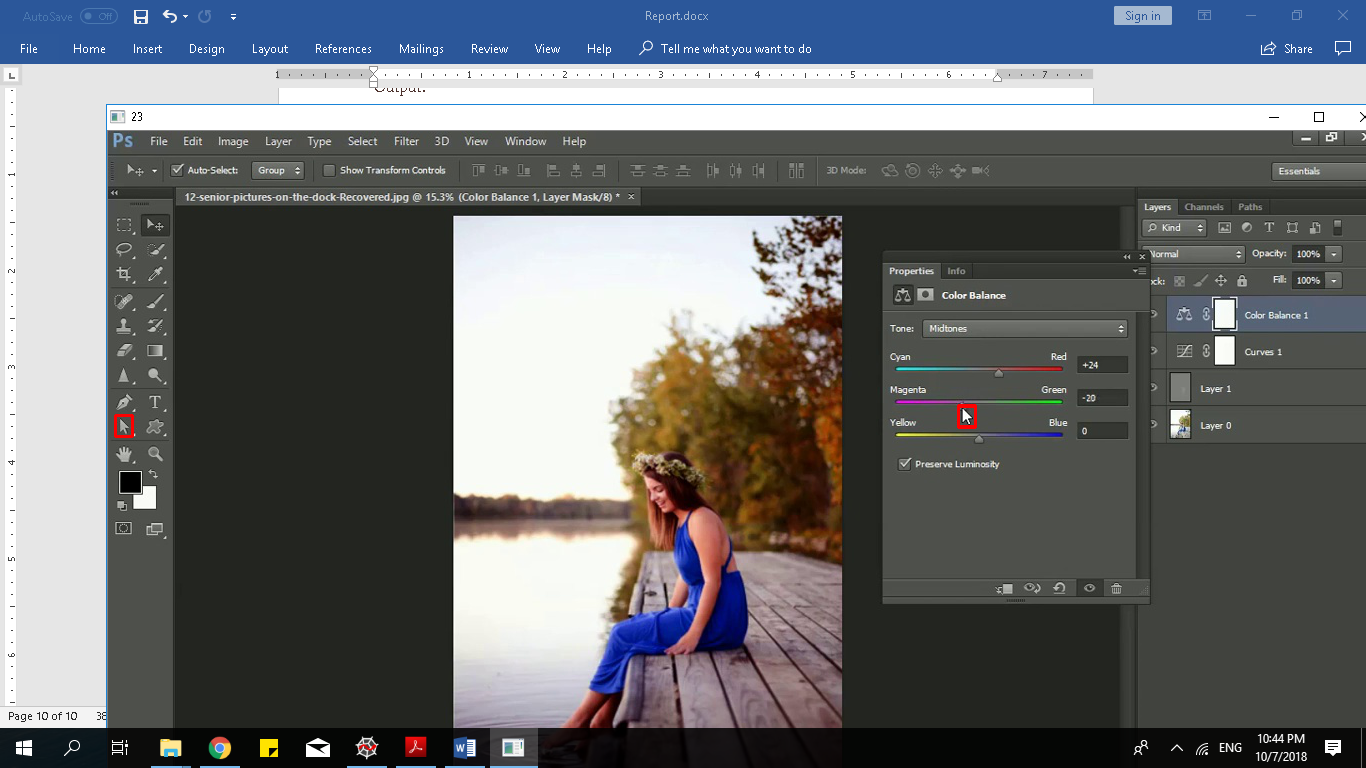
 

Figure 1-4: Cursor Detection Set B

The procedure for Set B is similar to Set A. The only difference is that each type of cursor is being matched in all the images and if found appropriate rectangle being made around it. Each cursor template is generated from the given images. Threshold values are varying for each type of cursor template.

References:

[1] –http://www.aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

[2] –https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/