CSE 573: Introduction to Computer Vision and Image Processing
(Fall 2018)
Instructor: Junsong Yuan

Project 1

October 8, 2018

Report By:
Siddheswar Chandrasekhar

Objective

The objective is to perform three independent tasks: Edge Detection, Keypoint Detection and Template Matching.

Task 1: Edge Detection

Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. [1]

In this task of our project, we use Sobel operator to detect edges.

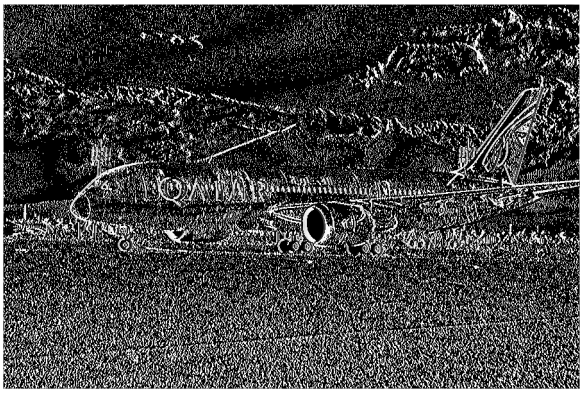
The Sobel Operator is an isotropic 3x3 image gradient operator. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. Sobel is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image. [2]

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \qquad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

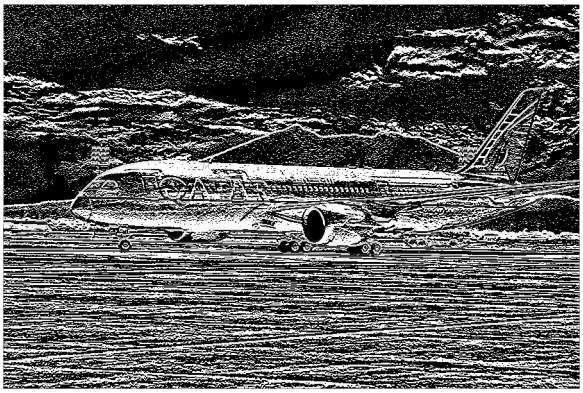
We have our image:

Fig 1.1 Original Image

When we convolve the Sobel operator with our image to find the edges along the x and y axis, we get the output as follows:



Gx



Gy

Fig 1.2 Output of Sobel Operator

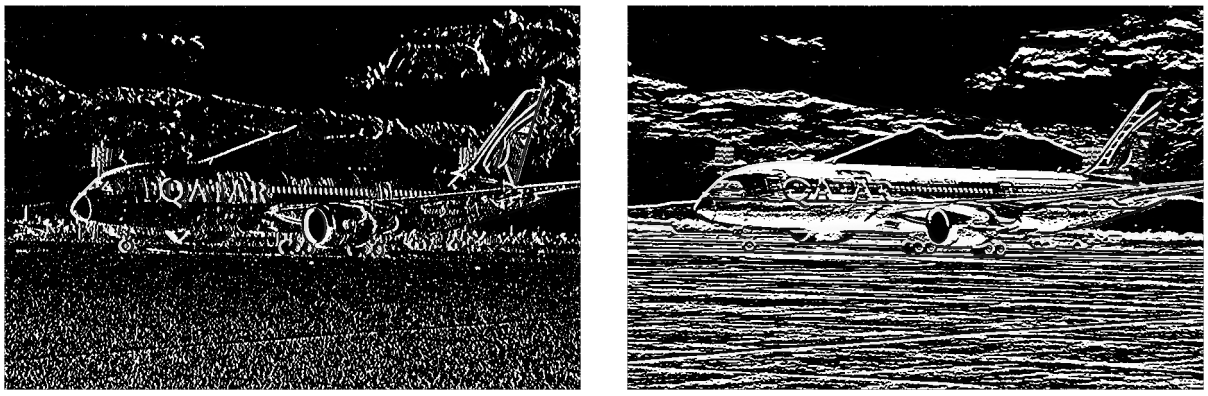
We notice that we get an edge image with a lot of 'Salt and Pepper' noise. Hence, we try to reduce the effect of this noise on our edge image by using a smoothing filter on our original image.

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Fig 1.3 Averaging Filter

We apply the averaging filter to our original image before convoluting it with the Sobel operator.

We then convolve the smoothed image with the Sobel operator and get our edges along the x and y axis as follows:



Gx Gy

Fig 1.4 Output of Sobel Operator on Smoothed Image

We see that this is a much sharper and noise-free image compared to the Sobel operator being convolved with the image directly.

Hence, the steps for Task 1 of our project are:

1. Read the input image as grayscale
2. Smooth the input image with an Averaging Filter
3. Convolve with the Sobel Operator along the x and y axis

Code

```
import cv2

img = cv2.imread('/Users/siddheswarc/Desktop/UB/CSE 573/Projects/Project 1/resources/task1.png', 0)
smooth_img = cv2.imread('/Users/siddheswarc/Desktop/UB/CSE 573/Projects/Project 1/resources/task1.png', 0)
gx = cv2.imread('/Users/siddheswarc/Desktop/UB/CSE 573/Projects/Project 1/resources/task1.png', 0)
gy = cv2.imread('/Users/siddheswarc/Desktop/UB/CSE 573/Projects/Project 1/resources/task1.png', 0)

# SMOOTHING
for x in range(1, img.shape[0] - 1):
    for y in range(1, img.shape[1] - 1):
        smooth_img[x][y] = (((img[x-1][y-1] * 1) / 9) + ((img[x][y-1] * 1) / 9) + ((img[x+1][y-1] * 1) / 9) +
                ((img[x-1][y] * 1) / 9) + ((img[x][y] * 1) / 9) + ((img[x+1][y] * 1) / 9) +
                ((img[x-1][y+1] * 1) / 9) + ((img[x][y+1] * 1) / 9) + ((img[x+1][y+1] * 1) / 9))

# Filtering with Sobel along the x direction
for x in range(1, img.shape[0] - 1):
    for y in range(1, img.shape[1] - 1):
        gx[x][y] = ((smooth_img[x-1][y-1] * -1) + (smooth_img[x][y-1] * -2) + (smooth_img[x+1][y-1] * -1) +
                (smooth_img[x-1][y] * 0) + (smooth_img[x][y] * 0) + (smooth_img[x+1][y] * 0) +
                (smooth_img[x-1][y+1] * 1) + (smooth_img[x][y+1] * 2) + (smooth_img[x+1][y+1] * 1)) / 8

#Filtering with Sobel along the y direction
for x in range(1, img.shape[0] - 1):
    for y in range(1, img.shape[1] - 1):
        gy[x][y] = ((smooth_img[x-1][y-1] * -1) + (smooth_img[x][y-1] * 0) + (smooth_img[x+1][y-1] * 1) +
                (smooth_img[x-1][y] * -2) + (smooth_img[x][y] * 0) + (smooth_img[x+1][y] * 2) +
                (smooth_img[x-1][y+1] * -1) + (smooth_img[x][y+1] * 0) + (smooth_img[x+1][y+1] * 1)) / 8

cv2.imwrite('Gx.png', gx)
cv2.imwrite('Gy.png', gy)
```

Task 2: Keypoint Detection

Feature Detection includes methods for computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not. The resulting features will be subsets of the image domain, often in the form of isolated points, continuous curves or connected regions. [3]

Types of image features:

- Edges
- Corners / interest points
- Blobs / regions of interest points
- Ridges

The steps for Task 2 of our project are:

1. Convolve four octaves with five gaussian filters each of different sigmas

2. Find Difference of Gaussian (DoG) for each octave
   For eg. Each octave will have D12, D23, D34, D45

3. Make two sets of three DoGs each
   For e.g. set 1 = D12, D23, D34 set 2 = D23, D34, D45

4. Find keypoints in each set by finding minimum and maximum values with respect to the center pixel of a 3x3 kernel

These steps form the basis of Scale-Invariant Feature Transform (SIFT). SIFT is a feature detection algorithm in computer vision to detect and describe local features in images.

SIFT keypoints of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. From the full set of matches, subsets of keypoints that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches. The determination of consistent clusters is performed rapidly by using an efficient hash table implementation of the generalised Hough transform. Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed model verification and subsequently outliers are discarded. Finally the probability that a particular set of features indicates the presence of an object is computed, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence. [4]

The image of the second octave is as follows:

Fig 2.1 Octave 2 (375 x 229)

The convolution of the second octave with respective sigmas are as follows:

Octave 2 Sigma 1

Octave 2 Sigma 2

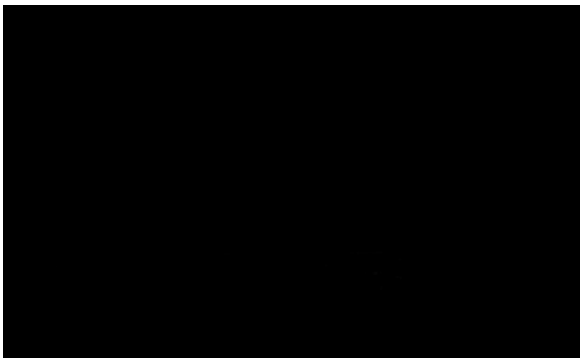
Octave 2 Sigma 3

Octave 2 Sigma 4

Octave 2 Sigma 5

Fig 2.2 Octave 2 Sigmas

The DoG images of the second octave are:



D12

D23

D34

D45

Fig 2.3 DoG Images of Octave 2

The image of the third octave is as follows:

Fig 2.4 Octave 3 (188 x 115)

The convolution of the third octave with respective sigmas are as follows:

Octave 3 Sigma 1

Octave 3 Sigma 2

Octave 3 Sigma 3

Octave 3 Sigma 4

Octave 3 Sigma 5

Fig 2.5 Octave 3 Sigmas

The DoG images of the third octave are as follows:

D12

D23

D34

D45

Fig 2.6 DoG Images of Octave 3

The keypoints detected in the image are shown using white dots. These keypoints are:

Fig 2.7 Octave 1 Keypoint Pair 1

Fig 2.8 Octave 1 Keypoint Pair 2

Fig 2.9 Octave 2 Keypoint Pair 1

Fig 2.10 Octave 2 Keypoint Pair 2

Octave 3 Keypoint Pair 1

Octave 3 Keypoint Pair 2

Fig 2.11 Octave 3 Keypoint Pairs

Octave 4 Keypoint Pair 1

Octave 4 Keypoint Pair 2

Fig 2.12 Octave 4 Keypoint Pairs

Code

```
import cv2
import math
import numpy as np


def flip(img):
    image = np.ones([img.shape[0], img.shape[1]])
    for x in range (img.shape[0]):
        for y in range (img.shape[1]):
            image[x][y] = img[6-x][6-y]
    return image


def resize_colored_octave(octave):
    resized_octave = np.zeros((math.ceil(octave.shape[0]/2) , math.ceil(octave.shape[1]/2) , math.ceil(octave.shape[2]
)))
    for c in range (3):
        for x in range (0, octave.shape[0], 2):
            for y in range (0, octave.shape[1], 2):
                resized_octave[math.ceil(x/2)][math.ceil(y/2)][c] = octave[x][y][c]
    return resized_octave


def resize_grayscale_octave(octave):
    resized_octave = np.zeros((math.ceil(octave.shape[0]/2) , math.ceil(octave.shape[1]/2)))
    for x in range (0, octave.shape[0], 2):
        for y in range (0, octave.shape[1], 2):
            resized_octave[math.ceil(x/2)][math.ceil(y/2)] = octave[x][y]
    return resized_octave


def gaussian_generator(sigma):
    g = np.zeros((7,7))
    for x in range (-3, 4):
        for y in range (3, -4, -1):
            temp = -((x ** 2) + (y ** 2))
```

```
            temp2 = temp / (2 * (sigma ** 2))
            exp = math.exp(temp2)
            fin = exp / (2 * math.pi * (sigma ** 2))
            g[x][y] = fin
    return g

def convolve(f, g):

    conv_img = np.zeros((f.shape[0], f.shape[1]))
    for x in range (3, f.shape[0] - 3):
        for y in range (3, f.shape[1] - 3):
            conv_img[x][y] = mat_mul(f[x-3:x+4,y-3:y+4], g)
    return conv_img

def mat_mul(image1, image2):
    sum = 0
    for x in range (image2.shape[0]):
        for y in range (image2.shape[1]):
            sum = sum + (image1[x][y] * image2[x][y])
    return sum

def is_distinguishable(mat1, mat2, mat3):
    lst = []
    for x in range (mat2.shape[0]):
        for y in range (mat2.shape[1]):
            lst.append(mat1[x][y])
            lst.append(mat2[x][y])
            lst.append(mat3[x][y])

    if (min(lst) == mat2[1][1] or max(lst) == mat2[1][1]):
        return True

def key_point_detection(octave, lst):
    octcopy1 = octave
```

```
    octcopy2 = octave

    set1 = [lst[0], lst[1], lst[2]]
    set2 = [lst[1], lst[2], lst[3]]

    for x in range (1, lst[1].shape[0]-1):
        for y in range (1, lst[1].shape[1]-1):
            if (is_distinguishable(lst[0][x-1:x+2,y-1:y+2], lst[1][x-1:x+2,y-1:y+2], lst[2][x-1:x+2,y-1:y+2])):
                octcopy1[x][y] = 255

    for x in range (1, lst[2].shape[0]-1):
        for y in range (1, lst[2].shape[1]-1):
            if (is_distinguishable(lst[1][x-1:x+2,y-1:y+2], lst[2][x-1:x+2,y-1:y+2], lst[3][x-1:x+2,y-1:y+2])):
                octcopy2[x][y] = 255

    return octcopy1, octcopy2



def diff_of_gauss(lst):
    dog = []
    for i in range (len(lst)-1):
        dog.append(lst[i+1] - lst[i])

    return dog

def task2():

    oct1c = cv2.imread('/Users/siddheswarc/Desktop/UB/CSE 573/Projects/Project 1/resources/task2.jpg')
    oct1g = cv2.imread('/Users/siddheswarc/Desktop/UB/CSE 573/Projects/Project 1/resources/task2.jpg', 0)

    oct2g = resize_grayscale_octave(oct1g)
    oct3g = resize_grayscale_octave(oct2g)
    oct4g = resize_grayscale_octave(oct3g)
```

```
octave_grayscale = []
octave_grayscale.append(oct1g)
octave_grayscale.append(oct2g)
octave_grayscale.append(oct3g)
octave_grayscale.append(oct4g)

oct2c = resize_colored_octave(oct1c)
oct3c = resize_colored_octave(oct2c)
oct4c = resize_colored_octave(oct3c)

octave_colored = []
octave_colored.append(oct1c)
octave_colored.append(oct2c)
octave_colored.append(oct3c)
octave_colored.append(oct4c)

sigma = [[1 / math.sqrt(2)], [1], [math.sqrt(2)], [2], [2 * math.sqrt(2)],\
      [math.sqrt(2)], [2], [2 * math.sqrt(2)], [4], [4 * math.sqrt(2)],\
      [2 * math.sqrt(2)], [4], [4 * math.sqrt(2)], [8], [8 * math.sqrt(2)],\
      [4 * math.sqrt(2)], [8], [8 * math.sqrt(2)], [16], [16 * math.sqrt(2)]]

sigma = np.array(sigma).reshape(4,5)

for i in range (len(octave_grayscale)):
   lst = []

   for j in range (5):
      g = np.zeros((7,7))
      conv_img = np.zeros((octave_grayscale[i].shape[0], octave_grayscale[i].shape[1]))
      g = gaussian_generator(sigma[i][j])
      gflip = flip(g)
      conv_img = convolve(octave_grayscale[i], gflip)
      lst.append(conv_img)
      cv2.imwrite('octave'+str(i+1)+'sigma'+str(j+1)+'.png', conv_img)
```

```
        dog = diff_of_gauss(lst)

        for k in range (len(dog)):
            cv2.imwrite('octave'+str(i+1)+'DoG'+str(k+1)+str(k+2)+'.png', dog[k])

        kpd1, kpd2 = key_point_detection(octave_colored[i], dog)

        cv2.imwrite('octave'+str(i+1)+'KeyPointDetection1.png', kpd1)
        cv2.imwrite('octave'+str(i+1)+'KeyPointDetection2.png', kpd2)



if __name__ == '__main__':
    task2()
```

Task 3: Cursor Detection

Template matching is a technique in digital image processing for finding small parts of an image which match a template image. [5]

cv2.matchTemplate() is a built-in function of OpenCV for template matching. However, for our task, since the template is not directly cut-out from the image, we cannot use the function directly. We need to make several modifications to the image and the template before using the *cv2.matchTemplate()* function to get desired output.

We first use a Gaussian blurring function *cv2.GaussianBlur()* to smooth the image and the template. The blurring function essentially smoothens the edges in the image making the difference between neighboring pixels smaller.

We then use Laplacian function *cv2.Laplacian()* to make the edges in the image and the template stand out. This helps in distinguishing the geometric features of elements in the image. Hence, we are able to easily match features of the image and the template when we run the template through the image as a filter.

We finally use the *cv2.matchTemplate()* function to match the template with the image.

The function looks like:

cv2.matchTemplate(image, template, result, method)

Parameters: [6]

- image: image where the search is running
- template: template that is being searched for in the image
- result: map of comparison results
- method: parameter specifying the comparison method

In our task, we use Normalized Cross-Correlation and have the result mapped as 255

The *cv2.matchTemplate()* function returns a gaussian which is represented by four parameters. Essentially, the function creates a gaussian based on the similarity between the template and the image at that pixel location. Bright spots indicate a match. Brighter the spot, closer the match.

The four parameters are: *min_val, max_val, min_loc and max_loc*

min_val and *max_val* are the minimum and maximum values of the gaussian.
Min_loc and *max_loc* are the locations of the minimum and maximum values.

Code [7]

```
import cv2
import numpy as np

def task3():

    images = ["resources/task3/pos_2.jpg",
              "resources/task3/pos_7.jpg",
              "resources/task3/pos_14.jpg"]
    template = cv2.imread('resources/task3/template.png',0)

    for i in range (len(images)):
        image = cv2.imread(images[i], 0)

        image1 = cv2.GaussianBlur(image,( 3, 3 ), 2)
        image1 = cv2.Laplacian(image1,cv2.CV_8U)
        ret, image1 = cv2.threshold(image1, 24, 255.0, cv2.THRESH_BINARY)
        image1 = cv2.GaussianBlur(image1,(3,3),1)

        temp = cv2.GaussianBlur(template,(3,3),2)
        temp = cv2.Laplacian(temp, cv2.CV_8U)
        ret, image1 = cv2.threshold(image1, 30, 255.0, cv2.THRESH_BINARY)
        temp = cv2.GaussianBlur(temp,(3,3),1)

        temp = cv2.resize(temp, (0, 0), fx=0.6, fy=0.6)

        result = cv2.matchTemplate(image1,temp, cv2.TM_CCORR_NORMED)

        width = temp.shape[0]
        height = temp.shape[1]

        threshold = 0.75
```

```
        loc = np.where(result>= threshold)
        for pt in zip(*loc[::-1]):
            cv2.rectangle(image, pt, (pt[0] + width, pt[1] + height), (255,255,255),2)

        cv2.imwrite('task3test'+str(i+1)+'.png', image)

if __name__ == '__main__':
    task3()
```

Output

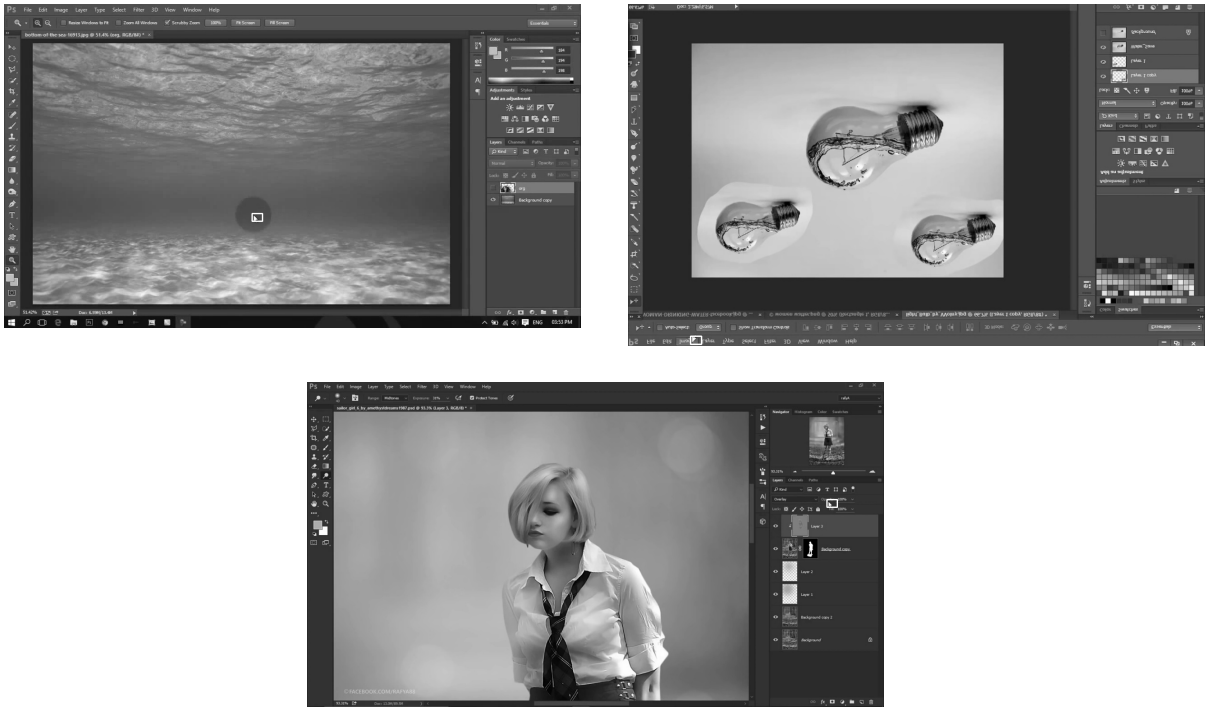


Fig 3.1 Template Matching

References:

1. "Edge Detection"
   https://en.wikipedia.org/wiki/Edge_detection

2. "Sobel Operator"
   https://en.wikipedia.org/wiki/Sobel_operator

3. "Feature Detection"
   https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)

4. "Scale-Invariant Feature Transform"
   https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

5. "Template Matching"
   https://en.wikipedia.org/wiki/Template_matching

6. "Object Detection"
   https://docs.opencv.org/2.4/modules/imgproc/doc/object_detection.html

7. "Template Matching using OpenCV in Python"
   https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/