CSE-573
PROJECT-1

Apoorva Biseria
UB PERSON NUMBER- 50291145

1. Edge Detection

Write programs to detect edges in Fig. 1 (along both x and y directions) using Sobel operator. In your report, please include two resulting images, one showing edges along x direction and the other showing edges along y direction.

Program 1.

```
import cv2
import numpy as np
#read image
img = cv2.imread('task1.png',0)
k,l = (img.shape)
#sobel for horizontal direction
kernel = ([-1,0,1],[-2,0,2],[-1,0,1])
m = k+3
n = l+3
matrix = [[0 for j in range(n)] for i in range(m)]
result = [[0 for x in range(n)] for  w in range(m)]
result2 = [[0 for x in range(n)] for  w in range(m)]
i,j=0,0
#padding of image
for i in range(k):
    for j in range(l):
        matrix[i+1][j+1] = img[i][j]

sum1,y= 0,0
for z in range(k):
    for y in range(l):
        sum1=0
        for i in range(3):
            for j in range(3):
                sum1 += matrix[z+i][y+j]*kernel[i][j]

        result[z][y] = sum1
#finding the absolute value
for i in range(k):
    for j in range(l):
        if(result[i][j]<0):
            result[i][j] = result[i][j] * (-1)
maxim = 0
for i in range(k):
    for j in range(l):
```

```
        if (result[i][j]>maxim):
            maxim = result[i][j]

#elimination of 0's
for i in range(k):
    for j in range(l):
        result[i][j] = result[i][j]/maxim



n1 = np.asarray(result)
cv2.imshow('image',n1)
cv2.waitKey(0)
nv = n1 *255
cv2.imwrite('horizontal.jpeg',nv)
#sobel operator for x direction
kernel2 = ([-1,-5,-1],[0,0,0],[1,5,1])
sum2,y= 0,0
for z in range(k):
    for y in range(l):
        sum2=0
        for i in range(3):
            for j in range(3):
                sum2 += matrix[z+i][y+j]*kernel2[i][j]

        result2[z][y] = sum2
minim = 255
for i in range(k):
    for j in range(l):
        if (result2[i][j]<minim):
            minim = result2[i][j]

for i in range(k):
    for j in range(l):
        if(result2[i][j]<0):
            result2[i][j] = result2[i][j] * (-1)
maxim = 0
for i in range(k):
    for j in range(l):
        if (result2[i][j]>maxim):
            maxim = result2[i][j]

for i in range(k):
    for j in range(l):
         result2[i][j] = result2[i][j]/maxim



n2 = np.asarray(result2)
cv2.imshow('image',n2)
cv2.waitKey(0)
nh = n2 *255
cv2.imwrite('vertical.jpeg',nh)
#showing the image along both x and y direction edges
```

```
n3 = (n1**2 + n2**2)**(1/2)
cv2.imshow('image',n3)
cv2.waitKey(0)
nc = n3 *255
cv2.imwrite('combined.jpeg',nc)
```

figure 1.1
Along X direction

figure- 1.2
Along y axis

figure- 1.3
Along both x and y axis

2. Keypoint Detection

Program 1 – 1st octave

```
import cv2

import numpy as np
img = cv2.imread('task2.jpg',0)

k,l = img.shape

m = k+6
n = l+6
matrix = [[0 for j in range(n)] for i in range(m)]
result1 = [[0 for j in range(n)] for i in range(m)]
main_kernel = [[0 for j in range(7)] for i in range(7)]
main_part = [[0 for j in range(n)] for i in range(m)]
main_part2 = [[0 for j in range(n)] for i in range(m)]
final =[[0 for j in range(n)] for i in range(m)]

#padding the image
i,j=0,0
for i in range(k):
    for j in range(l):
        matrix[i+3][j+3] = img[i][j]

i,j=0,0
for i in range(k):
    for j in range(l):
        final[i][j] = img[i][j]


#defining the kernel
def Kern( k):

    import numpy as np
    kernel2 = [[0 for j in range(4)] for i in range(4)]
    from math import e,pi
    i=0
    j=0
    s= k
    for i in range(4):
        for j in range(4):

            kernel2[i][j] = float(1/(2*pi*s*s))*e**(-0.5*float((float(i*i) + float(j*j))/(float(s*s))))

    i,j = 1,1
    g = float(1/(2*pi*s*s))*e**(-0.5*float((float(i*i*s*s) + float(j*j*s*s))/(float(s*s))))

    print("G value",g)
    sun =0
    summation =0
```

```
    for i in range(4):
        for j in range(4):
            main_kernel[i][j] = kernel2[3-i][3-j]
        #summation += kernel2[3-i][3-j]

    for i in range(3):
        for j in range(3):
            main_kernel[i][j+4] = kernel2[3-i][j+1]
        #summation += kernel2[3-i][j+1]

    for i in range(3):
        for j in range(3):
            main_kernel[i][j+4] = kernel2[3-i][j+1]
        #summation += kernel2[3-i][j+1]

    for i in range(3):
        for j in range(3):
            main_kernel[i+4][j] =kernel2[i+1][3-j]
        #summation += kernel2[i+1][3-j]

    for i in range(4):
        for j in range(4):
            main_kernel[i+3][j+3] = kernel2[i][j]

    kernel = np.asarray(main_kernel)
    np.set_printoptions(suppress=True)
    print(kernel)

    #funct = float(1/summation)
    #print(funct)
    for i in range(7):
        for j in range(7):
            summation += kernel[i][j]

    funct = float(1/summation)
    for i in range(7):
        for j in range(7):
            kernel[i][j] = funct * kernel[i][j]

    print(summation)
    summ =0
    for i in range(7):
        for j in range(7):
            summ += kernel[i][j]
    print("After",summ)

    return kernel


# In[2]:

#calculating different kernels for sigma values
sum1,y= 0,0
kernel = Kern(0.707)
for z in range(k):
    for y in range(l):
        sum1=0
        for i in range(7):
            for j in range(7):
                sum1 += matrix[z+i][y+j]*kernel[i][j]
```

```
        result1[z][y] = sum1


n1 = np.asarray(result1)
maxim = 0
for i in range(m):
    for j in range(n):
        if (result1[i][j]>maxim):
            maxim = result1[i][j]
rm =[[0 for j in range(l)] for i in range(k)]
for i in range(k):
    for j in range(l):
        rm[i][j] =result1[i][j]


rm = rm/maxim
rm = np.asarray(rm)
cv2.imshow('1stbluroct1',rm)
cv2.waitKey(3000)
resultant = rm*255
cv2.imwrite('1stbluroct1.jpeg',resultant)

result2 = [[0 for j in range(n)] for i in range(m)]
kernel = Kern(1)
sum1,y= 0,0
for z in range(k):
    for y in range(l):
        sum1=0
        for i in range(7):
            for j in range(7):
                sum1 += matrix[z+i][y+j]*kernel[i][j]

        result2[z][y] = sum1

n2 = np.asarray(result2)
maxim = 0
for i in range(k):
    for j in range(l):
        if (result2[i][j]>maxim):
            maxim = result2[i][j]
rm =[[0 for j in range(l)] for i in range(k)]
for i in range(k):
    for j in range(l):
        rm[i][j] =result2[i][j]


rm = rm/maxim
rm = np.asarray(rm)
cv2.imshow('2ndbluroct1',rm)
cv2.waitKey(3000)
resultant = rm*255
cv2.imwrite('2ndbluroct1.jpeg',resultant)

result3 = [[0 for j in range(n)] for i in range(m)]
kernel =Kern(1.414)
sum1,y= 0,0
for z in range(k):
    for y in range(l):
        sum1=0
```

```
        for i in range(7):
            for j in range(7):
                sum1 += matrix[z+i][y+j]*kernel[i][j]

        result3[z][y] = sum1

n3 = np.asarray(result3)
maxim = 0
for i in range(k):
    for j in range(l):
        if (result3[i][j]>maxim):
            maxim = result3[i][j]

rm =[[0 for j in range(l)] for i in range(k)]
for i in range(k):
    for j in range(l):
        rm[i][j] =result3[i][j]

rm = rm/maxim
rm = np.asarray(rm)
cv2.imshow('3rdbluroct1',rm)
cv2.waitKey(3000)
resultant = rm*255
cv2.imwrite('3rdbluroct1.jpeg',resultant)

result4 = [[0 for j in range(n)] for i in range(m)]
kernel =Kern(2)
sum1,y= 0,0
for z in range(k):
    for y in range(l):
        sum1=0
        for i in range(7):
            for j in range(7):
                sum1 += matrix[z+i][y+j]*kernel[i][j]

        result4[z][y] = sum1

n4 = np.asarray(result4)
maxim = 0
for i in range(k):
    for j in range(l):
        if (result4[i][j]>maxim):
            maxim = result4[i][j]
rm =[[0 for j in range(l)] for i in range(k)]
for i in range(k):
    for j in range(l):
        rm[i][j] =result4[i][j]


rm = rm/maxim
rm = np.asarray(rm)
cv2.imshow('4thbluroct1',rm)
cv2.waitKey(3000)
resultant = rm*255
cv2.imwrite('4thbluroct1.jpeg',resultant)

result5 = [[0 for j in range(n)] for i in range(m)]
kernel = Kern(2.828)
sum1,y= 0,0
for z in range(k):
```

```
    for y in range(l):
        sum1=0
        for i in range(7):
            for j in range(7):
                sum1 += matrix[z+i][y+j]*kernel[i][j]

        result5[z][y] = sum1

n5 = np.asarray(result5)
maxim = 0
for i in range(k):
    for j in range(l):
        if (result5[i][j]>maxim):
            maxim = result5[i][j]
rm =[[0 for j in range(l)] for i in range(k)]
for i in range(k):
    for j in range(l):
        rm[i][j] =result5[i][j]


rm = rm/maxim
rm = np.asarray(rm)
cv2.imshow('5thbluroct1',rm)
cv2.waitKey(3000)
resultant = rm*255
cv2.imwrite('5thbluroct1.jpeg',resultant)

#calculating different dog

dog1 = [[0 for j in range(n)] for i in range(m)]
dog2 = [[0 for j in range(n)] for i in range(m)]
dog3 = [[0 for j in range(n)] for i in range(m)]
dog4 = [[0 for j in range(n)] for i in range(m)]

dog1 = n1 - n2
dog2 = n2 - n3
dog3 = n3 - n4
dog4 = n4- n5




# In[3]:


nk1 = np.asarray(dog1)
cv2.imshow('image',nk1)
cv2.waitKey(3000)
nk1 = nk1 * 255
cv2.imwrite('dog1oct1.jpeg',nk1)
nk2 = np.asarray(dog2)
cv2.imshow('image',nk2)
cv2.waitKey(3000)
nk2 = nk2 * 255
cv2.imwrite('dog2oct1.jpeg',nk2)
nk3 = np.asarray(dog3)
cv2.imshow('imageoct1',nk3)
cv2.waitKey(3000)
nk3 = nk3 * 255
```

```
cv2.imwrite('dog3oct1.jpeg',nk3)
nk4 = np.asarray(dog4)
cv2.imshow('image',nk4)
cv2.waitKey(3000)
nk4 = nk4 * 255
cv2.imwrite('dog4oct1.jpeg',nk4)



count = 0

i = 3
j = 3
for i in range(m):
    for j in range(n):
        t = 0
        if(dog2[i][j]>dog2[i-1][j-1]):
            if(dog2[i][j]>dog2[i-1][j]):
                if(dog2[i][j]>dog2[i-1][j+1]):
                    if(dog2[i][j]>dog2[i][j-1]):
                        if(dog2[i][j]>dog2[i][j+1]):
                            if(dog2[i][j]>dog2[i+1][j+1]):
                                if(dog2[i][j]>dog2[i+1][j]):
                                    if(dog2[i][j]>dog2[i+1][j-1]):
                                        if(dog2[i][j]>dog1[i-1][j-1]):
                                            if(dog2[i][j]>dog1[i-1][j]):
                                                if(dog2[i][j]>dog1[i-1][j+1]):
                                                    if(dog2[i][j]>dog1[i][j+1]):
                                                        if(dog2[i][j]>dog1[i][j]):
                                                            if(dog2[i][j]>dog1[i][j-1]):
                                                                if(dog2[i][j]>dog1[i+1][j+1]):
                                                                    if(dog2[i][j]>dog1[i+1][j-1]):
                                                                        if(dog2[i][j]>dog1[i+1][j]):
                                                                            if(dog2[i][j]>dog3[i-1][j]):
                                                                                if(dog2[i][j]>dog3[i-1][j+1]):
                                                                                    if(dog2[i][j]>dog3[i-1][j-1]):
                                                                                        if(dog2[i][j]>dog3[i][j+1]):
                                                                                            if(dog2[i][j]>dog3[i][j]):
                                                                                                if(dog2[i][j]>dog3[i][j-1]):
                                                                                                    if(dog2[i][j]>dog3[i+1][j+1]):
                                                                                                        if(dog2[i][j]>dog3[i+1][j]):
                                                                                                            if(dog2[i][j]>dog3[i+1][j-1]):
                                                                                                                t = 1







        if(dog2[i][j]<dog2[i-1][j-1]):
            if(dog2[i][j]<dog2[i-1][j]):
                if(dog2[i][j]<dog2[i-1][j+1]):
                    if(dog2[i][j]<dog2[i][j-1]):
                        if(dog2[i][j]<dog2[i][j+1]):
                            if(dog2[i][j]<dog2[i+1][j+1]):
                                if(dog2[i][j]<dog2[i+1][j]):
                                    if(dog2[i][j]<dog2[i+1][j-1]):
                                        if(dog2[i][j]<dog1[i-1][j-1]):
                                            if(dog2[i][j]<dog1[i-1][j]):
```

```
                                    if(dog2[i][j]<dog1[i-1][j+1]):
                                        if(dog2[i][j]<dog1[i][j+1]):
                                            if(dog2[i][j]<dog1[i][j]):
                                                if(dog2[i][j]<dog1[i][j-1]):
                                                    if(dog2[i][j]<dog1[i+1][j+1]):
                                                        if(dog2[i][j]<dog1[i+1][j-1]):
                                                            if(dog2[i][j]<dog1[i+1][j]):
                                                                if(dog2[i][j]<dog3[i-1][j]):
                                                                    if(dog2[i][j]<dog3[i-1][j+1]):
                                                                        if(dog2[i][j]<dog3[i-1][j-1]):
                                                                            if(dog2[i][j]<dog3[i][j+1]):
                                                                                if(dog2[i][j]<dog3[i][j]):
                                                                                    if(dog2[i][j]<dog3[i][j-1]):
                                                                                        if(dog2[i][j]<dog3[i+1][j+1]):
                                                                                            if(dog2[i][j]<dog3[i+1][j]):
                                                                                                if(dog2[i][j]<dog3[i+1][j-1]):
                                                                                                    t = 1




        if(t == 1 ):
                main_part[i][j] = t


count = 0
for i in range(n):
    for j in range(m):
        if(main_part[j][i]==1 and count<5):
            count = count +1
            print(j,i)






z = np.asarray(main_part)
for x in range(m):
    for y in range(n):
        if(z[x][y]==2):
            z[x][y] = 0
        if(z[x][y] ==1):
            z[x][y] = 255


maxim = 0
for i in range(m):
    for j in range(n):
        if (z[i][j]>maxim):
            maxim = z[i][j]



z = z/maxim

cv2.imshow('keypoint2aoct1',z)
cv2.waitKey(3000)
z = z *255
```

```
cv2.imwrite('keypoint2aoct1.jpeg',z)



# In[4]:


count = 0

i = 3
j = 3
for i in range(m):
    for j in range(n):
        t = 0
        if(dog3[i][j]>dog3[i-1][j-1]):
            if(dog3[i][j]>dog3[i-1][j]):
                if(dog3[i][j]>dog3[i-1][j+1]):
                    if(dog3[i][j]>dog3[i][j-1]):
                        if(dog3[i][j]>dog3[i][j+1]):
                            if(dog3[i][j]>dog3[i+1][j+1]):
                                if(dog3[i][j]>dog3[i+1][j]):
                                    if(dog3[i][j]>dog3[i+1][j-1]):
                                        if(dog3[i][j]>dog2[i-1][j-1]):
                                            if(dog3[i][j]>dog2[i-1][j]):
                                                if(dog3[i][j]>dog2[i-1][j+1]):
                                                    if(dog3[i][j]>dog2[i][j+1]):
                                                        if(dog3[i][j]>dog2[i][j]):
                                                            if(dog3[i][j]>dog2[i][j-1]):
                                                                if(dog3[i][j]>dog2[i+1][j+1]):
                                                                    if(dog3[i][j]>dog2[i+1][j-1]):
                                                                        if(dog3[i][j]>dog2[i+1][j]):
                                                                            if(dog3[i][j]>dog4[i-1][j]):
                                                                                if(dog3[i][j]>dog4[i-1][j+1]):
                                                                                    if(dog3[i][j]>dog4[i-1][j-1]):
                                                                                        if(dog3[i][j]>dog4[i][j+1]):
                                                                                            if(dog3[i][j]>dog4[i][j]):
                                                                                                if(dog3[i][j]>dog4[i][j-1]):
                                                                                                    if(dog3[i][j]>dog4[i+1][j+1]):
                                                                                                        if(dog3[i][j]>dog4[i+1][j]):
                                                                                                            if(dog3[i][j]>dog4[i+1][j-1]):
                                                                                                                t = 1



        if(dog2[i][j]<dog2[i-1][j-1]):
            if(dog2[i][j]<dog2[i-1][j]):
                if(dog2[i][j]<dog2[i-1][j+1]):
                    if(dog2[i][j]<dog2[i][j-1]):
                        if(dog2[i][j]<dog2[i][j+1]):
                            if(dog2[i][j]<dog2[i+1][j+1]):
                                if(dog2[i][j]<dog2[i+1][j]):
                                    if(dog2[i][j]<dog2[i+1][j-1]):
                                        if(dog2[i][j]<dog1[i-1][j-1]):
                                            if(dog2[i][j]<dog1[i-1][j]):
                                                if(dog2[i][j]<dog1[i-1][j+1]):
                                                    if(dog2[i][j]<dog1[i][j+1]):
                                                        if(dog2[i][j]<dog1[i][j]):
                                                            if(dog2[i][j]<dog1[i][j-1]):
                                                                if(dog2[i][j]<dog1[i+1][j+1]):
                                                                    if(dog2[i][j]<dog1[i+1][j-1]):
```

```
                                        if(dog2[i][j]<dog1[i+1][j]):
                                            if(dog2[i][j]<dog3[i-1][j]):
                                                if(dog2[i][j]<dog3[i-1][j+1]):
                                                    if(dog2[i][j]<dog3[i-1][j-1]):
                                                        if(dog2[i][j]<dog3[i][j+1]):
                                                            if(dog2[i][j]<dog3[i][j]):
                                                                if(dog2[i][j]<dog3[i][j-1]):
                                                                    if(dog2[i][j]<dog3[i+1][j+1]):
                                                                        if(dog2[i][j]<dog3[i+1][j]):
                                                                            if(dog2[i][j]<dog3[i+1][j-1]):
                                                                                t = 1




        if(t == 1 ):
                main_part2[i][j] = t

count = 0
for i in range(n):
    for j in range(m):
        if(main_part[j][i]==1 and count<5):
            count = count +1
            print(j,i)


z = np.asarray(main_part2)
for x in range(m):
    for y in range(n):
        if(z[x][y]==2):
            z[x][y] = 0
        if(z[x][y] ==1):
            z[x][y] = 255


maxim = 0
for i in range(m):
    for j in range(n):
        if (z[i][j]>maxim):
            maxim = z[i][j]



z = z/maxim

cv2.imshow('keypint2boct1',z)
cv2.waitKey(3000)
z = z*255
cv2.imwrite('keypoint2boct1.jpeg',z)


for i in range(k):
            for j in range(l):
                        if(main_part[i+3][j+3]==1 or main_part2[i+3][j+3]==1):
                                    img[i][j]= 255
```

```
final = np.asarray(img)
cv2.imshow('final',final)
cv2.waitKey(0)
cv2.imwrite("finalimage.jpeg",final)
```

figure 2.1
1stblur octave 1

figure 2.2
2nd blur octave 1

figure 2.3
3rd blur octave 1

figure 2.4
4th blur octave 1

figure 2.5
5th blur octave 1

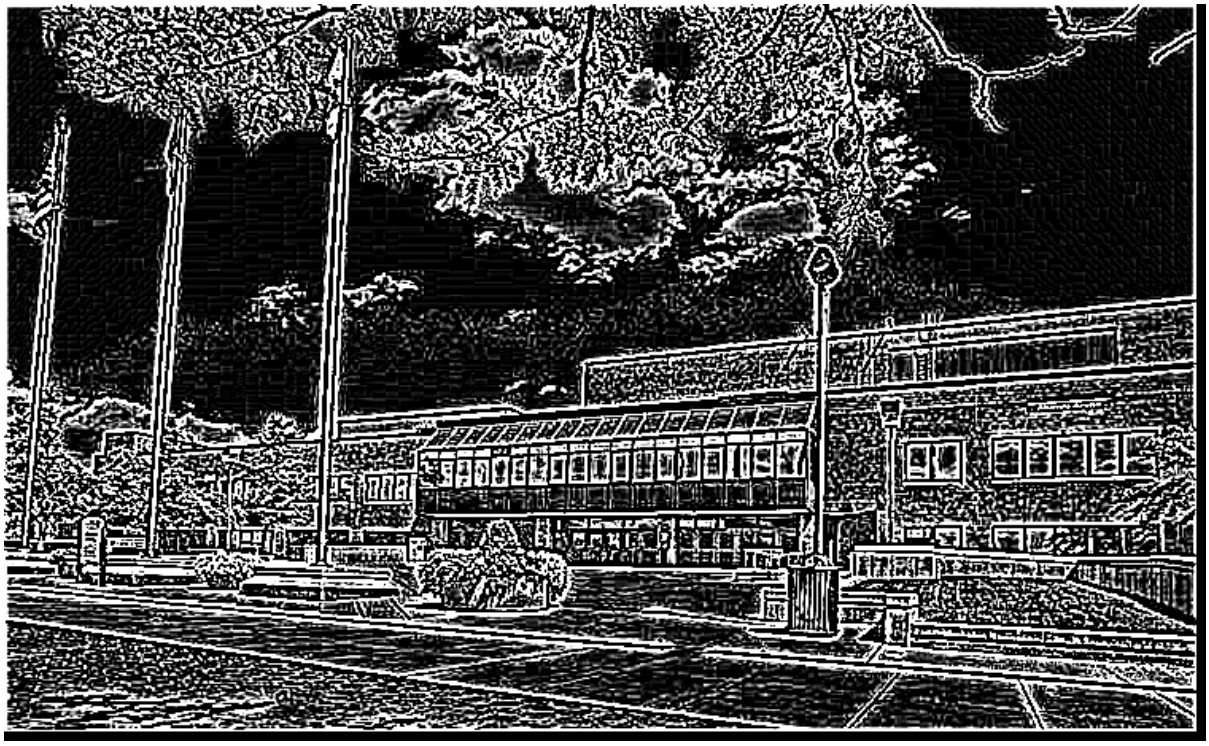


figure 2.6
dog 1 octave 1

figure 2.7
dog 2 octave 1

figure 2.8
dog 3 octave 1

figure 2.9
dog4 octave 1

figure 2.10
first keypoint detection between dog1 dog2 and dog3

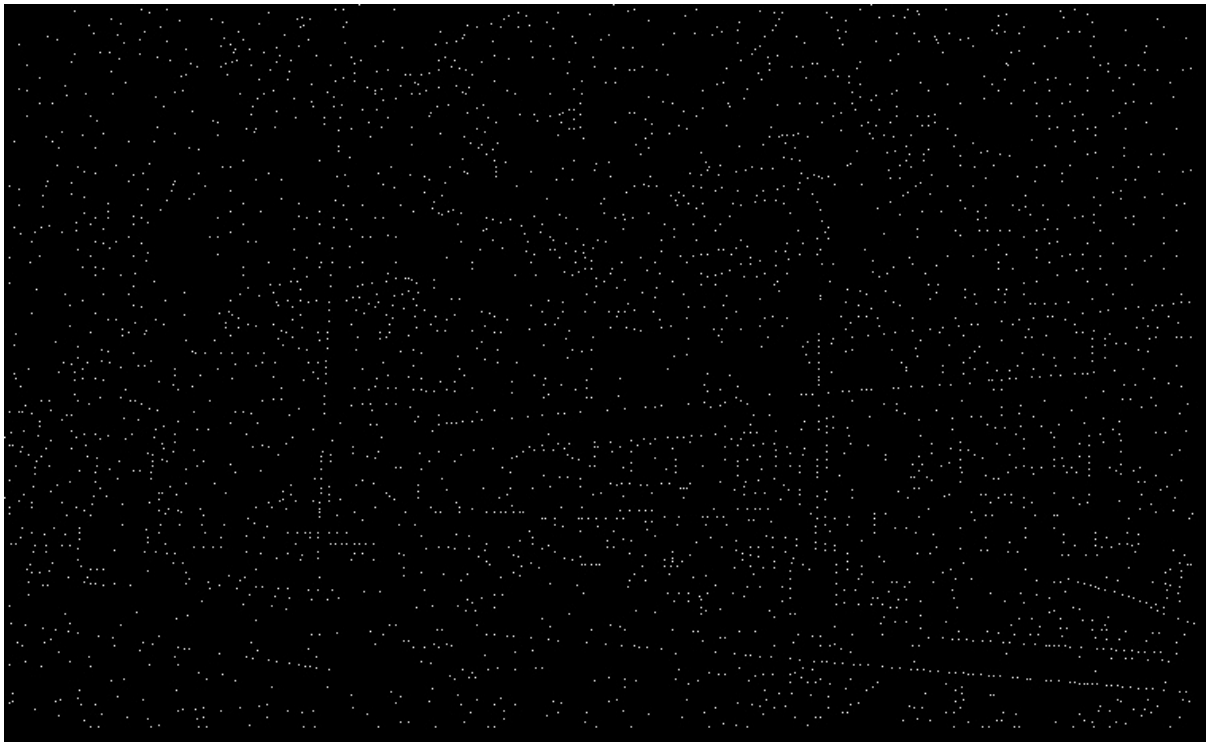


figure 2.11
second keypoint detection between dog2 dog3 and dog4

figure 2.12
keypoints on the original image

Program 2.2 for 2nd octave
I showed the resize part of the program, other code is same as octave 1 except for kernel values. Inclusion of full code made the report of more than 50 pages.

```
# coding: utf-8

# In[3]:


import cv2
import numpy as np
img = cv2.imread('task2.jpg',0)
p,q = img.shape
m = int(p/2)+1
n = int(q/2)+1
small_img = [[0 for j in range(n)] for i in range(m)]


# In[4]:


m =0
n=0
for i in range(0,p,2):
    n=0
    for j in range(0,q,2):
        small_img[m][n] = img[i][j]
        n = n+1
    m = m + 1
```

```
# In[5]:


newimage = np.asarray(small_img)



# In[6]:


cv2.imwrite('octave2.jpeg',newimage)


# In[7]:


octave2 = cv2.imread('octave2.jpeg',0)
k,l = octave2.shape
print(k,l)
octave2process = cv2.imread('octave2.jpeg',0)

# In[8]:
```

figure 2.13 1st blur octave 2

figure 2.14
2nd blur octave 2

figure 2.15
3rd blur octave 2

figure 2.15
4th blur octave 2

figure 2.16
5th blur octave 2

figure 2.16
dog1 octave 2

figure 2.17
dog 2 octave 2

figure 2.18
dog 3 octave 2

figure 2.19
dog 4 octave 2

figure 2.20
keypoint by dog1 dog2 and dog3

figure 2.21
keypoint by dog2 dog3 and dog4

figure 2.22
final keypoint on original image

Program 2.3 for 3rd octave

```
import cv2
import numpy as np
img = cv2.imread('octave2.jpeg',0)
p,q = img.shape
m = int(p/2)+1
n = int(q/2)+1
small_img = [[0 for j in range(n)] for i in range(m)]
m =0
n=0
for i in range(0,p,2):
    n=0
    for j in range(0,q,2):
        small_img[m][n] = img[i][j]
        n = n+1
    m = m + 1


newimage = np.asarray(small_img)
im4 = np.array(np.clip(newimage, 0, 255), dtype=np.uint8)
cv2.imwrite('octave3.jpeg',newimage)

octave3 = cv2.imread('octave3.jpeg',0)
k,l = octave3.shape
octave3process = cv2.imread('octave3.jpeg',0)
```

figure 2.23
1st blur octave 3

figure 2.24
2nd blur octave 3

figure 2.25
3rd blur octave 3

figure 2.26
4th blur octave 3

figure2.27
5th blur octave 3

figure 2.28
1st dog octave 3

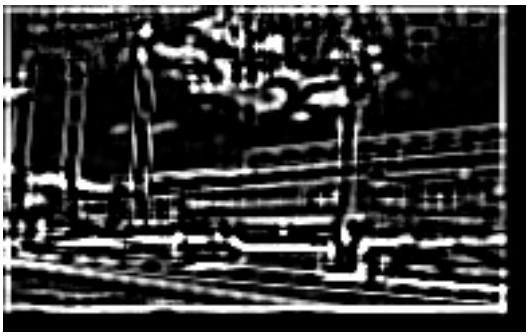


figure 2.29
2nd dog octave 3

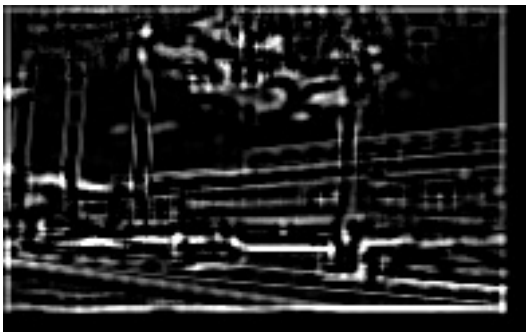


figure 2.30
3rd dog octave 3

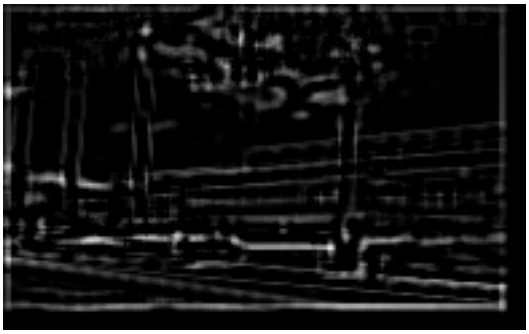


figure 2.31
4th dog octave 3

figure 2.32
keypoint bw dog1 dog2 dog3

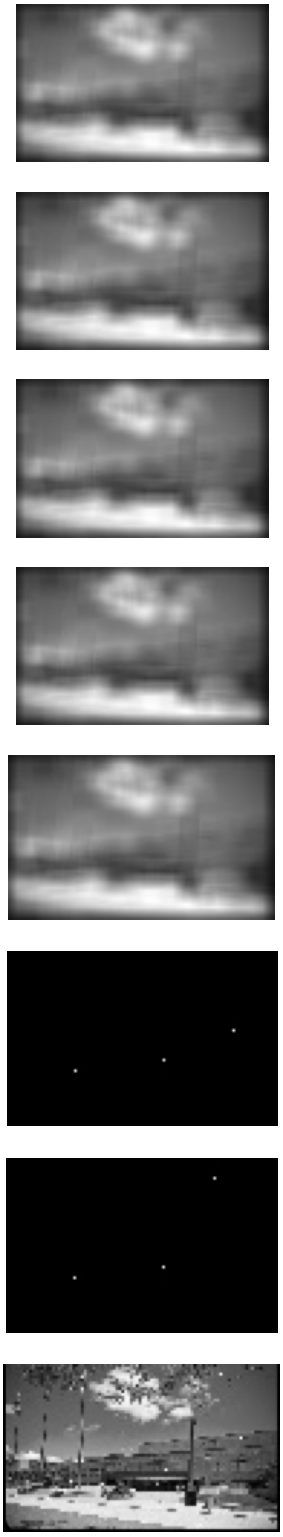
figure 2.33
keypoint bw dog 2 dog 3 and dog4

Program 2.3 for 4th octave

```
import cv2
import numpy as np
img = cv2.imread('octave3.jpeg',0)
p,q = img.shape
m = int(p/2)+1
n = int(q/2)+1
small_img = [[0 for j in range(n)] for i in range(m)]

m =0
n=0
for i in range(0,p,2):
    n=0
    for j in range(0,q,2):
        small_img[m][n] = img[i][j]
        n = n+1
    m = m + 1
```

newimage = np.asarray(small_img)



1) Resolution of 2nd octave (width *height) = 376 * 230
    a. Resolution of 3rd octave (width * height) = 189 *116
    b. Images included above

2) Images included above

3) Images included above

4) For 1st octaveKeypoints – (271,0), (310, 0), (1 ,1), (231,1), (266 ,1)
For 2nd octave keypoints- (57,0), (111,5), (103,7), (42,12), (22,13)
For 3rd octave keypoints – (42,16), (0,18), (32,33), (7,41), (76,42)
For 4th octave keypoiints – (44,25), (40,58), (29,84)

3. Cursor detection

```
import numpy as np
import imutils
import cv2
# taken from the imutils package for python
def resize(image, width = None, height = None, inter = cv2.INTER_AREA):
	# initialize the dimensions of the image to be resized and
	# grab the image size
	dim = None
	(h, w) = image.shape[:2]

	# if both the width and height are None, then return the
	# original image
	if width is None and height is None:
		return image

	# check to see if the width is None
	if width is None:
		# calculate the ratio of the height and construct the
		# dimensions
		r = height / float(h)
		dim = (int(w * r), height)

	# otherwise, the height is None
	else:
		# calculate the ratio of the width and construct the
		# dimensions
		r = width / float(w)
		dim = (width, int(h * r))

	# resize the image
	resized = cv2.resize(image, dim, interpolation = inter)

	# return the resized image
	return resized

# Performing the gaussian blur on the image
img= cv2.imread('pos_1.jpg')
blur = cv2.GaussianBlur(img,(3,3),0)
image= cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)


template = cv2.imread("template.png")
templategray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
#Performing the laplacian on image and template
image = np.asarray(image)
imagelap = cv2.Laplacian(image,cv2.CV_8U)
```

```
templatelap = cv2.Laplacian(templategray,cv2.CV_8U)

resultant = None
for i in np.linspace(0.1,0.6,60):
        small = resize(templatelap, width = int(templatelap.shape[1] * i))
        print(small.shape)
        result = cv2.matchTemplate(imagelap, small, cv2.TM_CCOEFF)
        minvalue, maxvalue, minloc, maxloc = cv2.minMaxLoc(result)
        print(maxvalue)
        if resultant is None or maxvalue > resultant[0]:
                resultant = (maxvalue, maxloc)

(maxvalue, maxloc )= resultant

(x, y) = (int(maxloc[0] ), int(maxloc[1] ))
(x2, y2)= (int((maxloc[0] + 30)), int((maxloc[1] + 30) ))
if(maxvalue >140000 or (maxvalue <130000 and maxvalue > 100000)):
        cv2.rectangle(img, (x-10, y-10), (x2, y2), (0, 0, 255), 2)
cv2.imshow("Image", img)
cv2.waitKey(0)
```

Other approaches I tired for template matching

1. I tried to implement by using the canny edge detector with resizing the image, but this method did not have sufficient accuracy.
2. Here I tried appling Gaussian blur on the image to reduce noise and then laplacian transformation on both the image and template further resizing the template to get a better result.