Computer Vision And Image Processing

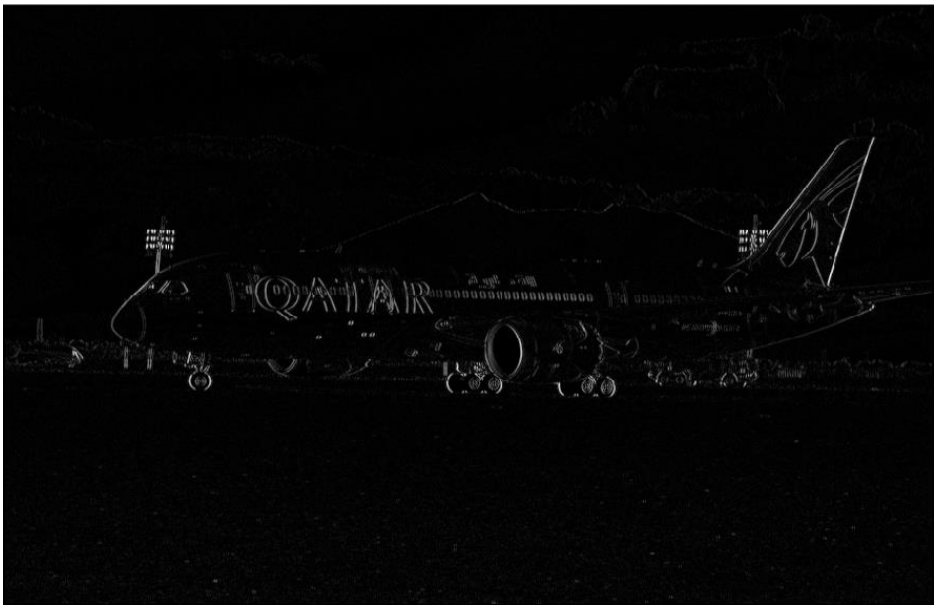
Cse573

Project report

UBIT :v33 UB# :50292192

TASK 1

Edges in y direction



edges in x direction

```
import cv2
from skimage import color
import numpy as np
img=cv2.imread("C:/task1.png",0)



img = img.tolist()


for row in range(len(img)):
     img[row] = [(i/255) for i in img[row]]
       #img[row][no] = img[row][no][0] * 0.0721 + img[row][no][1] * 0.7154 +
img[row][no][2] * 0.2125



#print(np.asarray(r))

cv2.namedWindow('imag', cv2.WINDOW_NORMAL)
cv2.imshow('imag', np.asarray(img))



def convolution(image, kernal):
    Matrix = [[0 for a in range(len(image[0]))] for b in range(len(image))]
    for z in range(2, len(image[0]) - 2):
        for x in range(2, (len(image) - 2)):
            value = 0;
            for i in range(len(kernal)):
                for j in range(len(kernal)):
                    u = i - len(kernal) // 2
                    v = j - len(kernal) // 2
                    value = value + kernal[i][j]* image[x - u][z - v]
            Matrix[x][z] = value

    return Matrix


sobe = [[1,0,-1], [2,0,-2], [1,0,-1]]

sobel_y = [[1, 2,1 ], [0, 0, 0], [-1, -2, -1]]



def calculate_positive_edges(img):
    after_pos_edg=[]
    minimum = min([min(j) for j in img])
    maximum = max([max(j) for j in img])
    den = maximum-minimum
    for i in range(len(img)):
        img[i][:] = [x - minimum for x in img[i]]
        img[i][:] = [x/den for x in img[i]]


def calculate_positive_edges_meth2(img):
    after_pos_edg=[]

    for i in range(len(img)):
        img[i]=[abs(j) for j in img[i]]
    maximum = max([max(j) for j in img])
```

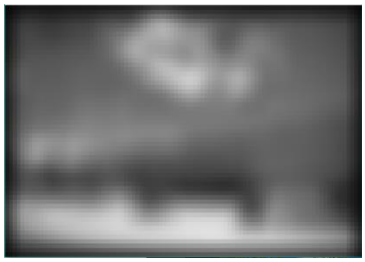
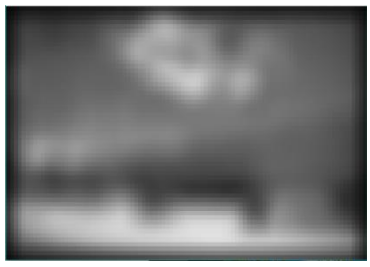
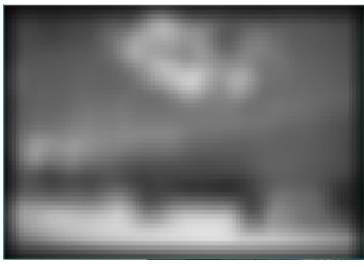
```
    for i in range(len(img)):
        img[i][:] = [x / maximum for x in img[i]]

    return img


edges_in_X = convolution(img,kernal= sobe)
edges_in_Y = convolution(img, kernal=sobel_y)

pos_edge_x =  calculate_positive_edges_meth2(edges_in_X)
pos_edge_y = calculate_positive_edges_meth2(edges_in_Y)

print(np.asarray(pos_edge_x))

final_x = img
final_y = img


cv2.namedWindow('x_edges', cv2.WINDOW_NORMAL)
cv2.imshow('x_edges', np.asarray(edges_in_X))

cv2.namedWindow('y_edges', cv2.WINDOW_NORMAL)
cv2.imshow('y_edges', np.asarray(pos_edge_y









cv2.waitKey(0)
```

edge detection code

TASK 2 :



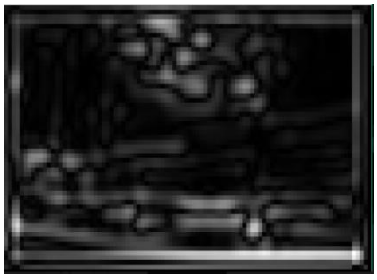
images in 2nd octave

images in 3rd octave




DOGs for 2nd octave

DOGs for 3rd octave

Key points

Source code for SIFT :

```
import cv2
import numpy as np
from math import sqrt ,pi ,e
import matplotlib.pyplot as plt
img=cv2.imread("C:/task2.jpg",0)
imag = img.tolist()
from scipy.ndimage import gaussian_filter
imog = cv2.imread("C:/task2.jpg")
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)


for row in range(len(imag)):
        imag[row] = [(i/255) for i in imag[row]]


def padd ( img):
    Matrix = [[0 for a in range(len(img[0])+6)] for b in range(len(img)+6)]
    for i in range(len(img)):
        for j in range (len(img[0])):
            Matrix[i+3][j+3]=img[i][j]

    return  Matrix

c = len(imag)-1
print(str(len(imag))+"  " +str(len(imag[c]))+"  ")

print(str(len(imag))+"  " +str(len(imag[0]))+"  ")
def GaussionKernal(s ,dimension):
    matr = [[0 for i in range(dimension)]for j in range(dimension)]
    sum =0
```

```
    for x in range (dimension):
        for y in range (dimension):
            u=x- dimension//2
            v=y-dimension//2
            matr[x][y] = (1/((2*pi)*(s**2)))*e**(-0.5*((u**2)+(v**2))/(s**2))
            sum = sum + matr[x][y]

    for row in range(len(matr)):
        matr[row] = [(i / sum) for i in matr[row]]
    return  matr




def calculate_positive_edges_meth2(img):
    after_pos_edg=[]

    for i in range(len(img)):
        img[i]=[abs(j) for j in img[i]]
    maximum = max([max(j) for j in img])
    for i in range(len(img)):
        img[i][:] = [x / maximum for x in img[i]]

    return img

def convolution( imag , kernal):
    Matrix = [[0 for a in range(len(imag[0]))] for b in range(len(imag))]
    imag = padd(imag)

    for z in range(3,len(imag[0])-3):
        for x in range(3, (len(imag)-3)):
            value = 0;

            for i in range(len(kernal)):
                for j in range(len(kernal)):
                    u = i - len(kernal) // 2
                    v = j - len(kernal) // 2
                    value = value + kernal[i][j] * imag[x - u][z - v]
            #print(x)
            #print(z)
            Matrix[x-3][z-3] = value

    return Matrix



def reduce( imag , kernal):
    #imag = padd(imag)
    Matrix = [[0 for a in range(len(imag[0])//2)] for b in range(len(imag)//2)]
    for z in range(1, (len(imag[0])//2)-1):
        for x in range(1, (len(imag)//2)-1):
            value = 0;
            for i in range(len(kernal)):
                for j in range(len(kernal)):
                    u = i - len(kernal) // 2
                    v = j - len(kernal) // 2
                    value = value + kernal[i][j] * imag[2*x - u][2*z - v]
            Matrix[x][z] = value
    return Matrix

sam = [[0,0,0,0,0],[0,0,0,0,0],[0,0,1,0,0],[0,0,0,0,0],[0,0,0,0,0]]

def generate_octave(si,imag):
```

```
    octave=[]
    #gauus = GaussionKernal(s=5, dimension=3)
    sig=si
    for i in range(0,5):
        gauus = GaussionKernal(s=sig, dimension=7)
        blurred = convolution(imag, gauus)
        #blurred = gaussian_filter(imag, sigma=si)
        octave.append(blurred)
        sig =sig*sqrt(2)
    return octave

def generate_scale_space(si, image):
    scale_space=[]
    si =1/sqrt(2)

    oct1=generate_octave(si,imag)
    scale_space.append(oct1)

    next_oct = reduce(oct1[2],sam)
    oct2=generate_octave(sqrt(2),next_oct)
    scale_space.append(oct2)

    next_oct = reduce(oct2[2],sam)
    oct3=generate_octave(2*sqrt(2),next_oct)
    scale_space.append(oct3)

    next_oct=reduce(oct3[2],sam)
    oct4 = generate_octave(4* sqrt(2), next_oct)
    scale_space.append(oct4)

    return scale_space

def subtract(m1 ,m2):
    matr = [[0 for i in range(len(m1[0]))]for j in range(len(m1))]


    for i in range(len(m1)):
        for j in range(len(m1[0])):
            matr[i][j] = m1[i][j]- m2[i][j]

    return matr



def generate_DOG(scale_space):

    all_dogs =[]
    for i in range(len(scale_space)):
        dog = []
        for j in range(len(scale_space[0])-1):
            x1 = subtract(scale_space[i][j],scale_space[i][j+1])
           # x1=calculate_positive_edges_meth2(x1)
            dog.append(x1)
        all_dogs.append(dog)

    return all_dogs


sc = generate_scale_space(0,imag)
m =1
for i in range(len(sc)):
    for j in range(len(sc[0])):
```

```
        plt.subplot(len(sc),len(sc[0]),m)
        plt.imshow(sc[i][j],cmap='gray')
        m=m+1

plt.show()


#sc = generate_scale_space(0,imag)
dogs = generate_DOG (sc)

m =1
for i in range(len(dogs)):
    for j in range(len(dogs[0])):

        plt.subplot(len(dogs),len(dogs[0]),m)
        plt.imshow(dogs[i][j],cmap='gray')
        m=m+1

plt.show()

def keypointdetection( scalespace ):
    kpm = []
    count = 1
    for oc in scalespace:
        Matrix = [[0 for a in range(len(oc[0][0]))] for b in range(len(oc[0]))]
        dh = len(oc[0])
        dw = len(oc[0][0])
        for i in range(1, dh - 1):
            for j in range(1, dw - 1):
                b1 = 0
                c1 = 0
                for l in range(3):
                    for k in range(3):
                        b = max(oc[0][i - 1 + l][j - 1 + k], oc[1][i - 1 + l][j - 1 +
k],
                                oc[2][i - 1 + l][j - 1 + k])
                        c = min(oc[0][i - 1 + l][j - 1 + k], oc[1][i - 1 + l][j - 1 +
k],
                                oc[2][i - 1 + l][j - 1 + k])
                        if (b >= b1):
                            b1 = b
                        if (c <= c1):
                            c1 = c

                if (oc[1][i][j] >= b1):
                    Matrix[i][j] = 2 * oc[1][i][j]

                    imog[i*count][j*count]=(0,0,255)
                elif (oc[1][i][j] <= c1):
                    Matrix[i][j] = 2 * oc[1][i][j]
                    imog[i*count][j*count]= (0,0,255)
        count = count *2
        kpm.append(Matrix)
    return kpm




print (np.nonzero(np.asarray(dogs[1][1])))
print(np.nonzero(np.asarray(dogs[1][2])))
```

```
kpm = keypointdetection(dogs)
#opium = keypointmap(kpm)
j=1
for ima in dogs[1]:
    cv2.imwrite('C:/dog2/x'+str(j)+'.jpg', np.asarray(ima))
    j= j+1

j=1
for ima in dogs[2]:
    cv2.imwrite('C:/dog3/y'+str(j)+'.jpg', np.asarray(ima))
    j= j+1

sc = generate_scale_space(0,imag)
cv2.namedWindow('', cv2.WINDOW_NORMAL)
cv2.imshow('dog_2_1', np.asarray(dogs[1][0]))

cv2.namedWindow('oc2', cv2.WINDOW_NORMAL)
cv2.imshow('dog_2_2',np.asarray(dogs[1][1]))

cv2.namedWindow('oc3', cv2.WINDOW_NORMAL)
cv2.imshow('dog_2_3', np.asarray(dogs[1][2]))

cv2.namedWindow('oc4', cv2.WINDOW_NORMAL)
cv2.imshow('dog_2_4',np.asarray(dogs[1][3]))

cv2.namedWindow('oc5', cv2.WINDOW_NORMAL)
cv2.imshow('dog_3_1',np.asarray(dogs[2][0]))

cv2.namedWindow('oc6', cv2.WINDOW_NORMAL)
cv2.imshow('dog_3_2',np.asarray(dogs[2][1]))

cv2.namedWindow('oc7', cv2.WINDOW_NORMAL)
cv2.imshow('dog_3_3',np.asarray(dogs[2][3]))
cv2.namedWindow('oc8', cv2.WINDOW_NORMAL)
cv2.namedWindow("sdf",cv2.WINDOW_NORMAL)
cv2.imshow ("sdf",np.asarray(imog))
cv2.waitKey(0)
```

TASK3:

In task3, a template is created by cropping the cursor from pos4. Fig3 shows the cropped cursor

Fig 3

Two copies of this template is generated. First copy of the template is subjected to thresholding by adjusting the values such that the background of the cursor is black. In the next step gaussian blur of 3 and sigma 1 is applied on the second copy of the template and Laplacian of the gaussian blurred image is calculated. similarly, Image is also subjected to gaussian blur and laplace tranfrom. Now all the rows that are entirely zero in first copy of the image are removed from second copy of the image.

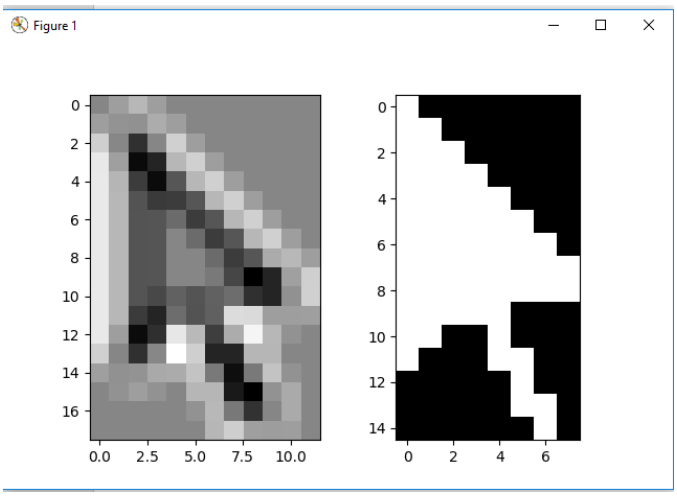


Fig 4

While stripping the columns, 2 columns on the either side of the image are left out because LOG of the template contains the information related to the cursor in this columns. Now, the final template is matched with the images using Normalized cross correlation. After matching the image using NCC, pixels greater than 0.6 are detected with a rectangle

Source code for task3:

```
from scipy import fftpack
import numpy as np
import cv2
import ImageTk
from PIL import ImageTk as tk
import pylab as py
from skimage import data, color
from skimage.feature import match_template
from matplotlib import pyplot as plt
import os
import homofilt


def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename),0)
        if img is not None:
            images.append(img)
    return images

all_images = load_images_from_folder("C:/task3_bonus/")
cv2.imshow("all_images",all_images[1])
```

```
image = cv2.imread("C:/task3/pos_1.jpg",0)
template = cv2.imread("C:/task3/template.png")
template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
ori_template2 = cv2.imread("C:/pos_ (4).jpg",0)
template2 = ori_template2
cv2.imshow("sdfs",template2)

template = template2
templ = template2
x=np.where((template>200))
template[x]=250
y=np.where((template<150))
template[y]=200
z=np.where((template>101) & (template<200))
template[z]=245
print(template.shape)

templ[y] = 0
print(templ.shape)
templ=templ[:,12:24]
templ = templ[~np.all(template== 0, axis=1)]


templ= cv2.GaussianBlur(templ,(3,3),0)
templ = cv2.Laplacian(templ,cv2.CV_64F)

x=np.where((template2>200))
template2[x]=250
y=np.where((template2<165))
z=np.where((template2>175) & (template2<200))
template2[z]=245
template2[y]=165
template2 = template2[:, 12:30]


template2 = template2[~np.all(template2 == 165, axis=1)]
ab=np.transpose(template2)
ab = ab[~np.all(ab == 165, axis=1)]
template2 = np.transpose(ab)
print(template.shape)


scale_w = 11/14
scale_h = 9/24


results = []
for img in all_images:

    blur = cv2.GaussianBlur(img, (3, 3), 0)
    lap = cv2.Laplacian(blur, cv2.CV_64F)
    result = match_template(lap, templ)
    results.append(result)

threshold = 0.6
start_width, start_height = image.shape[::-1]


cv2.imshow("ncc",result)
cv2.imshow("image",image)
plt.subplot(121),plt.imshow(templ, cmap = 'gray')
plt.subplot(122),plt.imshow(template2, cmap = 'gray')
```

```
plt.show()


threshold = 0.6

for i  in range(len(results)):
    loc = np.where(results[i]>= threshold)
    w, h = template2.shape[::-1]
    scale = 1
    for pt in zip(*loc[::-1]):
        cv2.rectangle(all_images[i] , (int(pt[0] * scale), int(pt[1] * scale)),
                      (int((pt[0] + w + 4) * scale), int((pt[1] + h + 4) * scale)),
                      ((255, 255, 255)), 2)


j=0
for ima in all_images:
    cv2.imwrite('C:/results_bonus/Images'+str(j)+'.jpg', ima)
    j= j+1

cv2.waitKey(0)
```