Image Processing

Dhayanidhi Gunasekaran
Department of Computer Science
University at Buffalo
Buffalo, NY 14260
dhayanid@buffalo.edu

Abstract

To perform the image processing tasks such as Edge detection, Key point detection and template matching in the given set of images.

1 Edge Detection

Edge detection aims at identifying points in the digital image at which the image brightness changes sharply or discontinuities. The point at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. More commonly used edge detection methods are Sobel, Canny and fuzzy logic. In this project we use Sobel operator for finding the edges.

1.1 Sobel Method

Sobel operator is a discrete differentiation operator computing an approximation of the gradient of the image intensity function. At each point in the image, the result of Sobel filter is either the corresponding gradient vector or the norm of this vector. The operator uses 3 * 3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal (x-axis) and one for vertical(y-axis)

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

*represents the 2-Dimensional signal processing convolution operation.

The formula to compute the magnitude of the operation is as follows.

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

1.2 Task Objective

To detect the edges in the given input image using Sobel operator. Sobel operator has to be applied across both x and y axis and to get the corresponding images.

Figure 1: Input Image for Edge Detection

1.3 Procedure

The implementation is done using python, opencv and numpy libraries.

a) Image is read using imread function.
b) Output matrix of same size as the input is generated.
c) Gaussian kernel for x and y axis is initialized.
d) For each pixel in the input image, a 3*3 patch matrix is generated.
e) The generated Gaussian kernel for x-axis is applied to each and every location.
f) Similarly, Sobel operator is computed for y axis.
g) Finally, the magnitude is computed using the above formula.
h) The output images are printed using cv.imwrite function.

1.4 Code

```
# Import Required Libraries
import numpy as np
import cv2 as cv

def get_Max(matrix):
   largest_num = matrix[0][0]
   for row_idx, row in enumerate(matrix):
      for col_idx, num in enumerate(row):
         if num > largest_num:
            largest_num = num

   return largest_num


def Normalise_Matrix(Matrix):
   MAX_VALUE = get_Max(Matrix)
   for i in range(600):
      for j in range(900):
         Matrix[i][j] = (Matrix[i][j]/MAX_VALUE)*255

   return Matrix


def initialise_matrix(row, col):
   matrix = [[0 for x in range(col)] for y in range(row)]
   return matrix


def elem_wise_multiple(MAT_A, MAT_B, row, col):
```

```
    MAT = initialise_matrix(3, 3)
    for i in range(row):
        for j in range(col):
            MAT[i][j] = MAT_A[i][j] * MAT_B[i][j]
    return MAT

def sum_of_elems(MAT):
    value = 0
    row = len(MAT)
    col = len(MAT[0])
    for i in range(row):
        for j in range(col):
            value = value + MAT[i][j]

    return value

# funtion to get 3*3 matrix based on its position
def get_3_cross3(matrix, row, col):
    MAT = initialise_matrix(3, 3)
    if row == 0 or col == 0:
        MAT[0][0] = 0
    else:
        MAT[0][0] = matrix[row-1][col-1]

    if row == 0:
        MAT[0][1] = 0
    else:
        MAT[0][1] = matrix[row-1][col]

    if row == 0 or col == len(matrix[0])-1:
        MAT[0][2] = 0
    else:
        MAT[0][2] = matrix[row-1][col+1]

    if col == 0:
        MAT[1][0] = 0
    else:
        MAT[1][0] = matrix[row][col-1]

    MAT[1][1] = matrix[row][col]

    if col == len(matrix[0])-1:
        MAT[1][2] = 0
    else:
        MAT[1][2] = matrix[row][col+1]

    if row == len(matrix)-1 or col == 0:
        MAT[2][0] = 0
    else:
        MAT[2][0] = matrix[row+1][col-1]

    if row == len(matrix)-1:
        MAT[2][1] = 0
    else:
        MAT[2][1] = matrix[row+1][col]

    if row == len(matrix)-1 or col == len(matrix[0])-1:
        MAT[2][2] = 0
```

```
    else:
        MAT[2][2] = matrix[row+1][col+1]

    return MAT


# Read the input image in Gray scale format
INPUT_IMAGE = cv.imread('task1.png', cv.IMREAD_GRAYSCALE)

# cv.namedWindow("Input Image")
# cv.imshow('Image',inputim)
# cv.waitKey(0)
print(INPUT_IMAGE.shape)


# Initialise the Gx and  Gy matrix
Gx = [[1, 0, -1],
[2, 0, -2],
 [1, 0, -1]]
Gy = [[1, 2, 1],
[0, 0, 0],
 [-1, -2, -1]]

rows = len(INPUT_IMAGE)
cols = len(INPUT_IMAGE[0])
INPUTXEDGE = initialise_matrix(rows, cols)
NUM_MAT = initialise_matrix(rows, cols)
INPUTYEDGE = initialise_matrix(rows, cols)

# loop through the matrices and calculate Gx * Input
for i in range(rows):
    for j in range(cols):
        THREE_CROSS_THREE = get_3_cross3(INPUT_IMAGE, i, j)
        OUTPUT = elem_wise_multiple(Gx, THREE_CROSS_THREE, 3, 3)
        INPUTXEDGE[i][j] = sum_of_elems(OUTPUT)

for i in range(rows):
    for j in range(cols):
        THREE_CROSS_THREE = get_3_cross3(INPUT_IMAGE, i, j)
        OUTPUT = elem_wise_multiple(Gy, THREE_CROSS_THREE, 3, 3)
        INPUTYEDGE[i][j] = sum_of_elems(OUTPUT)

OP = initialise_matrix(rows, cols)

for i in range(rows):
    for j in range(cols):
        OP[i][j] = (((INPUTXEDGE[i][j]**2)+(INPUTYEDGE[i][j]**2))**0.5)

INPUTXEDGE = Normalise_Matrix(INPUTXEDGE)
INPUTYEDGE = Normalise_Matrix(INPUTYEDGE)
OP = Normalise_Matrix(OP)

INPUTXEDGE = np.asarray(INPUTXEDGE)
INPUTYEDGE = np.asarray(INPUTYEDGE)
OP = np.asarray(OP)

cv.imwrite( "Edge_along_x.jpg",INPUTXEDGE )
cv.imwrite( "Edge_along_y.jpg",INPUTYEDGE )
```

```
cv.imwrite( "output.jpg", OP)
print("Output Image Generated")
```

1.4 Output Images

The Edge detected along the x-axis, y-axis and the magnitude computed is shown below.

Figure 2: Edge detected along x-axis

Figure 3: Edge detected along y-axis

Figure 4: Output Image showing the edges

2 Key point Detection using SIFT

Scale-Invariant Feature Transform commonly called as SIFT is a method used to detect the key features in a given image. SIFT is a widely used approach for key point detection. The objective of this method is to extract the interesting points for any object in the given image. SIFT can identify objects even among clutter and under partial occlusion since it is invariant to uniform scaling, orientation, illumination changes and partially invariant to affine transformations.

2.1 SIFT Steps

a) Scales space: In this step, several octaves of the original image is generated. Each octave's image size is half the size of previous one. Within an octave, images are progressively blurred using the gaussian operator.

b) Log approximations: Blurred images generated in the previous step is used to compute the Difference of Gaussian (DoG) images.

c) Finding Key points: Finding the local Maxima and Minima in DoG Images.

d) Removing Low contrast points: Based on the magnitude of the intensity, low contrast features and edges are removed.

e) Key point orientation: Finding and assigning the most prominent orientation to the key point based on the collected gradient directions and orientations.

f) Generating a feature: An unique fingerprint for a key point is generated which is a vector which has 128 different numbers.

2.2 Task Objective

To implement the first three steps of the SIFT algorithm. Also to generate four octaves, each octave composed of 5 images blurred with gaussian kernels for different bandwidth parameters given in table 1.

To compute DoG of all the octaves and to detect key points located at the maxima and minima of the DoG images.

Octave					
1st	$\frac{1}{\sqrt{2}}$	1	$\sqrt{2}$	2	$2\sqrt{2}$
2nd	$\sqrt{2}$	2	$2\sqrt{2}$	4	$4\sqrt{2}$
3rd	$2\sqrt{2}$	4	$4\sqrt{2}$	8	$8\sqrt{2}$
4th	$4\sqrt{2}$	8	$8\sqrt{2}$	16	$16\sqrt{2}$

Table 1: Different bandwidth Parameters

Figure 5: Input image for Task 2

2.3 Procedure

a) Input image is read using cv.imread function call.

b) Input image is scaled down into 4 different sizes each having half the size of previous image.

c) Each image is then passed into the Octave function.

d) When an image is passed to the Octave function, Gaussian Kernels for different bandwidth parameters are generated.

e) Each gaussian kernel is then processed to the image thus giving 5 result images for each octave.

f) Next step is to calculate the DoG for the previously processed 5 images. Thus resulting in 4 Dog Images. DoG is nothing but the difference of two gaussian kernel images.

g) Key points are generated based on calculating the maxima or minima. If a point is either maximum when compared to the corresponding pixel of the two DoG images which is located above and below the current image.

h) Key points are then plotted in the original image.

2.4 Code

```
# This python program is to get an input image and find out the key points in the given image
# using SIFT Detection Algorithm
import numpy as np
import cv2 as cv
import math as math
```

```
#This function generates a matrix with given row,col size filled with zeros
def initialise_matrix(row, col):
    matrix = [[0 for x in range(col)] for y in range(row)]
    return matrix

#this function returns the max value for the given matrix
def get_Max(matrix):
    largest_num = matrix[0][0]
    for row_idx, row in enumerate(matrix):
        for col_idx, num in enumerate(row):
            if num > largest_num:
                largest_num = num

    return largest_num

#This function normalises the image matrix to 0-255 scale
def Normalise_Matrix(Matrix):
    row = len(Matrix)
    col = len(Matrix[0])
    MAX_VALUE = get_Max(Matrix)
    for i in range(row):
        for j in range(col):
            Matrix[i][j] = (Matrix[i][j]/MAX_VALUE)*255
    return Matrix

# funtion to get 3*3 matrix based on its position
def get_3_cross3(matrix, row, col):
    MAT = initialise_matrix(3, 3)
    if row == 0 or col == 0:
        MAT[0][0] = 0
    else:
        MAT[0][0] = matrix[row-1][col-1]

    if row == 0:
        MAT[0][1] = 0
    else:
        MAT[0][1] = matrix[row-1][col]

    if row == 0 or col == len(matrix[0])-1:
        MAT[0][2] = 0
    else:
        MAT[0][2] = matrix[row-1][col+1]

    if col == 0:
        MAT[1][0] = 0
    else:
        MAT[1][0] = matrix[row][col-1]

    MAT[1][1] = matrix[row][col]

    if col == len(matrix[0])-1:
        MAT[1][2] = 0
    else:
        MAT[1][2] = matrix[row][col+1]

    if row == len(matrix)-1 or col == 0:
        MAT[2][0] = 0
```

```
    else:
        MAT[2][0] = matrix[row+1][col-1]

    if row == len(matrix)-1:
        MAT[2][1] = 0
    else:
        MAT[2][1] = matrix[row+1][col]

    if row == len(matrix)-1 or col == len(matrix[0])-1:
        MAT[2][2] = 0
    else:
        MAT[2][2] = matrix[row+1][col+1]

    return MAT

#this function pads 0 for the edge rows and cols
def generatePatchMatrix(matrix, row, col):
    PATCH_MAT = initialise_matrix(5, 5)
    row_i = row - 2
    for i in range(5):
        col_i = col - 2
        for j in range(5):
            if row_i < 0 or row_i > len(matrix)-1 or col_i < 0 or col_i > len(matrix[0])-1:
                PATCH_MAT[i][j] = 0
            else:
                PATCH_MAT[i][j] = matrix[row_i][col_i]
            col_i = col_i + 1

        if i > 4:
            row_i = row - 2
        else:
            row_i = row_i + 1
    return PATCH_MAT

# this function normalised the gaussian kernel
def GaussianNormalisation(matrix):
    row = len(matrix)
    col = len(matrix[0])
    total = 0
    for i in range(row):
        for j in range(col):
            total = total + matrix[i][j]

    for i in range(row):
        for j in range(col):
            matrix[i][j] = matrix[i][j]/total
    return matrix

# this function generates the gaussian kernel for the given Sigma and weight
def generateGaussianKernel(weight, sigma):
    matrix = initialise_matrix(weight, weight)
    row_i = -2
    for i in range(weight):
        col_i = -2
        for j in range(weight):
            power_elem = -((row_i**2)+(col_i**2))/(2*(sigma**2))
            matrix[i][j] = (1/(2*math.pi*(sigma**2)))*math.exp(power_elem)
            col_i = col_i + 1
```

```
        if i == 4:
            row_i = -2
        else:
            row_i = row_i + 1

    matrix = GaussianNormalisation(matrix)
    return matrix

# this function returns the convoluted matrix
def computeConvolutedMat(matrix):
    row = len(matrix)
    col = len(matrix[0])
    row_i = row - 1
    col_i = col - 1
    OUTPUT = initialise_matrix(row, col)
    for i in range(row):
        col_i = col - 1
        for j in range(col):
            OUTPUT[i][j] = matrix[row_i][col_i]
            col_i = col_i - 1
        if i == 457:
            row_i = row - 1
        else:
            row_i = row_i - 1

    return OUTPUT

# this function does element wise multiplication of the given 2 matrices
def elem_wise_operation(kernel, pos):
    op = initialise_matrix(5, 5)
    for i in range(5):
        for j in range(5):
            op[i][j] = kernel[i][j]*pos[i][j]
    return op

# this function returns the sum of all the values in a matrix
def sum_of_elems(MAT):
    value = 0
    row = len(MAT)
    col = len(MAT[0])
    for i in range(row):
        for j in range(col):
            value = value + MAT[i][j]

    return value

# this is a main function which applies gaussian filter to each n every
# pixel in a given input image
def AddGaussianFilterToImage(sigma, image):
    GaussianKernel = generateGaussianKernel(5, sigma)
    op_mat = initialise_matrix(len(image), len(image[0]))
    for i in range(len(image)):
        for j in range(len(image[0])):
            pos_mat = generatePatchMatrix(image, i, j)
          #  pos_con_mat = computeConvolutedMat(pos_mat)
            computed_mat = elem_wise_operation(pos_mat, GaussianKernel)
            op_mat[i][j] = sum_of_elems(computed_mat)
```

```
    return op_mat

# this function scales down the given image matrix to half
def ScaleDownImage(Mat):
    row = len(Mat)
    col = len(Mat[0])
    row_i = row // 2
    col_i = col // 2
    op = initialise_matrix(row_i, col_i)
    for i in range(row_i):
        for j in range(col_i):
            op[i][j] = Mat[2*i][2*j]

    return op

# this function calculates the DoG of given 2 Matrices
# Gaussian 1 - Gaussian 2.. Element wise subtraction
def calculate_DOG(Mat1, Mat2):
    row = len(Mat1)
    col = len(Mat1[0])
    output = initialise_matrix(row,col)
    for i in range(row):
        for j in range(col):
            output[i][j] = Mat1[i][j] - Mat2[i][j]

    return output

#this function get three matrices and check whether the value is maximum of minimum
def getKeyPoint(Mat1,Mat2,Mat3,Value):
    row = len(Mat1)
    col = len(Mat1[0])
    complete_list = []
    for i in range(row):
        for j in range(col):
            complete_list.append(Mat1[i][j])
            complete_list.append(Mat2[i][j])
            complete_list.append(Mat3[i][j])

    max_value = max(complete_list)
    min_value = min(complete_list)

    if Value == max_value or Value == min_value:
        return Value
    else:
        return 0

#this function returns and output matrix with only the maxima and minima values
def calculate_keypoints(Mat1,Mat2,Mat3):
    row = len(Mat1)
    col = len(Mat1[0])
    op = initialise_matrix(row,col)
    for i in range(row):
        for j in range(col):
            Pix1 =  get_3_cross3(Mat1,i,j)
            Pix2 =  get_3_cross3(Mat2,i,j)
            Pix3 =  get_3_cross3(Mat3,i,j)
            op[i][j] = getKeyPoint(Pix1,Pix2,Pix3,Mat2[i][j])
```

```
    return op

def mark_keypoints(Key1,Key2,Color_image):
    row = len(Color_image)
    col = len(Color_image[0])
    for i in range(row):
        for j in range(col):
            if Key1[i][j] > 3.0 or Key2[i][j] > 3.0:
                Color_image[i][j] = 255

    return Color_image

def octave1_gaussian(INPUT_IMAGE):
    # OCTAVE 1 Code starts here
    # Computing the Matrix for the given original image for various sigma values
    # Sigma values = 1/root2,1, root2,2,2root2
    OCT1GAUS1 = AddGaussianFilterToImage(0.707, INPUT_IMAGE)
    OCT1GAUS2 = AddGaussianFilterToImage(1, INPUT_IMAGE)
    OCT1GAUS3 = AddGaussianFilterToImage(1.414, INPUT_IMAGE)
    OCT1GAUS4 = AddGaussianFilterToImage(2, INPUT_IMAGE)
    OCT1GAUS5 = AddGaussianFilterToImage(2.828, INPUT_IMAGE)
    print("Octave 1 Gaussian Completed")
    cv.imwrite( "octave1_Gaus1.jpg", np.asarray(Normalise_Matrix(OCT1GAUS1)))
    cv.imwrite( "octave1_Gaus2.jpg", np.asarray(Normalise_Matrix(OCT1GAUS2)))
    cv.imwrite( "octave1_Gaus3.jpg", np.asarray(Normalise_Matrix(OCT1GAUS3)))
    cv.imwrite( "octave1_Gaus4.jpg", np.asarray(Normalise_Matrix(OCT1GAUS4)))
    cv.imwrite( "octave1_Gaus5.jpg", np.asarray(Normalise_Matrix(OCT1GAUS5)))
    #These function calls calculate the difference of Gaussian
    #Store it in the respective Identifiers
    OCT1DOG1 = calculate_DOG(OCT1GAUS1,OCT1GAUS2)
    OCT1DOG2 = calculate_DOG(OCT1GAUS2,OCT1GAUS3)
    OCT1DOG3 = calculate_DOG(OCT1GAUS3,OCT1GAUS4)
    OCT1DOG4 = calculate_DOG(OCT1GAUS4,OCT1GAUS5)
    print("Octave 1 DoG Completed")
    cv.imwrite( "octave1_dog1.jpg", np.asarray(Normalise_Matrix(OCT1DOG1)))
    cv.imwrite( "octave1_dog2.jpg", np.asarray(Normalise_Matrix(OCT1DOG2)))
    cv.imwrite( "octave1_dog3.jpg", np.asarray(Normalise_Matrix(OCT1DOG3)))
    cv.imwrite( "octave1_dog4.jpg", np.asarray(Normalise_Matrix(OCT1DOG4)))

    #This part of code Locates the maxima and minima to find the keypoints
    OCT1Key1 = calculate_keypoints(OCT1DOG1,OCT1DOG2,OCT1DOG3)
    OCT1Key2 = calculate_keypoints(OCT1DOG2,OCT1DOG3,OCT1DOG4)
    #cv.imshow('octave1_Key1', np.asarray(OCT1Key1))
    cv.imwrite( "octave1_key1.jpg", np.asarray(Normalise_Matrix(OCT1Key1)))
    cv.imwrite( "octave1_key2.jpg", np.asarray(Normalise_Matrix(OCT1Key2)))
    print("Octave 1 Keypoint images Generated")
    #cv.waitKey(0)

    #this part of code points the generated key point value in original image
    OCT1FINAL = mark_keypoints(OCT1Key1,OCT1Key2,INPUT_IMAGE)
    ORGFINAL = mark_keypoints(OCT1Key1,OCT1Key2,Color_image)
    cv.imwrite("octave1_final.jpg",np.asarray(OCT1FINAL))
    cv.imwrite("original_key_final.jpg",np.asarray(ORGFINAL))
    print("Octave 1 final Image generated")

def octave2_gaussian(second_image):
    # OCTAVE 2 Code starts here
```

```
    # Computing the Matrix for the given original image for various sigma values
    # Sigma values = 1.414,2,2.828,4,5.657
    OCT2GAUS1 = AddGaussianFilterToImage(1.414, second_image)
    OCT2GAUS2 = AddGaussianFilterToImage(2, second_image)
    OCT2GAUS3 = AddGaussianFilterToImage(2.828, second_image)
    OCT2GAUS4 = AddGaussianFilterToImage(4, second_image)
    OCT2GAUS5 = AddGaussianFilterToImage(5.657, second_image)
    print("Octave 2 Gaussian Completed")
    cv.imwrite( "octave2_Gaus1.jpg", np.asarray(Normalise_Matrix(OCT2GAUS1)))
    cv.imwrite( "octave2_Gaus2.jpg", np.asarray(Normalise_Matrix(OCT2GAUS2)))
    cv.imwrite( "octave2_Gaus3.jpg", np.asarray(Normalise_Matrix(OCT2GAUS3)))
    cv.imwrite( "octave2_Gaus4.jpg", np.asarray(Normalise_Matrix(OCT2GAUS4)))
    cv.imwrite( "octave2_Gaus5.jpg", np.asarray(Normalise_Matrix(OCT2GAUS5)))

    #These function calls calculate the difference of Gaussian
    #Store it in the respective Identifiers
    OCT2DOG1 = calculate_DOG(OCT2GAUS1,OCT2GAUS2)
    OCT2DOG2 = calculate_DOG(OCT2GAUS2,OCT2GAUS3)
    OCT2DOG3 = calculate_DOG(OCT2GAUS3,OCT2GAUS4)
    OCT2DOG4 = calculate_DOG(OCT2GAUS4,OCT2GAUS5)
    cv.imwrite( "octave2_dog1.jpg", np.asarray(Normalise_Matrix(OCT2DOG1)))
    cv.imwrite( "octave2_dog2.jpg", np.asarray(Normalise_Matrix(OCT2DOG2)))
    cv.imwrite( "octave2_dog3.jpg", np.asarray(Normalise_Matrix(OCT2DOG3)))
    cv.imwrite( "octave2_dog4.jpg", np.asarray(Normalise_Matrix(OCT2DOG4)))
    print("Octave 2 DoG Completed")

    #This part of code Locates the maxima and minima to find the keypoints
    OCT2Key1 = calculate_keypoints(OCT2DOG1,OCT2DOG2,OCT2DOG3)
    OCT2Key2 = calculate_keypoints(OCT2DOG2,OCT2DOG3,OCT2DOG4)
    cv.imwrite( "octave2_key1.jpg", np.asarray(Normalise_Matrix(OCT2Key1)))
    cv.imwrite( "octave2_key2.jpg", np.asarray(Normalise_Matrix(OCT2Key2)))
    print("Octave 2 Keypoint images Generated")

    #this part of code points the generated key point value in original image
    OCT2FINAL = mark_keypoints(OCT2Key1,OCT2Key2,second_image)
    cv.imwrite("octave2_final.jpg",np.asarray(OCT2FINAL))
    print("Octave 2 final Image generated")


def octave3_gaussian(third_image):
    # OCTAVE 3 Code starts here
    # Computing the Matrix for the given original image for various sigma values
    # Sigma values = 2.828,4,5.657,8,11.314
    OCT3GAUS1 = AddGaussianFilterToImage(2.828, third_image)
    OCT3GAUS2 = AddGaussianFilterToImage(4, third_image)
    OCT3GAUS3 = AddGaussianFilterToImage(5.657, third_image)
    OCT3GAUS4 = AddGaussianFilterToImage(8, third_image)
    OCT3GAUS5 = AddGaussianFilterToImage(11.314, third_image)

    cv.imwrite( "octave3_Gaus1.jpg", np.asarray(OCT3GAUS1))
    cv.imwrite( "octave3_Gaus2.jpg", np.asarray(OCT3GAUS2))
    cv.imwrite( "octave3_Gaus3.jpg", np.asarray(OCT3GAUS3))
    cv.imwrite( "octave3_Gaus4.jpg", np.asarray(OCT3GAUS4))
    cv.imwrite( "octave3_Gaus5.jpg", np.asarray(OCT3GAUS5))
    print("Octave 3 Gaussian Completed")

    #These function calls calculate the difference of Gaussian
    #Store it in the respective Identifiers
```

```
    OCT3DOG1 = calculate_DOG(OCT3GAUS1,OCT3GAUS2)
    OCT3DOG2 = calculate_DOG(OCT3GAUS2,OCT3GAUS3)
    OCT3DOG3 = calculate_DOG(OCT3GAUS3,OCT3GAUS4)
    OCT3DOG4 = calculate_DOG(OCT3GAUS4,OCT3GAUS5)
    cv.imwrite( "octave3_dog1.jpg", np.asarray(Normalise_Matrix(OCT3DOG1)))
    cv.imwrite( "octave3_dog2.jpg", np.asarray(Normalise_Matrix(OCT3DOG2)))
    cv.imwrite( "octave3_dog3.jpg", np.asarray(Normalise_Matrix(OCT3DOG3)))
    cv.imwrite( "octave3_dog4.jpg", np.asarray(Normalise_Matrix(OCT3DOG4)))
    print("Octave 3 DoG Completed")

    #This part of code Locates the maxima and minima to find the keypoints
    OCT3Key1 = calculate_keypoints(OCT3DOG1,OCT3DOG2,OCT3DOG3)
    OCT3Key2 = calculate_keypoints(OCT3DOG2,OCT3DOG3,OCT3DOG4)
    cv.imwrite( "octave3_key1.jpg", np.asarray(Normalise_Matrix(OCT3Key1)))
    cv.imwrite( "octave3_key2.jpg", np.asarray(Normalise_Matrix(OCT3Key2)))
    print("Octave 3 Keypoint images Generated")

    #this part of code points the generated key point value in original image
    OCT3FINAL = mark_keypoints(OCT3Key1,OCT3Key2,third_image)
    cv.imwrite("octave3_final.jpg",np.asarray(OCT3FINAL))
    print("Octave 3 final Image generated")

def octave4_gaussian(fourth_image):
    # OCTAVE 4 Code starts here
    # Computing the Matrix for the given original image for various sigma values
    # Sigma values = 5.657,8,11.314,16,22.627
    OCT4GAUS1 = AddGaussianFilterToImage(5.657, fourth_image)
    OCT4GAUS2 = AddGaussianFilterToImage(8, fourth_image)
    OCT4GAUS3 = AddGaussianFilterToImage(11.314, fourth_image)
    OCT4GAUS4 = AddGaussianFilterToImage(16, fourth_image)
    OCT4GAUS5 = AddGaussianFilterToImage(22.627, fourth_image)

    cv.imwrite( "octave4_Gaus1.jpg", np.asarray(OCT4GAUS1))
    cv.imwrite( "octave4_Gaus2.jpg", np.asarray(OCT4GAUS2))
    cv.imwrite( "octave4_Gaus3.jpg", np.asarray(OCT4GAUS3))
    cv.imwrite( "octave4_Gaus4.jpg", np.asarray(OCT4GAUS4))
    cv.imwrite( "octave4_Gaus5.jpg", np.asarray(OCT4GAUS5))
    print("Octave 4 Gaussian Completed")

    #These function calls calculate the difference of Gaussian
    #Store it in the respective Identifiers
    OCT4DOG1 = calculate_DOG(OCT4GAUS1,OCT4GAUS2)
    OCT4DOG2 = calculate_DOG(OCT4GAUS2,OCT4GAUS3)
    OCT4DOG3 = calculate_DOG(OCT4GAUS3,OCT4GAUS4)
    OCT4DOG4 = calculate_DOG(OCT4GAUS4,OCT4GAUS5)
    cv.imwrite( "octave4_dog1.jpg", np.asarray(Normalise_Matrix(OCT4DOG1)))
    cv.imwrite( "octave4_dog2.jpg", np.asarray(Normalise_Matrix(OCT4DOG2)))
    cv.imwrite( "octave4_dog3.jpg", np.asarray(Normalise_Matrix(OCT4DOG3)))
    cv.imwrite( "octave4_dog4.jpg", np.asarray(Normalise_Matrix(OCT4DOG4)))
    print("Octave 4 DoG Completed")

    #This part of code Locates the maxima and minima to find the keypoints
    OCT4Key1 = calculate_keypoints(OCT4DOG1,OCT4DOG2,OCT4DOG3)
    OCT4Key2 = calculate_keypoints(OCT4DOG2,OCT4DOG3,OCT4DOG4)
    cv.imwrite( "octave4_key1.jpg", np.asarray(Normalise_Matrix(OCT4Key1)))
    cv.imwrite( "octave4_key2.jpg", np.asarray(Normalise_Matrix(OCT4Key2)))
    print("Octave 4 Keypoint images Generated")
```

```
    #this part of code points the generated key point value in original image
    OCT4FINAL = mark_keypoints(OCT4Key1,OCT4Key2,fourth_image)
    cv.imwrite("octave4_final.jpg",np.asarray(OCT4FINAL))
    print("Octave 4 final Image generated")

# This is the implementation of SIFT algorithm for key point detection
# This contains First 3 Steps of SIFT Algorithm
# Read the input image in Gray scale format
Color_image = cv.imread('task2.jpg')
INPUT_IMAGE = cv.imread('task2.jpg', cv.IMREAD_GRAYSCALE)
second_image = ScaleDownImage(INPUT_IMAGE)
third_image = ScaleDownImage(second_image)
fourth_image = ScaleDownImage(third_image)

print("Process initiated...")
octave1_gaussian(INPUT_IMAGE)
octave2_gaussian(second_image)
octave3_gaussian(third_image)
octave4_gaussian(fourth_image)
print("process completed")
```

2.5 Output

2.5.1 Input images for Octave 2 and Octave 3

Resolution of Octave 2 is 229 x 375.

Resolution of Octave 3 is 114 x 187.

Octave 2 Gaussian Images

Octave 3 Guassian Images

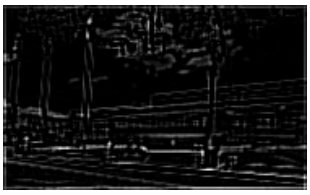


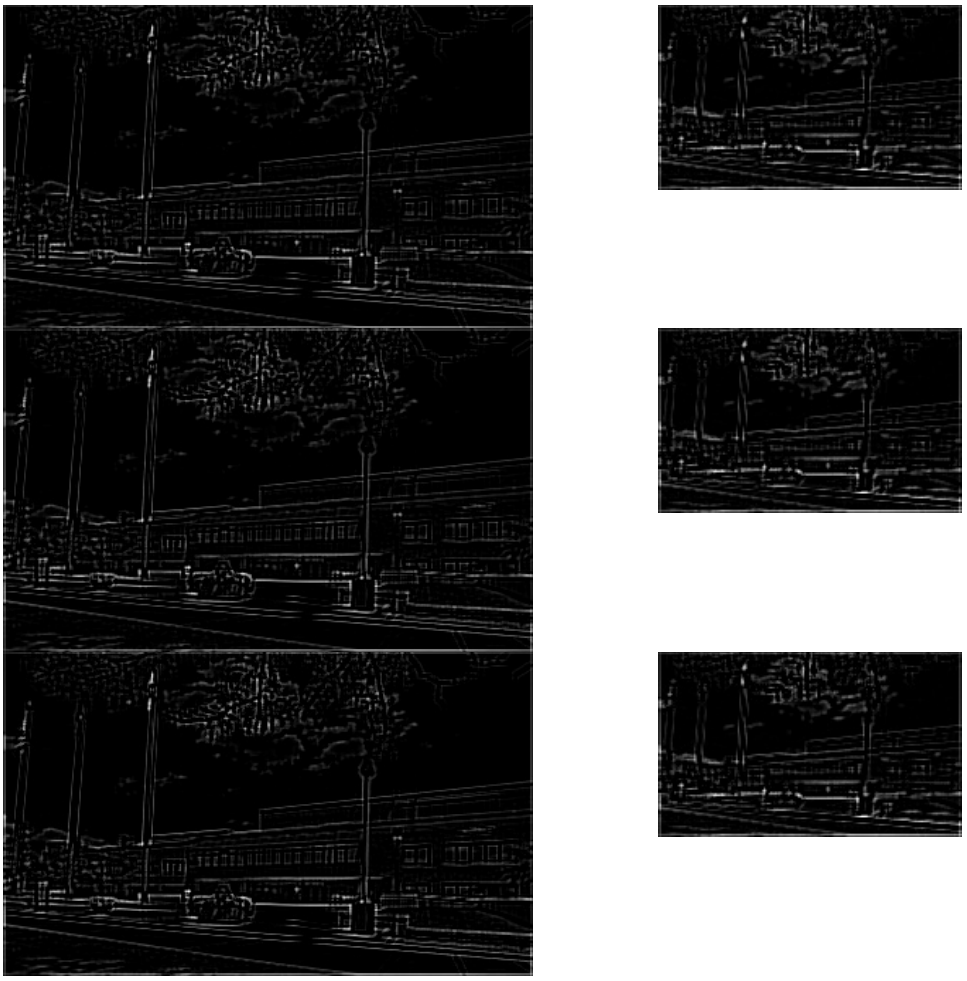



2.5.2 Dog Images Obtained from Octave 2 and Octave 3

DoG Images of Octave 2

DoG Images of Octave 3





2.5.3 Key points plotted on original image

Figure: Octave 1 Keypoints

Figure:Octave 2 Keypoints

Figure:Octave 3 Keypoints

Figure:Octave 4 Keypoints

Figure: Key points plotted on original Image

2.5.4 Coordinates of 5 left most key points

The Left most key points are
(1,1), (1,35), (1,65), (1,79), (1,84)

3 Cursor Detection using Template matching

Template matching is an image processing technique in which two images are compared, one being template image and another is the image in which the template has to be found. There are 6 different methods to match the template. The most commonly used methods are Sum of squared Differences, Cross Correlation, Normalized Cross Correlation and so on. In this task, we use Normalized Cross correlation to match the given template. We need to process the image as well as the template in order for the matching algorithm to work well.

3.1 Task Objective

Set of 25 images and a template image is provided. Task is to read match the template image with the given set of images. And to handle the positive as well as negative scenarios such that if the template is not found it does nothing. If a template is found, it marks the template position in the sample image.

3.2 Normalized cross correlation

Cross correlation is a similarity of two series as a function of the displacement of one relative to another. It is commonly used for searching the long signal for the shorter, i.e the known feature.

Normalized cross correlation uses the below formula

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

3.3 Steps

Before the template is correlated with the given image, it needs to be processed so that the accuracy or the rate of matching would be higher.

1) Both the template and Image is filtered using Gaussian kernel of size(3*3) and bandwidth value of 1.
2) Gaussian kernel reduces the noise in the image and template.
3) After The image is filtered with Gaussian it is then filtered again with Laplacian filter
4) Once the noise is reduced, the edges in the images can be detected using Laplacian Filters.
5) Now the template and Image contains no noise and edges are clearly visible.
6) After this preprocessing step, Template matching is done using Normalized cross correlation.
7) After matching is done, Values such as Threshold, Bandwidth parameters can be tweaked to optimize the performance of the algorithm.

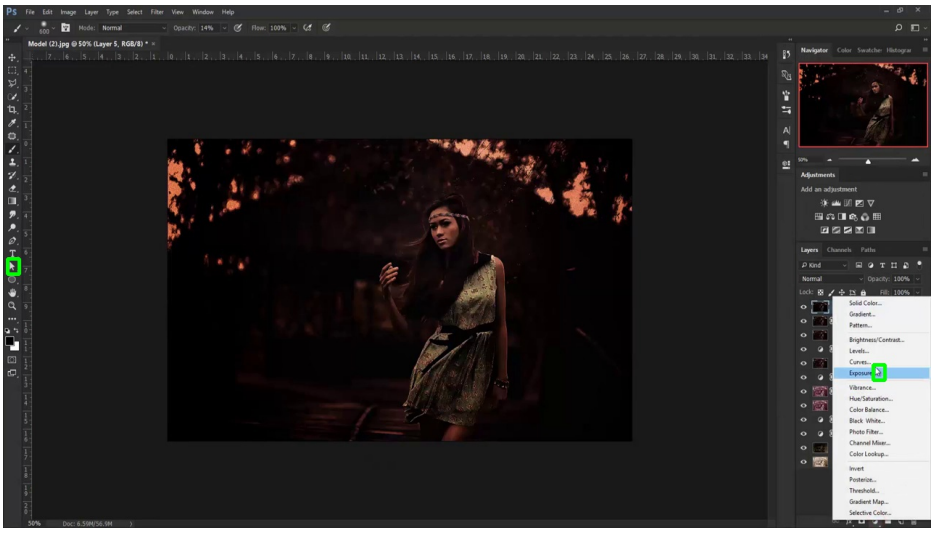
Note: Sobel filter is used before using Laplacian. Commented code can be found in the python file. Since the number of positive matching in Sobel method is way lesser than the current method.

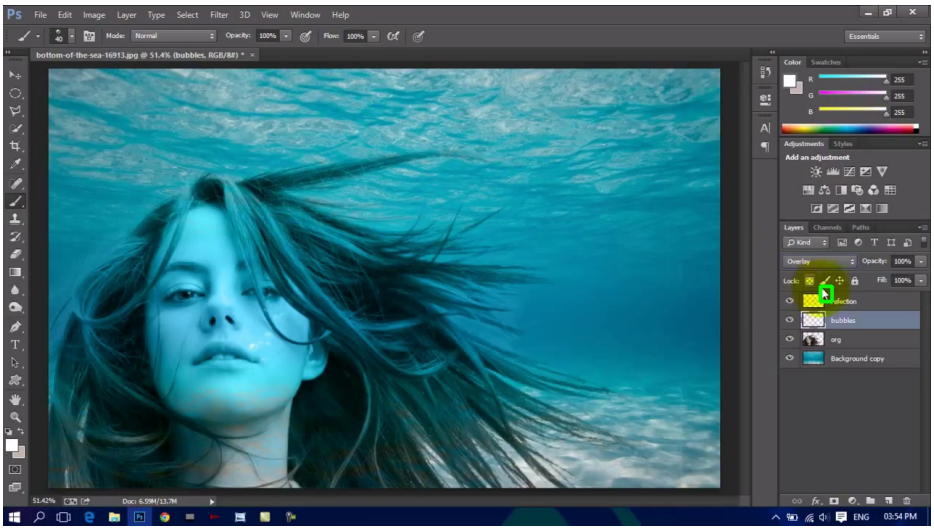
3.4 Code

```
import numpy as np
import cv2 as cv

#This funciton initialises the empty matrix with zero values in it
def initialise_matrix(row, col):
    matrix = [[0 for x in range(col)] for y in range(row)]
    return matrix

#This function is used for applying the sobel filter for given Image
def getSobel(Image):
    sobelx = cv.Sobel(Image, cv.CV_64F, 1, 0, ksize=3)
    sobely = cv.Sobel(Image, cv.CV_64F, 0, 1, ksize=3)
    OP = initialise_matrix(len(sobelx), len(sobelx[0]))
    for i in range(len(sobelx)):
        for j in range(len(sobelx[0])):
            OP[i][j] = (((sobelx[i][j]**2)+(sobely[i][j]**2))**0.5)

    return OP

#this list gets input files
list = ["neg_1","neg_2","neg_3","neg_4","neg_5","neg_6","neg_8","neg_9","neg_10","pos_1",
"pos_2","pos_3","pos_4","pos_5","pos_6","pos_7","pos_8","pos_9","pos_10","pos_11","pos_12","pos_13"
,"pos_14","pos_15"]

template = cv.imread("task3/template.png", cv.IMREAD_GRAYSCALE)
```

```
resize_template = cv.resize(template, None, fx=0.55, fy=0.55)
blur_template = cv.GaussianBlur(resize_template, (3,3), 0)

for filename in list:
    file = "./task3/"+filename+".jpg"
    img = cv.imread(file)
    gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    w, h = resize_template.shape[::-1]
    blurredimage = cv.GaussianBlur(gray_img, (3,3), 0)
    # cv.imshow("gray",gray_img)
    # cv.imshow("gauss_blur",blurredimage)
    # cv.imshow("gauss_blur_template",blur_template)

    laplacian = cv.Laplacian(blurredimage, 10)
    lap_template = cv.Laplacian(blur_template, 10)
    #cv.imshow("laplacian", laplacian)
    #cv.imshow("lap_template",lap_template)

    #sobelImage = getSobel(gray_img)
    #sobelTemplate = getSobel(blur_template)
    #cv.imwrite("t_sample.jpg", np.asarray(sobelImage))
    #cv.imwrite("template_sample.jpg", np.asarray(sobelTemplate))
    result = cv.matchTemplate(np.asarray(laplacian).astype(np.float32),
np.asarray(lap_template).astype(np.float32), cv.TM_CCOEFF_NORMED)
    loc = np.where(result >= 0.55)
    cv.imshow("result", result)

    for pt in zip(*loc[::-1]):
        cv.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0, 255, 0), 3)
    fileop = filename+"_op.jpg"
    cv.imwrite(fileop,img)
```

3.4 Output





4 Conclusion

All the three tasks implemented in the project provided the intuitive understanding of the image processing techniques such as Edge Detection, Template matching and many algorithms used in the process. The practical implementation without using predefined libraries helped in understanding the actual logistics behind the work.

References

[1] Richard Szeliski (2010), Computer Vision: Algorithms and Applications

[2] https://docs.opencv.org/2.4/doc/tutorials/

[3] https://docs.python.org/3/