CSE 573 Project 1 Report

Hardik Babariya, Person no : 50289621

October 8, 2018

1 Task 1 : Edge Detection

Figure 1: Edges along X-Axis

Figure 2: Edges along Y-Axis

Figure 3: Edges along Both-Axis

Source Code:

```
import cv2
import numpy as np

image = cv2.imread('task1.png', 0)

rows = image.shape[0]
columns = image.shape[1]

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

outputX = np.zeros((rows,columns), dtype = np.float)
outputY = np.zeros((rows,columns), dtype = np.float)
magnitude = np.zeros((rows,columns), dtype = np.float)
positiveX = np.zeros((rows,columns), dtype = np.float)
positiveY = np.zeros((rows,columns), dtype = np.float)

for row in range(0,rows-2):
    for column in range(0,columns-2):

        window=image[row:row+3,column:column+3]
       # print(window);
        gx = ( window.copy() * Th ).sum();
        gy = ( window.copy() * Tv ).sum();

        outputX[row+1][column+1]=gx
        outputY[row+1][column+1]=gy

        g = np.sqrt(gx ** 2 + gy ** 2)
```

```
        magnitude[row+1][column+1] = g

magnitude /= magnitude.max()
positiveX = (outputX - outputX.min()) / (outputX.max() - outputX.min())
positiveY = (outputY - outputY.min()) / (outputY.max() - outputY.min())


cv2.imwrite('outputX.png', outputX)
cv2.imwrite('outputY.png', outputY)
cv2.imwrite('magnitude.png', magnitude * 255)
cv2.imwrite('positiveX.png', positiveX * 255)
cv2.imwrite('positiveY.png', positiveY * 255)
```

2 Task 2 : Keypoint Detection

Source Code

```
"""
To reduce the testing time of the program, the following code computes
the Difference of Gaussians(DoGs) for only one octave. However,
by changing the value of sigma in the 'gauss' function, it can be used
to calculate DoGs for any octave.
"""

import numpy as np
import cv2

# Gaussian kernel
def gauss(l, sig):
    """
    creates gaussian kernel with side length l and a sigma of sig
    """

    ax = np.arange(-l // 2 + 1., l // 2 + 1.)
    xx, yy = np.meshgrid(ax, ax)

    kernel = np.exp(-(xx**2 + yy**2) / (2. * sig**2))

    return kernel / np.sum(kernel)

#T=gauss(7,5.)
T1=gauss(7, 1/(2**(1/2)))
T2=gauss(7, 1 )
T3=gauss(7, (2**(1/2)) )
```

```
T4=gauss(7, 2)
T5=gauss(7, 2*(2**(1/2)) )

img = cv2.imread('task2.jpg',0)

#img=(img[0::2,0::2])              #  Image is resized by dropping alternative
                                   #  columns and rows to generate octave 2
print(img.shape)                   #  Dimension (229,375)

#img=(img[0::2,0::2])             #  Half of the original Image is resized by
                                   #   dropping alternative columns and rows
                                   #   to generate octave 3


print(img.shape)                   # Dimension (115,188)

img1=img.copy();
img2=img.copy();
img3=img.copy();
img4=img.copy();
img5=img.copy();

images= [ img1,img2,img3,img4,img5 ]
scales = [ T1,T2,T3,T4,T5 ]
blured=[]
dog=[]

rows=img.shape[0];
columns=img.shape[1];
print(rows,columns)

zero_matrix=np.zeros((rows,columns),dtype=np.float)

#  convolve image and gaussian kernel
for i in range(len(images)):
    output=zero_matrix.copy()
    for row in range(0,rows-6):
        for column in range(0,columns-6):

            window=images[i][row:row+7,column:column+7]
            # print(window);
            sum1 = ( window * scales[i] ).sum();

            output[row+3][column+3]=sum1
    blured.append(output.copy())
```

```
for i in range(len(blured)-1):
    cv2.imwrite('octave1_img' + str(i) + '.jpg', blured[i])
    dog.append(blured[i+1]-blured[i])

cv2.imwrite('octave1_img' + str(4) + '.jpg',blured[4])

print(len(dog))
#cv2.imshow('org',img)

# DoGs of
for i in range(len(dog)):
    cv2.imwrite('dog' + str(i+1) + '_' + str(i) + str(1) +'.jpg' ,dog[i])

dog_x=dog[0].shape[0];
dog_y=dog[0].shape[1];

dog_matrix=np.zeros((dog_x,dog_y),dtype=np.float)
extrema = []
for i in range(1,len(dog)-1):
    for row in range(0,dog[0].shape[0]-2):
        for column in range(0,dog[0].shape[1]-2):

            middle = dog[i][row:row+3, column:column+3]
            first = dog[i-1][row:row+3, column:column+3]
            last = dog[i+1][row:row+3, column:column+3]

            pivot=middle[1][1]
            mini=pivot
            maxi=pivot

            for roww in range(middle.shape[0]):
                for coll in range(middle.shape[1]):
                    if(middle[roww][coll]<mini):
                        mini=middle[roww][coll]
                    if(first[roww][coll]<mini):
                        mini=first[roww][coll]
                    if(last[roww][coll]<mini):
                        mini=last[roww][coll]

                    if(middle[roww][coll]>maxi):
                        maxi=middle[roww][coll]
                    if(first[roww][coll]>maxi):
                        maxi=first[roww][coll]
                    if(last[roww][coll]>maxi):
                        maxi=last[roww][coll]
```

```
                    if(pivot<maxi and pivot>mini):
                        break;
                if(pivot<maxi and pivot>mini):
                    break;

            if( pivot==maxi or pivot==mini ):
                dog_matrix[row+1][column+1] = 255
            else:
                dog_matrix[row+1][column+1]=0
    extrema.append(dog_matrix.copy())

cv2.imwrite("extrema1_1.jpg",extrema[0])
cv2.imwrite("extrema2_1.jpg",extrema[1])

for i in range(extrema[0].shape[0]):
    for j in range(extrema[0].shape[1]):
        if(extrema[1][i][j]==255):
            img[i][j]=255
cv2.imwrite("whiteDotted.jpg",img)

counter=0


"""
We will start detecting points from third row and third column because the first
row and column are unattended because of 3*3 gaussian mask. So,there will be more
white dots in the first and second row,column because of blur effect, which are
not the keypoints

"""
for j in range(2,img.shape[1]):
    for i in range(2,img.shape[0]):
        if(counter==5):
            break

        if(img[i][j]==255):
            counter+=1
            print("Left most keypoint number " + str(counter)  )
            print("Detected at position: (" + str(i) + "," + str(j) + ")" )
    if(counter==5):
        break


cv2.waitKey(100000)
```

```
cv2.destroyAllWindows()
```

Coordinates of the five left most detected points:

(5,2), (6,2), (14,2), (22,2), (25,2)

Octave 2 and Octave 3 representation

Figure 4: Images in the second octave Dimensions = [width: 375, height: 229] pixels

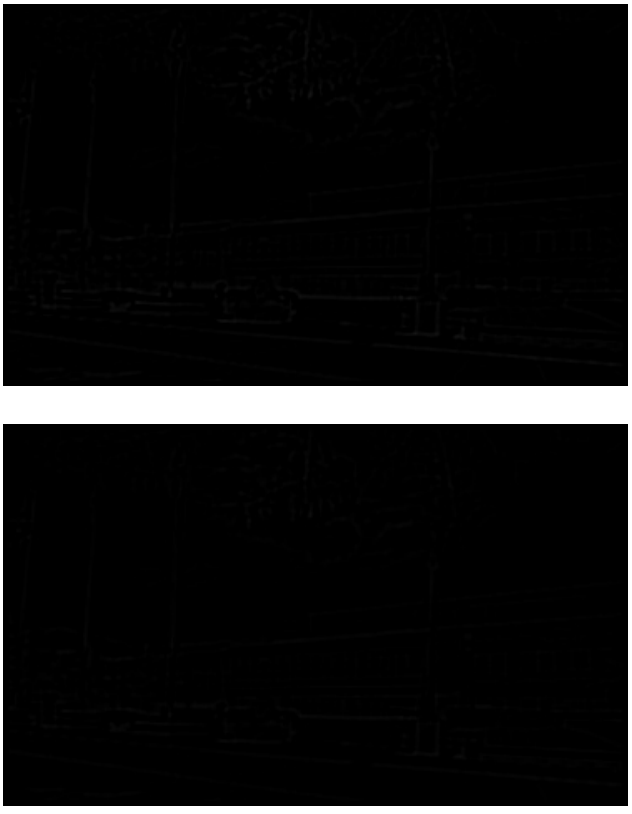
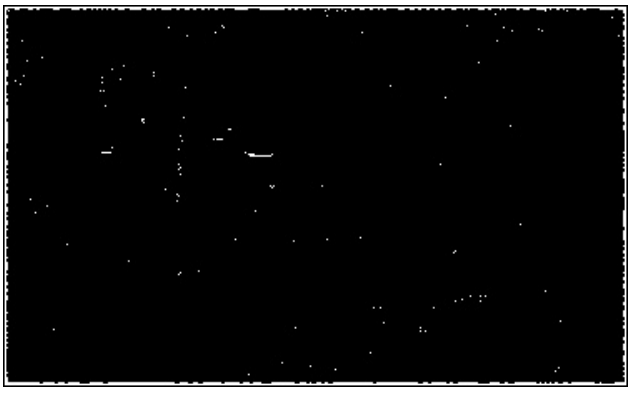


Figure 5: DoGs in the second octave



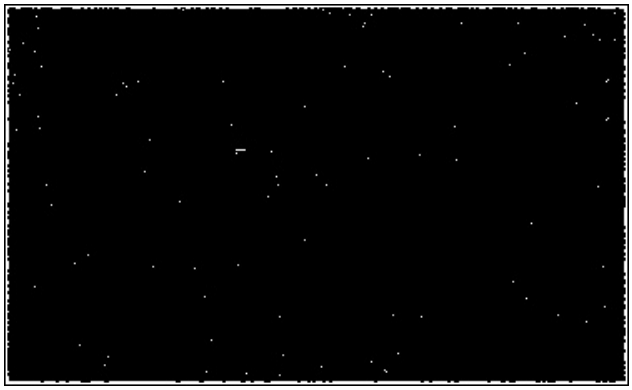
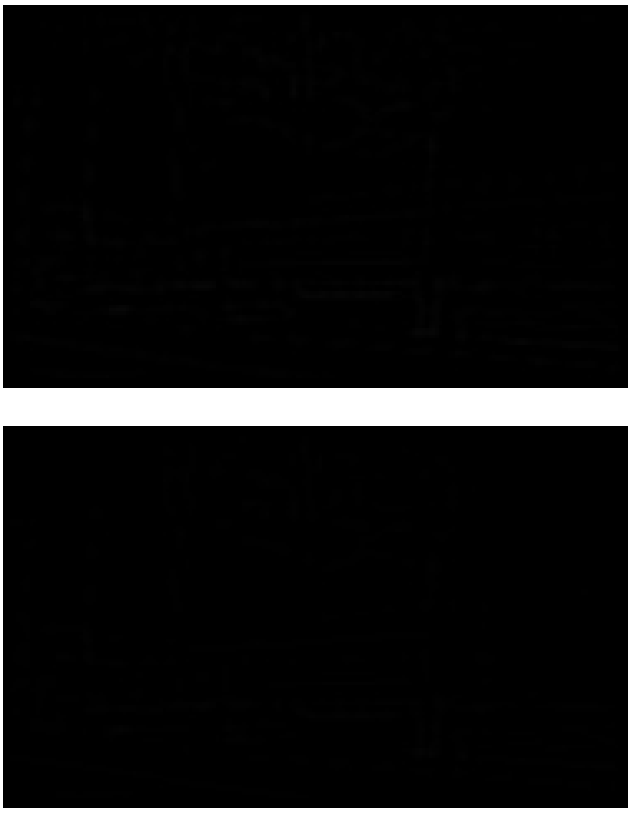


Figure 6: Maxima & minima of DoGs in the second octave

Figure 7: Images in the third octave Dimensions = [width: 188, height: 115] pixels



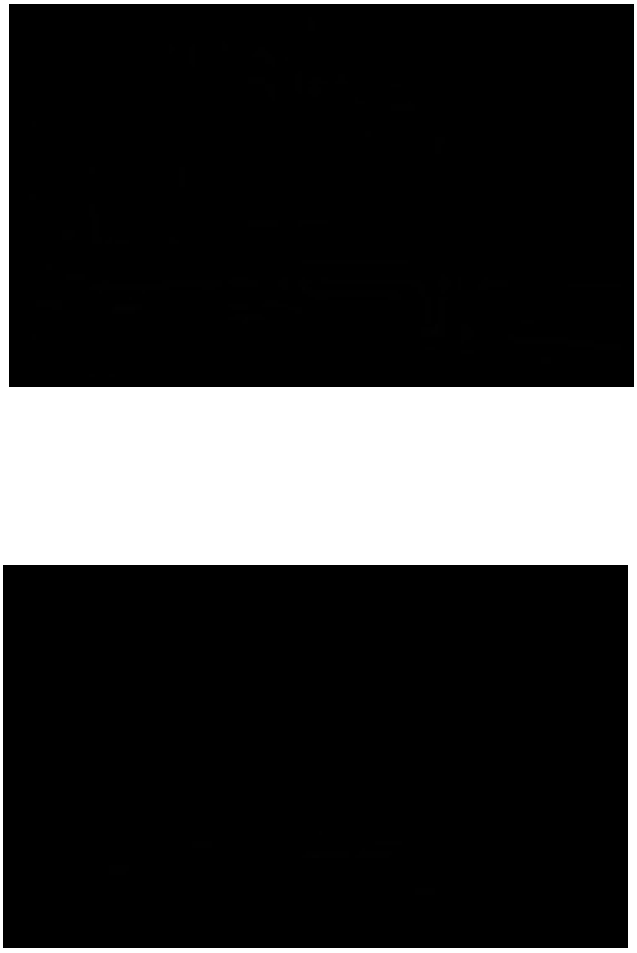


Figure 8: DoGs in the third octave

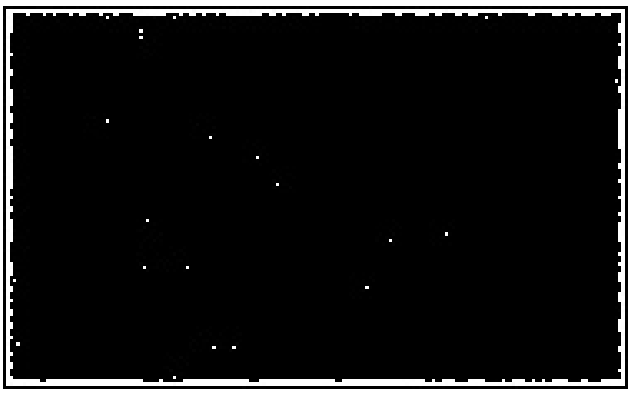


Figure 9: Maxima & minima of DoGs in the third octave

Figure 10: Keypoints marked as white dots

3 Task 3 : Cursor Detection

For the given problem, I have used Normalized Cross Correlation(NCC) for template matching & Canny Edge Detector to detect the edges in the preprocessing part. Detailed explanation of the proposed method is given in the code using comments.

Source Code:

```
image = cv2.imread("pos_3.jpg",0)
template = cv2.imread("template.png",0)

""" edge detection in the template using canny detector """
template = cv2.Canny(template, 150, 255)
(tH, tW) = template.shape[:2]
print(image.shape)
print(template.shape)
#cv2.imshow("image_initial",image)
#cv2.imshow("template",template)


pixels=[[0,0,0],[0,0,0]]

for scale in np.linspace(0.1, 3.0,200)[::-1]:
    """
    we will resize the image according to the scale and
    then keep track of the ratio of the resizing so that later on we can
    multiply that ratio with the founded coordinates and get back the
    position in the original image to draw rectangle.

    Also, we resize the image because cursor in the image & cursor template
    may not have the same dimensions. So, to make sure that they overlap well
    on each other we try to cover range of scale values to resize the image.

    """

    resized = imutils.resize(image, width = int(image.shape[1] * scale))
    r = image.shape[1] / float(resized.shape[1])
    """
    if the resized image is smaller than the template, then we break
    from the loop

    """
    if resized.shape[0] < tH or resized.shape[1] < tW:
        break
```

```
    """
    once the image is resized, we try to detect edges in both template and
    the image because in template matching, images with their edges detected
    performs better and gives good results. There are many edge detectors
    available. I have used canny edge detector.

    """
    edged = cv2.Canny(resized, 150, 255)
    result = cv2.matchTemplate(edged, template, cv2.TM_CCORR_NORMED)
    (_, maxVal, _, maxLoc) = cv2.minMaxLoc(result)
    #print(maxVal[0])
    """
    if we have found a new maximum correlation value, then we will update
    the bookkeeping variable.

    """

    """
    Here, there is an issue of false cursoer being detected on the side bar.
    So, I am trying to find two different co-ordinates where the correlation value
    is the highest and  the second highest because when there are two
    cursors in an image, difference between the correlation value at
    both coordinates(cursors) will be minute.

    """
    maxima=[maxVal,maxLoc,r]
    if(pixels[0][0]>pixels[1][0]):
        mini = pixels[1]
    else:
        mini = pixels[0]

    if(maxima[0]>mini[0]):
        item=pixels.index(mini)
        del pixels[item]
        pixels.insert(0,maxima)

print(pixels)

for i in range(len(pixels)):
    (_, maxLoc, r) = tuple(pixels[i])

    """
    we are determining the coordinates where the correlation value is highest in
    the resized image. Then, using resize ratio we will determine the corresponding
    coordinates in the original image.
```

```
    """
    (startX, startY) = (int(maxLoc[0] * r), int(maxLoc[1] * r))

    """
    After that we are using template's dimensions to know the dimensions of the
    rectangle we will be drawing to
    depict the detected cursor.

    """
    (endX, endY) = (int((maxLoc[0] + tW) * r), int((maxLoc[1] + tH) * r))

    # draw a bounding box around the detected result and display the image
    cv2.rectangle(image, (startX, startY), (endX, endY), (0, 0, 255), 2)


#cv2.imshow("temp",template)
#cv2.imshow("edged",edged)
cv2.imshow("Image", image)
cv2.waitKey(100000)
cv2.destroyAllWindows()
```