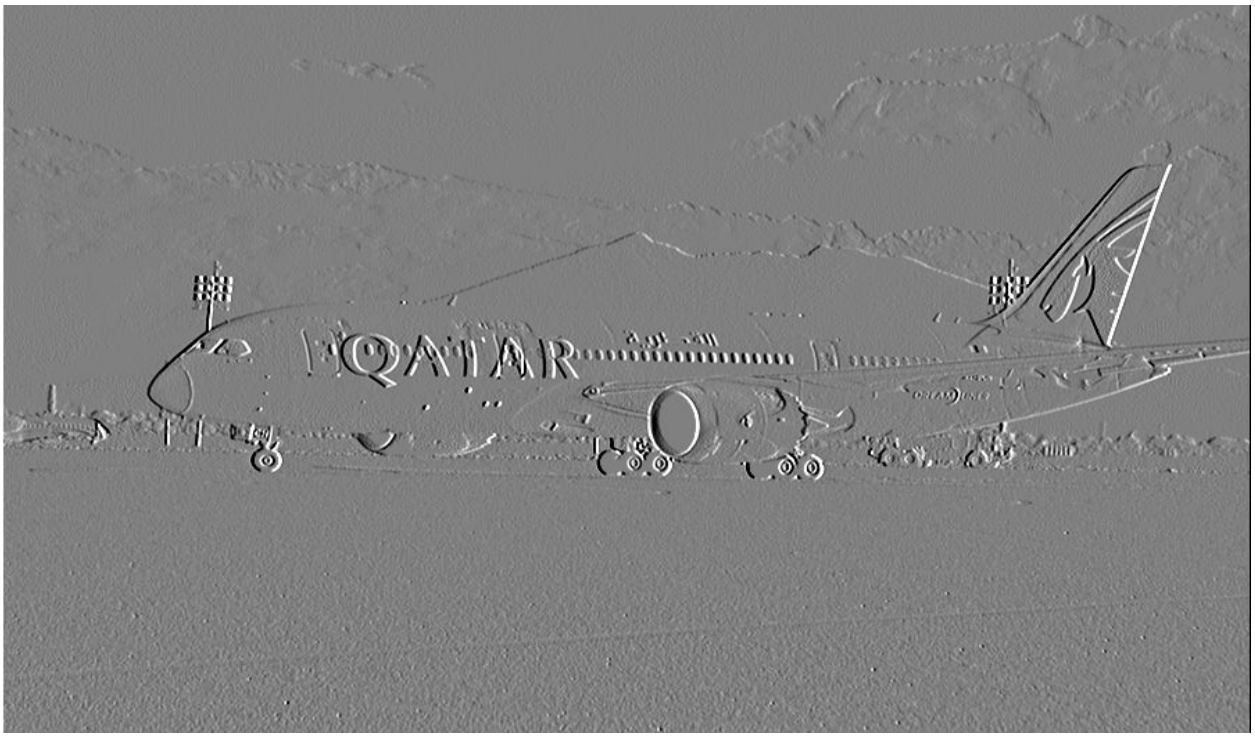
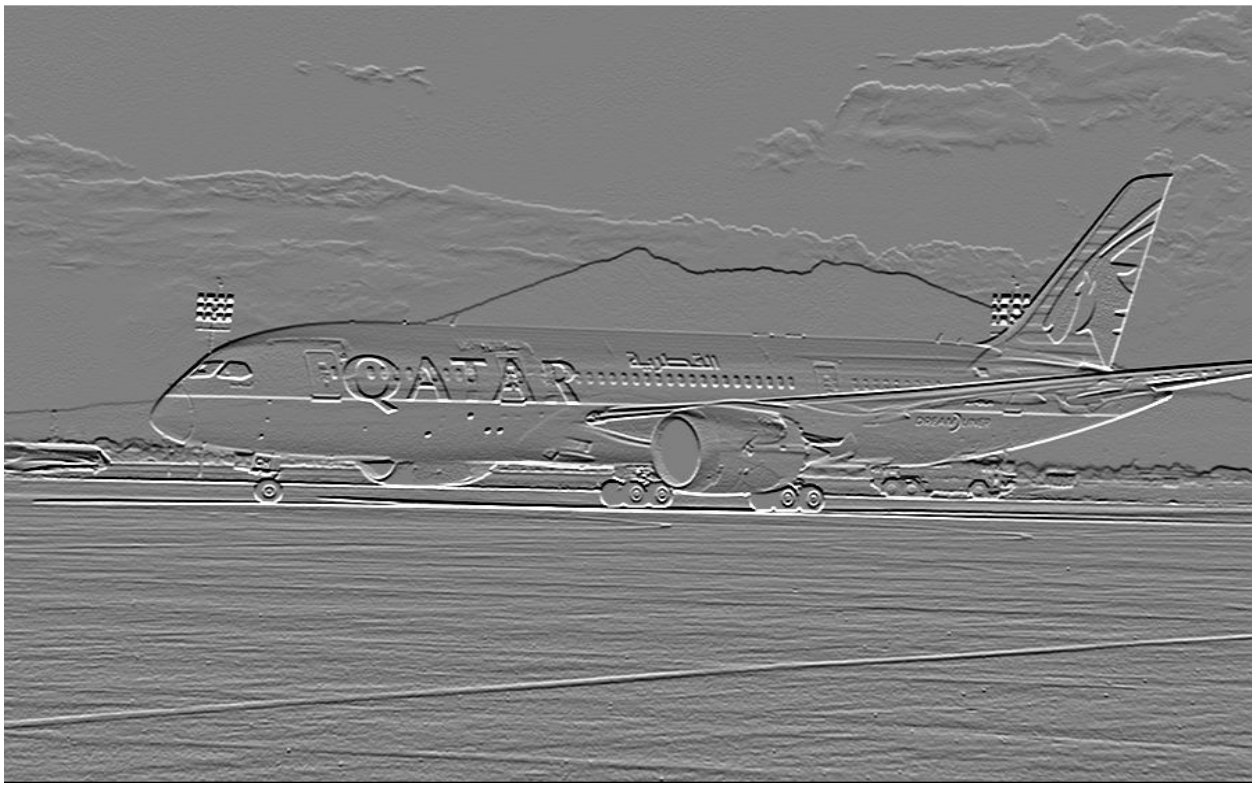
TASK I: EDGE DETECTION



Gradient X



Gradient Y

Source Code for finding Gradient X using Sobel Operator:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("/Users/stutishukla/Downloads/proj1_cse573/task1.png", cv2.IMREAD_GRAYSCALE)
a = np.asarray(img)
sobelW, sobelH = 3, 3
sobelData = [[0 for x in range(sobelW)] for y in range(sobelH)]
sobelData[0][0] = -1
sobelData[0][1] = 0
sobelData[0][2] = 1
sobelData[1][0] = -2
sobelData[1][1] = 0
sobelData[1][2] = 2
sobelData[2][0] = -1
sobelData[2][1] = 0
sobelData[2][2] = 1

imageH=len(img)
imageW=len(img[0])

updatedImageData = [[0 for x in range(imageW)] for y in range(imageH)]

def calculateSobelat(widthindex , heightindex): #applying sobel operator on every pixel of the matrix
    result = 0;
     for i in range(sobelH):
        for j in range(sobelW):
            currentWidthIndex = widthindex-1+j
            currentHeightIndex = heightindex-1+i
                if((currentHeightIndex<0) or (currentHeightIndex >= imageH)):
                      continue
                 if ((currentWidthIndex < 0) or (currentWidthIndex >= imageW)):
                       continue
                 result += (a[currentHeightIndex][currentWidthIndex]*sobelData[i][j])
        return result

def getDimensions(a):
    matHeight = len(a)
    if (matHeight == 0):
        return matHeight, matHeight
    matWidth = len(a[0])
    return matHeight, matWidth



def normalizeImage(updatedImage): #normalizing the output image
    imgHeight, imgWidth = getDimensions(updatedImage)
    updatedImagenorm = [[0 for x in range(imgWidth)] for y in range(imgHeight)]
    for i in range(imgHeight):
```

```
        for j in range(imgWidth):
            updatedImagenorm[i][j] = 128 + int(updatedImage[i][j]/2)
    return updatedImagenorm
#Program starts from here
for i in range(imageH):
    for j in range(imageW):
        currentWidthOffset = j
        currentHeightOffset = i
        sobelValue = calculateSobelat(currentWidthOffset,currentHeightOffset)
        updatedImageData[i][j] = sobelValue
image=normalizeImage(updatedImageData)
#print(updatedImageData)
gradX=np.asarray(image)
cv2.imwrite('/Users/stutishukla/Downloads/Result/task1/GradientX.png',gradX)
```

Source Code for finding Gradient X using Sobel Operator:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("/Users/stutishukla/Downloads/proj1_cse573/task1.png", cv2.IMREAD_GRAYSCALE)
a = np.asarray(img)
sobelW, sobelH = 3, 3
sobelData = [[0 for x in range(sobelW)] for y in range(sobelH)]
sobelData[0][0] = -1
sobelData[0][1] = -2
sobelData[0][2] = -1
sobelData[1][0] = 0
sobelData[1][1] = 0
sobelData[1][2] = 0
sobelData[2][0] = 1
sobelData[2][1] = 2
sobelData[2][2] = 1

imageH=len(img)
imageW=len(img[0])

updatedImageData = [[0 for x in range(imageW)] for y in range(imageH)]

def calculateSobelat(widthindex , heightindex): #applying sobel operator on every pixel of the matrix
    result = 0;
     for i in range(sobelH):
        for j in range(sobelW):
            currentWidthIndex = widthindex-1+j
            currentHeightIndex = heightindex-1+i
                if((currentHeightIndex<0) or (currentHeightIndex >= imageH)):
                    continue
                 if ((currentWidthIndex < 0) or (currentWidthIndex >= imageW)):
                     continue
```

```
                result += (a[currentHeightIndex][currentWidthIndex]*sobelData[i][j])
        return result

def getDimensions(a):
    matHeight = len(a)
    if (matHeight == 0):
        return matHeight, matHeight
    matWidth = len(a[0])
    return matHeight, matWidth

def normalizeImage(updatedImage): #normalizing the output image
    imgHeight, imgWidth = getDimensions(updatedImage)
    updatedImagenorm = [[0 for x in range(imgWidth)] for y in range(imgHeight)]

    for i in range(imgHeight):
        for j in range(imgWidth):
            updatedImagenorm[i][j] = 128 + int(updatedImage[i][j]/2)
    return updatedImagenorm
#Program starts from here
  for i in range(imageH):
    for j in range(imageW):
        currentWidthOffset = j
        currentHeightOffset = i
        sobelValue = calculateSobelat(currentWidthOffset,currentHeightOffset)
        updatedImageData[i][j] = sobelValue
image=normalizeImage(updatedImageData)

#print(updatedImageData)
gradY=np.asarray(image)
cv2.imwrite('/Users/stutishukla/Downloads/Result/task1/GradientY.png',gradY)
```

TASK 2: KEYPOINT DETECTION

Octave 2 has a resolution (width × height, unit pixel) of **375 × 229** after resizing the original image 750 × 458

(image 1 of octave 2)

(image 2 of octave 2)


(image 3 of octave 2)


(image 4 of octave 2)


(image 5 of octave 2)

Octave 3 has a resolution (width × height, unit pixel) of **187 × 114** after resizing second octave 375 × 229

Image 1 of octave 3

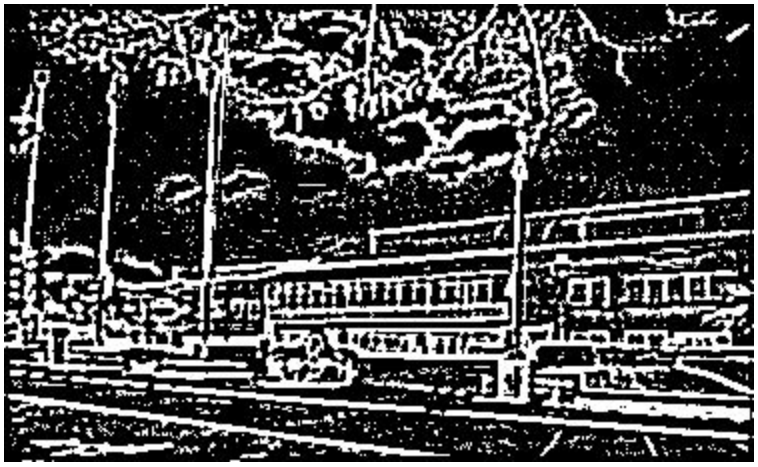
image 2 of octave 3

image 3 of octave 3

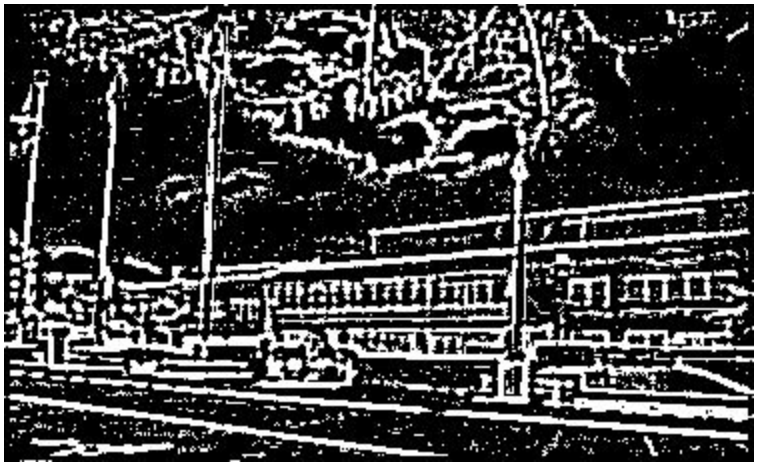
Image 4 of octave 3

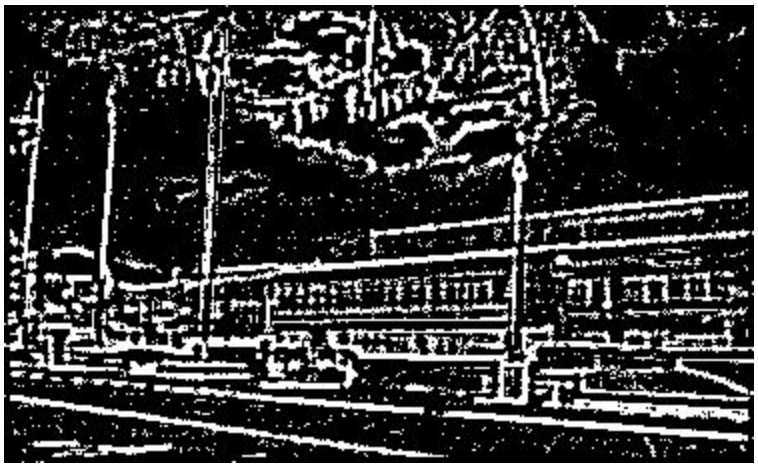
image 5 of octave 3

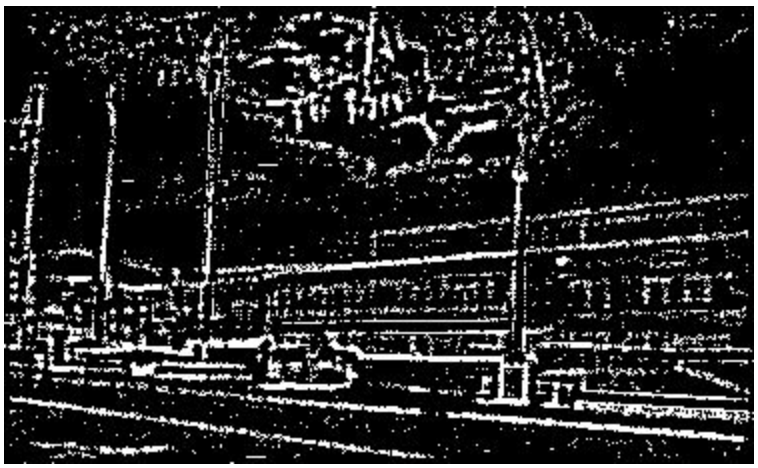
Difference of Gaussian (DOGs) of octave 2



(DOG 1 of octave 2)


(DOG 2 of octave 2)


(DOG 3 of octave 2)


(DOG 4 of octave 2)

Difference of Gaussian (DOGs) of octave 3

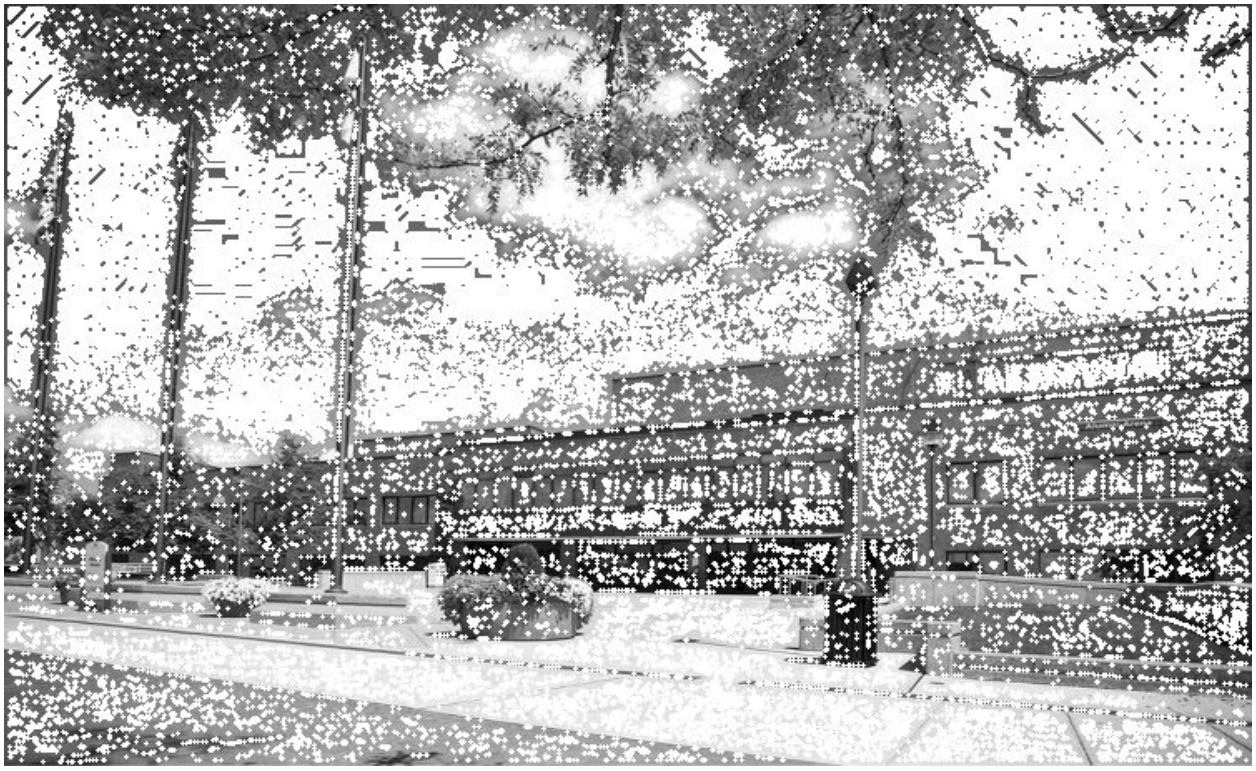
DOG1 of octave 3

DOG 2 of octave 3

DOG 3 of octave 3

DOG 4 of octave 3

All the detected white points on the original image



FIVE LEFTMOST DETECTED KEYPOINTS ARE: [2, 200], [2, 201], [3, 4], [3, 15], [3, 109]

Source Code for Octave 2:

```
import math
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("/Users/stutishukla/Downloads/proj1_cse573/task2.jpg", cv2.IMREAD_GRAYSCALE)
a=np.asarray(img)
def getDimensionsOfMatrix(a):
    matHeight = len(a)
    if (matHeight == 0):
        return matHeight, matHeight
    matWidth = len(a[0])
    return matWidth, matHeight

def resizeMatrix(a): #resize the given matrix
    resizedMatrix = []
    matrixWidth, matrixHeight = getDimensionsOfMatrix(a)
    for heightIndex in range(matrixHeight):
      if(heightIndex%2 == 1):
          continue
        tempRow = []
        for widthIndex in range(matrixWidth):
            if(widthIndex%2 == 0):
                tempRow.append(a[heightIndex][widthIndex])
        resizedMatrix.append(tempRow)
    return resizedMatrix

UpdatedMatrix = resizeMatrix(a)
b=np.asarray(UpdatedMatrix)

 def getDimensions(a):
    matHeight = len(a)
    if (matHeight == 0):
        return matHeight, matHeight
    matWidth = len(a[0])
    return matHeight, matWidth

def calculateGaussat(imagedata , widthindex, heightindex , gauss_kernel): # gives value of each pixel using kernel
    result = 0
    kernalHeight,kernalWidth = getDimensions((gauss_kernel))
    if(kernalHeight != kernalWidth):
        return 0
    midPointOffset = (kernalWidth-1)/2
    imageHeight,imageWidth = getDimensions(imagedata)
    for i in range(kernalHeight):
        for j in range(kernalWidth):
            currentHeightIndex = heightindex - midPointOffset + i
            currentWidthIndex = widthindex - midPointOffset + j
            if ((currentHeightIndex < 0) or (currentHeightIndex >= imageHeight)):
```

```
            continue
        if ((currentWidthIndex < 0) or (currentWidthIndex >= imageWidth)):
            continue
        result += (UpdatedMatrix[int(currentHeightIndex)][int(currentWidthIndex)] * gauss_kernel[int(i)][int(j)])
    return int(result)

def fxy(x,y,sigma):          #calculating sigma values
    temp1 = 1/(2 * math.pi * sigma * sigma)
    temp2 = math.exp( -1* ((x*x + y*y)/(2*sigma*sigma)))
    return  temp1*temp2

def getGuassianKernalforSigma(sigmaValue):   #calculating gaussian kernel
    total=0;
    gauss_kernel = [[0 for x in range(7)] for y in range(7)]
    for k in range(-3,4):
        for j in range(3,-4,-1):
            # result =
((1/(2*3.14*(sigmaValues[i]*sigmaValues[i])))*(math.exp(-((j*j)+(k*k))/2*sigmaValues[i]*sigmaValues[i])))
            result=fxy(j,k,sigmaValue)
            gauss_kernel[k + 3][3 - j] = result
            total=total+result
    return gauss_kernel,total

def applyGuassianKernaltoImage(imageData , gaussianKernal):   #applying gaussian kernel
    imageHeight,imageWidth = getDimensions(imageData)
    updatedImageData = [[0 for x in range(imageWidth)] for y in range(imageHeight)]
    for i in range(imageHeight):
        for j in range(imageWidth):
            currentHeightOffset = i
            currentWidthOffset = j
            gossValue = calculateGaussat(imageData,currentWidthOffset,currentHeightOffset,gaussianKernal)
            updatedImageData[i][j] = gossValue
    return updatedImageData

def normalizeGaussKernel(gauss_kernel_raw,total):   #normalizing gaussian kernel
     kernalHeight,kernalWidth = getDimensions((gauss_kernel_raw))
     for i in range(kernalHeight):
        for j in range(kernalWidth):
            gauss_kernel_raw[i][j]=gauss_kernel_raw[i][j]/total
     return gauss_kernel_raw

def DiffOfGauss(img1, img2):   #calculating DOGs
    imgHeight, imgWidth = getDimensions(img2)
    DOG = [[0 for x in range(imgWidth)] for y in range(imgHeight)]
    for i in range(imgHeight):
        for j in range(imgWidth):
            DOG[i][j] = (img2[i][j] - img1[i][j])
    return DOG
```

```
def normalizeDOG(DOG):   #normalizing DOGs
    imgHeight, imgWidth = getDimensions(DOG)
    DOGnorm = [[0 for x in range(imgWidth)] for y in range(imgHeight)
    maximum=np.max(DOG)
    minimum=np.min(DOG)
    for i in range(imgHeight):
        for j in range(imgWidth):
            DOGnorm[i][j] = (DOG[i][j]- minimum) / (maximum-minimum)
    return DOGnorm

def keypointMaximumDetection(dog_mid, dog_up, dog_down,finaMatrix):   #keypoint detection maxima
    height, width = getDimensions(dog_mid)
    for h in range(1, height - 1):
        for w in range(1, width - 1):
            # traversing and comparing 26 neighbours'
            is_maxima = True
            for i in range(h - 1, h + 2):
                for j in range(w - 1, w + 2):
                    if (dog_mid[h][w] < dog_mid[i][j]) or (dog_mid[h][w] < dog_up[i][j]) or (dog_mid[h][w] < dog_down[i][j]):
                        is_maxima = False
                        break
                if not is_maxima:
                        break
            if is_maxima:
                finaMatrix.append([h,w])

    return

def keypointMinimumDetection(dog_mid, dog_up, dog_down,finaMatrix):   #keypoint detection minima
    height, width = getDimensions(dog_mid)
    for h in range(1, height - 1):
        for w in range(1, width - 1):
            # traversing and comparing 26 neighbours'
            is_minima = True
            for i in range(h - 1, h + 2):
                for j in range(w - 1, w + 2):
                    if (dog_mid[h][w] > dog_mid[i][j]) or (dog_mid[h][w] > dog_up[i][j]) or (dog_mid[h][w] > dog_down[i][j]):
                        is_minima = False
                        break
                if not is_minima:
                        break
            if is_minima:
                finaMatrix.append([h,w])
    return

# Program starts from here :
sigmaValue1=(math.sqrt(2))
gauss_kernel_raw, total = getGuassianKernalforSigma(sigmaValue1)  # computing gaussian kernal for given sigma
value
```

```
gauss_kernel = normalizeGaussKernel(gauss_kernel_raw, total)
outputImage1 = applyGuassianKernaltoImage(UpdatedMatrix, gauss_kernel)  # applying computed gaussian kernal to our image
# print outputImage here
b1 = np.asarray(outputImage1)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-1.png',b1)

sigmaValue2=2
gauss_kernel_raw, total = getGuassianKernalforSigma(sigmaValue2)  # computing gaussian kernal for given sigma value
gauss_kernel = normalizeGaussKernel(gauss_kernel_raw, total)
outputImage2 = applyGuassianKernaltoImage(UpdatedMatrix, gauss_kernel)  # applying computed gaussian kernal to our image
# print outputImage here
b2 = np.asarray(outputImage2)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-2.png',b2)

DogI = DiffOfGauss(b1, b2) #DOG1 of octave 2
DogIA = normalizeDOG(DogI)
DogInorm = np.asarray(DogIA)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-1DOG.png', DogInorm)

sigmaValue3=2*(math.sqrt(2))
# for sigmaValue in sigmaValues: # Iterating for each sigma value
gauss_kernel_raw, total = getGuassianKernalforSigma(sigmaValue3)  # computing gaussian kernal for given sigma value
gauss_kernel = normalizeGaussKernel(gauss_kernel_raw, total)
outputImage3 = applyGuassianKernaltoImage(UpdatedMatrix, gauss_kernel)  # applying computed gaussian kernal to our image
# print outputImage here
b3 = np.asarray(outputImage3)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-3.png',b3)

DogII = DiffOfGauss(b2, b3) #DOG2 of octave 2
DogIIA = normalizeDOG(DogII)
DogInorm = np.asarray(DogIIA)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-2DOG.png', DogInorm)

sigmaValue4=4
# for sigmaValue in sigmaValues: # Iterating for each sigma value
gauss_kernel_raw, total = getGuassianKernalforSigma(sigmaValue4)  # computing gaussian kernal for given sigma value
gauss_kernel = normalizeGaussKernel(gauss_kernel_raw, total)
outputImage4 = applyGuassianKernaltoImage(UpdatedMatrix, gauss_kernel)  # applying computed gaussian kernal to our image
# print outputImage here
b4 = np.asarray(outputImage4)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-4.png',b4)
```

```
DogIII = DiffOfGauss(b3, b4) #DOG3 of octave 2
DogIIIA = normalizeDOG(DogIII)
DogInorm = np.asarray(DogIIIA)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-3DOG.png', DogInorm)

sigmaValue5=4*(math.sqrt(2))
# for sigmaValue in sigmaValues: # Iterating for each sigma value
gauss_kernel_raw, total = getGuassianKernalforSigma(sigmaValue5)  # computing gaussian kernal for given sigma value
gauss_kernel = normalizeGaussKernel(gauss_kernel_raw, total)
outputImage5 = applyGuassianKernaltoImage(UpdatedMatrix, gauss_kernel)  # applying computed gaussian kernal to our image
# print outputImage here
b5 = np.asarray(outputImage5)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-5.png',b5)

DogIV = DiffOfGauss(b4, b5) #DOG4 of octave 2
DogIVA = normalizeDOG(DogIV)
DogInorm = np.asarray(DogIVA)
cv2.imwrite('/Users/stutishukla/Downloads/Result/octave2/task2-4DOG.png', DogInorm)

finaMatrix = []

#keypointMaximumDetection(DogIIA, DogIA, DogIIIA,finaMatrix)
keypointMaximumDetection(DogIIA, DogIA, DogIIIA,finaMatrix)
#keypointDetectMax = np.asarray(finaMatrix)
for keypoint in finaMatrix:
    cv2.circle(b,(keypoint[1],keypoint[0]),1,(255,255,0), -1)
keypointDetectMax=np.asarray(img)
cv2.imwrite('/Users/stutishukla/Downloads/Result/task2/keyPointDetection/OctaveII/keyPointDetectMaxDOGII.png', b)

keypointMinimumDetection(DogIIA, DogIA, DogIIIA,finaMatrix)
#keypointMinimumDetection(DogIIA, DogIA, DogIIIA,finaMatrix)
for keypoint in finaMatrix:
    cv2.circle(b,(keypoint[1],keypoint[0]),1,(255,255,0), -1)
cv2.imwrite('/Users/stutishukla/Downloads/Result/task2/keyPointDetection/OctaveII/keyPointDetectDOGII.png', b)

keypointMaximumDetection(DogIIIA, DogIIA, DogIVA,finaMatrix)
#keypointDetectMax = np.asarray(finaMatrix)
for keypoint in finaMatrix:
    cv2.circle(b,(keypoint[1],keypoint[0]),1,(255,255,0), -1)
keypointDetectMax=np.asarray(img)
cv2.imwrite('/Users/stutishukla/Downloads/Result/task2/keyPointDetection/OctaveII/keyPointDetectMaxDOGIII.png', b)

keypointMinimumDetection(DogIIIA, DogIIA, DogIVA,finaMatrix)
#keypointMinimumDetection(DogIIA, DogIA, DogIIIA,finaMatrix)
for keypoint in finaMatrix:
```

```
    cv2.circle(b,(keypoint[1],keypoint[0]),1,(255,255,0), -1)
cv2.imwrite('/Users/stutishukla/Downloads/Result/task2/keyPointDetection/OctaveII/keyPointDetectDOGIII.png', b)
#Due to constraint of report pages, I have only included the important piece of code. This is not a complete code.
Kindly refer my code submission on UBlerans for the same.
#Similar is the source code for octave 3 with a few changes
```

TASK 3: CURSOR DETECTION

Approach for cursor detection:

I attempted this task using two approaches:

First Approach:

1. Read the image and the template from a source location.
2. Convert them to grayscale.
3. Resize the given template for the ease of matching.
4. Apply gaussian blur to the image.
5. Apply laplacian transformation to both the image and the template.
6. Apply cv2.matchTemplate() method. There are six different methods for matching template. I tried all the six methods and amongst them cv2.TM_CCOEFF_NORMED gave the best results.

With this approach, I was able to detect cursor in a few images. However, its accuracy was less.

Image 1

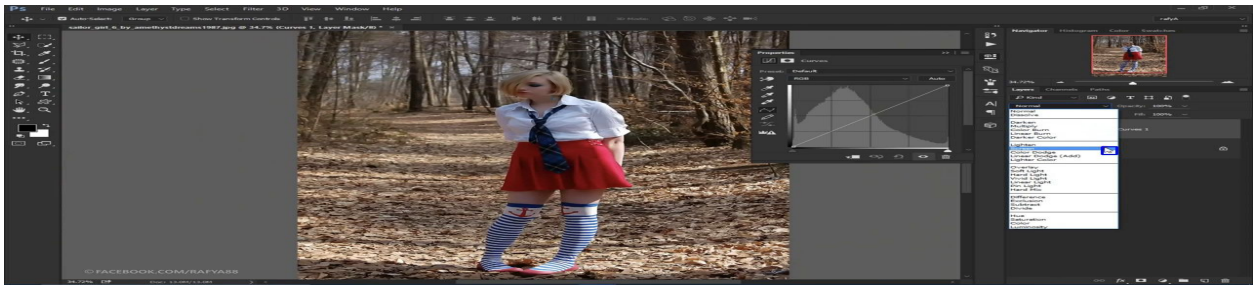


Image 2

Source Code for cursor detection:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('/Users/stutishukla/Downloads/proj1_cse573/task3/neg_5.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
template = cv2.imread('/Users/stutishukla/Downloads/proj1_cse573/task3/template.png')
template_gray=cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

def getDimensionsOfMatrix(a):
    matHeight = len(a)
    if (matHeight == 0):
        return matHeight, matHeight
    matWidth = len(a[0])
    return matWidth, matHeight

def resizeMatrix(a): #customizing the given template by resizing it
    resizedMatrix = []
    matrixWidth, matrixHeight = getDimensionsOfMatrix(a)
    for heightIndex in range(matrixHeight):
        if(heightIndex%2 == 1):
            continue
        tempRow = []
        for widthIndex in range(matrixWidth):
            if(widthIndex%2 == 0):
                tempRow.append(a[heightIndex][widthIndex])
        resizedMatrix.append(tempRow)

    return resizedMatrix
template_grayResized=resizeMatrix(template_gray)
a=np.asarray(template_grayResized)
blurImage = cv2.GaussianBlur(img_gray,(3,3),0) #Applying gaussian blur on the image
blurTemplate = cv2.GaussianBlur(a,(3,3),0)        #Applying gaussian blur on the template
laplacian1 = cv2.Laplacian(blurImage,cv2.CV_64F)  #Applying laplacian transformation on the image
laplacian2 = cv2.Laplacian(blurTemplate,cv2.CV_64F)  #Applying laplacian transformation on the template
w, h= a.shape[::-1]
 # Apply template Matching
res = cv2.matchTemplate(laplacian1.astype(np.float32),laplacian2.astype(np.float32),cv2.TM_CCOEFF_NORMED)
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
top_left = max_loc
bottom_right = (top_left[0] + w, top_left[1] + h)
cv2.rectangle(img,top_left, bottom_right, 255, 2)
cv2.imshow('Task3.png',img)
cv2.imwrite('/Users/stutishukla/Downloads/Result/task3/task3-neg5.png',img)
```

-Referred online sources for the code of cursor detection

Second Approach:

For second approach, the aim was to be able to detect cursor with more accuracy and also multiple templates in an image (if present). For this purpose, I used the **threshold value** and inferred that templates in all the images fell within the threshold range of **0.4 to 0.7**. For neg_images, the threshold was constant at **0.6.** Although this approach gave slightly better results, it was detecting lots of false templates as well.