# CS 532 Homework 3

**Part1**

In this part, we find the x and the y derivatives of the Images, I left and I right. For this, we first define a kernel for each direction i.e. the x kernel and the y kernel. These kernels are then convoluted with the original images to get Ix and Iy. Ixx was found by convolving Ix again with the x kernel and Iyy similarly with y kernel.

**Part2**

Gaussian smoothing is applied on a 5x5 window of the obtained images, i.e. Ix, Iy, Ixx, Iyy and smoothed images are obtained. We do this by convolving the Gaussian kernel with the images. Gaussian kernel formula for width N was used from the implementation of 2D homework 1.

**Part3**

The above smoothed images will give us a matrix called moment matrix defined by,

$$M = \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix}$$

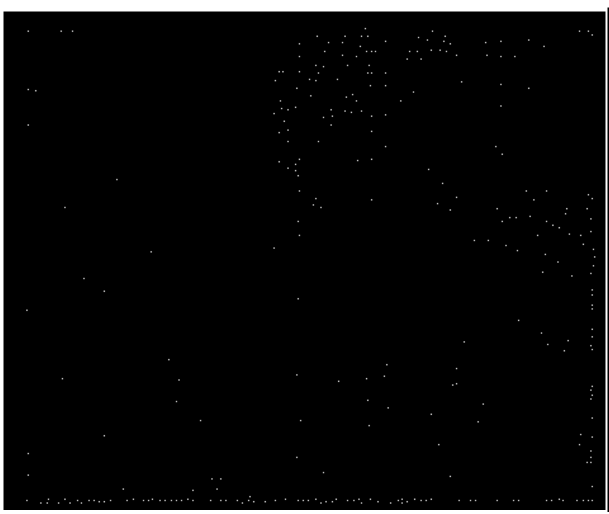
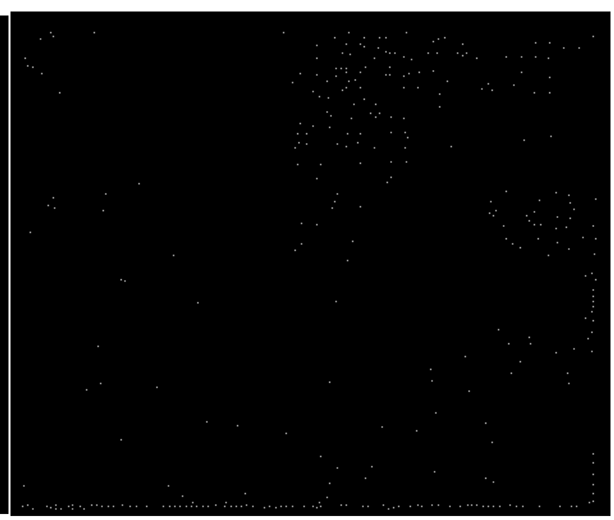
This is then used in the following formula of Response, given by,

$$R = \det(M) - \alpha \operatorname{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

We get corners using the Response function R.

**Part 4**

Non Max Suppression is applied on the received corner image, i.e. a 3x3 window is defined around each corner and the maximum amongst all the 9 values is retained whereras others are suppressed to zero. This reduces the number of extra white corners which are unneeded. The following images show Non Max Suppression used on the left and right corner detected image.

**Part 5**

SAD is applied on the above images by keeping the left image as the reference and taking SAD over a 3x3 window on each of the corners on the right image. 323(left image corners) x 293(right image corners) are the total number of SAD values in the set.

**Part 6**

Top x percent(minimum SADs) of the given SAD location values are used for finding out the correct correspondences using the ground truth map given(x increases with a step of 5 percent). This is done by finding the difference between the addition of disparity value at a given location on the reference image i.e. (row_l, col_l) and col_l i.e. (row_l, col_l + disparity), and the location col_r of the right image, i.e (col_l + disparity – col_r). If the disparity closely matches i.e. if it is equal or less or greater than 1, the correspondences match, else not.

CODE (Python):

```
from PIL import Image

import numpy as np

import copy

import math


def padwithzeros(vector, pad_width, iaxis, kwargs):

    vector[:pad_width[0]] = 0

    vector[-pad_width[1]:] = 0

    return vector


def convolution(image, kernel, kernel_width):

    width, height = image.size

#    print(width, height)

    image = np.asarray(image)


    e = -1; f = -1; summation = 0

#    img_op = [[]]

    img_op = copy.copy(image)

    img_op = np.int32( np.uint32(img_op) )

#    print(img_op)


    image = np.lib.pad(image, 1, padwithzeros)

#    print(image)


    kernel_flip = np.flipud(kernel)

    kernel_flip = np.fliplr(kernel_flip)
```

```
    for i in range(0,height-(kernel_width-1)):
        for j in range(0, width-(kernel_width-1)):
            for k in range(i,i+kernel_width):
                e += 1
                for l in range(j,j+kernel_width):
                    f += 1
                    summation = summation + kernel_flip[e][f]*image[k][l]
                f = -1
            img_op[i][j] = summation
            e = -1
            summation = 0
#    print(img_op)
    return img_op


def Gaussian(image, N, sigma):
    ind = range(-(math.floor(N/2)),math.floor(N/2)+1)
    xvalues = yvalues = np.array([ind])
    X, Y = np.meshgrid(xvalues, yvalues)
    h = np.exp(-(np.square(X) + np.square(Y)) / (2*np.square(sigma)))
    gaussian_kernel = h/h.sum()
#    print(gaussian_kernel)
#    print(gaussian_kernel.sum())
    return convolution(image,gaussian_kernel, N)


def Harris_threshold(image):
    width, height = image.size;
    image = np.asarray(image)
    img_op = copy.copy(image)
    img_op = np.int32( np.uint32(img_op) )
    for i in range(0,height):
        for j in range(0,width):
            if image[i][j] < 5000:
                img_op[i][j] = 0
```

```
    return img_op

def Non_max_suppression(image):
    width, height = image.size; maximum = 0; ind_x = 0; ind_y = 0
    print(width, height)
    image = np.asarray(image)
    img_op = copy.copy(image)
    img_op = np.int32( np.uint32(img_op) )
    for i in range(0,height-2):
        for j in range(0,width-2):
            for k in range(i,i+3):
                for l in range(j,j+3):
                    if image[k][l] > maximum:
                        img_op[ind_x][ind_y] = 0
                        ind_x = k; ind_y = l
                    else:
                        img_op[k][l] = 0
            ind_x = 0; ind_y = 0
    return img_op

def corner_count(image):
    corner_count = 0; width, height = image.size;
    image = np.asarray(image)
    for i in range(0,height):
        for j in range(0,width):
            if image[i][j] > 0:
                corner_count += 1



    print(corner_count)
    return corner_count
```

```
def SAD(Left_image, Right_image, left_corner_count, right_corner_count, percent):
    width, height = Right_image.size;
#    Distance_array = [[] for i in range(right_corner_count)]
    Distance_array_top_corr = []; Distance_array = [] ; SAD = 0; Left_image = np.asarray(Left_image); Right_image = np.asarray(Right_image); index = -1
    Left_image = np.lib.pad(Left_image, 1, padwithzeros); Right_image = np.lib.pad(Right_image, 1, padwithzeros)
    for i in range(0,height+2):
        for j in range(0,width+2):
            if Left_image[i][j] > 0:
                index += 1
                e = i-2; f = j-2
                for k in range(0,height+2):
                    for l in range(0,width+2):
                        if Right_image[k][l] > 0:
#                            corner_left_xy[][].append(k)
                            for m in range(k-1,k+2):
                                e += 1
                                for n in range(l-1,l+2):
                                    f += 1
                                    SAD = SAD + abs(Left_image[e][f] - Right_image[m][n])
                                f = j-2
                            Distance_array.append([SAD, [k,l], [i,j]])
                            SAD = 0; e = i-2;
    correspondence_percent = percent/100
    for i in range(left_corner_count):
        Distance_array = sorted(Distance_array)
#        print(Distance_array)
    Distance_array_top_corr = Distance_array[0:round(correspondence_percent*len(Distance_array))]


    return Distance_array_top_corr


def Comparison_GT(Distance_array_top_corr, I_GT):
    I_GT = np.asarray(I_GT)
    correct_count = 0
```

```
    for i in range(len(Distance_array_top_corr)):
#       if abs(I_GT[Distance_array_top_corr[i][2][0]-1][Distance_array_top_corr[i][2][1]-1] -
abs((Distance_array_top_corr[i][1][1]-1) - (Distance_array_top_corr[i][2][1]-1)) ) <= 1:
#           correct_count += 1
        if abs((I_GT[Distance_array_top_corr[i][1][0]-1][Distance_array_top_corr[i][1][1]-1] + [Distance_array_top_corr[i][1][1]-
1]) - [Distance_array_top_corr[i][2][1]-1] ) <= 1:
            correct_count += 1


    print(correct_count)
    correct_count_percentage = (correct_count/len(Distance_array_top_corr))*100


    return correct_count_percentage


I_left = Image.open("C:/Users/Jolton/Desktop/teddyL.pgm")
I_left.show()
I_right = Image.open("C:/Users/Jolton/Desktop/teddyR.pgm")
I_right.show()
I_GT = Image.open("C:/Users/Jolton/Desktop/disp2.pgm")


kernel_x = [[-1, 0, 1],[-1, 0, 1],[-1, 0, 1]]
kernel_y = np.transpose(kernel_x)



## LEFT IMAGE Ixx, Iyy and Ixy
Ix_left = convolution(I_left, kernel_x, 3)
Ix_left_image = Image.fromarray(Ix_left)
#Ix_left_image.show()


Iy_left = convolution(I_left, kernel_y, 3)
Iy_left_image = Image.fromarray(Iy_left)
#Iy_left_image.show()


Ixx_left = convolution(Ix_left_image, kernel_x, 3)
#Ixx_left = np.multiply(Ix_left,Ix_left)
Ixx_left_image = Image.fromarray(Ixx_left)
```

```
#Ixx_left_image.show()

Iyy_left = convolution(Iy_left_image, kernel_y, 3)
#Iyy_left = np.multiply(Iy_left,Iy_left)
Iyy_left_image = Image.fromarray(Iyy_left)

Ixy_left = np.multiply(Ix_left,Iy_left)
Ixy_left_image = Image.fromarray(Ixy_left)
#Iyy_left_image.show()



## RIGHT IMAGE Ixx, Iyy and Ixy
Ix_right = convolution(I_right, kernel_x, 3)
Ix_right_image = Image.fromarray(Ix_right)
Ix_right_image.save = ("Ix_right_image.png")
#Ix_right_image.show()

Iy_right = convolution(I_right, kernel_y, 3)
Iy_right_image = Image.fromarray(Iy_right)
#Iy_right_image.show()

Ixx_right = convolution(Ix_right_image, kernel_x, 3)
#Ixx_right = np.multiply(Ix_right, Ix_right)
Ixx_right_image = Image.fromarray(Ixx_right)
Ixx_right_image.save = ("Ixx_right_image.png")

#Ixx_right_image.show()

Iyy_right = convolution(Iy_right_image, kernel_y, 3)
#Iyy_right = np.multiply(Iy_right, Iy_right)
Iyy_right_image = Image.fromarray(Iyy_right)
#Iyy_right_image.show()
```

```
Ixy_right = np.multiply(Ix_right,Iy_right)

Ixy_right_image = Image.fromarray(Ixy_right)


## LEFT IMAGE SMOOTHING

Gaussian_smooth_Ix_left = Gaussian(Ix_left_image, 5, 3)

Gaussian_smooth_Iy_left = Gaussian(Iy_left_image, 5, 3)

Gaussian_smooth_Ixx_left = Gaussian(Ixx_left_image, 5, 3)

Gaussian_smooth_Iyy_left = Gaussian(Iyy_left_image, 5, 3)

Gaussian_smooth_Ixy_left = Gaussian(Ixy_left_image, 5, 3)


Gaussian_smooth_Ixx_left_image = Image.fromarray(Gaussian_smooth_Ixx_left)

Gaussian_smooth_Iyy_left_image = Image.fromarray(Gaussian_smooth_Iyy_left)

#Gaussian_smooth_Ixy_left_image = Image.fromarray(Gaussian_smooth_Ixy_left)


## RIGHT IMAGE SMOOTHING

Gaussian_smooth_Ix_right = Gaussian(Ix_right_image, 5, 3)

Gaussian_smooth_Iy_right = Gaussian(Iy_right_image, 5, 3)

Gaussian_smooth_Ixx_right = Gaussian(Ixx_right_image, 5, 3)

Gaussian_smooth_Iyy_right = Gaussian(Iyy_right_image, 5, 3)

Gaussian_smooth_Ixy_right = Gaussian(Ixy_right_image, 5, 3)


Gaussian_smooth_Ixx_right_image = Image.fromarray(Gaussian_smooth_Ixx_right)

Gaussian_smooth_Iyy_right_image = Image.fromarray(Gaussian_smooth_Iyy_right)

#Gaussian_smooth_Ixy_right_image = Image.fromarray(Gaussian_smooth_Ixy_right)

Gaussian_smooth_Iyy_right_image.save("Gaussian_smooth_Iyy_right_image.png")



## lEFT IMAGE HARRIS

#Harris_operator_response_left = np.multiply(Ixx_left,Iyy_left)-np.square(Ixy_left)

Harris_operator_response_left = (np.multiply(Gaussian_smooth_Ixx_left,Gaussian_smooth_Iyy_left)-
np.square(Gaussian_smooth_Ixy_left)) - (0.05*(np.square(np.add(Gaussian_smooth_Ixx_left,Gaussian_smooth_Iyy_left))))

Harris_image_left = Image.fromarray(Harris_operator_response_left)
```

```
#Harris_image_left.show()

Harris_image_left_thresholded = Image.fromarray(Harris_threshold(Harris_image_left))
Harris_image_left_thresholded.save("Harris_image_left_thresholded.png")
#Harris_image_left_thresholded.show()



Harris_image_left_non_max_suppressed = Image.fromarray(Non_max_suppression(Harris_image_left_thresholded))
Harris_image_left_non_max_suppressed.show()
Harris_image_left_non_max_suppressed.save("Harris_left_supp.png")

left_corner_count = corner_count(Harris_image_left_non_max_suppressed)



## RIGHT IMAGE HARRIS
#Harris_operator_response_right = np.multiply(Ixx_right,Iyy_right)-np.square(Ixy_right)
Harris_operator_response_right = (np.multiply(Gaussian_smooth_Ixx_right,Gaussian_smooth_Iyy_right)-
np.square(Gaussian_smooth_Ixy_right)) - (0.05* (np.square(np.add(Gaussian_smooth_Ixx_right,Gaussian_smooth_Iyy_right))))
Harris_image_right = Image.fromarray(Harris_operator_response_right)
#Harris_image_right.show()

Harris_image_right_thresholded = Image.fromarray(Harris_threshold(Harris_image_right))
#Harris_image_right_thresholded.show()

Harris_image_right_non_max_suppressed = Image.fromarray(Non_max_suppression(Harris_image_right_thresholded))
Harris_image_right_non_max_suppressed.show()
Harris_image_right_non_max_suppressed.save("Harris_right_supp.png")

right_corner_count = corner_count(Harris_image_right_non_max_suppressed)



Distance_array_top_corr = SAD(Harris_image_left_non_max_suppressed, Harris_image_right_non_max_suppressed,
left_corner_count, right_corner_count, 65)
correct_count_percentage = Comparison_GT(Distance_array_top_corr, I_GT)
print("Correct count percentage is ", correct_count_percentage)
```

## OUTPUT:

The correct count keeps decreasing as we increase the percentage of the correspondences to be included to find the correctness increases.

RESULTS:

Top 5% -> 1.3736263736263736%, match count = 65

Top 10% -> 1.246830092983939%, match count = 118

Top 15% -> 1.2256973795435333%, match count = 174

Top 20% -> 1.2151310228233305%, match count = 230

Top 25% -> 1.1918850380388841%, match count = 282

Top 30% -> 1.2080867850098618%, match count = 343

Top 35% -> 1.186450911725637%, match count = 393

Top 40% -> 1.1570160608622146%, match count = 438

Top 45% -> 1.1482107635953789%, match count = 489

Top 50% -> 1.1411665257819104%, match count = 540

Top 55% -> 1.1142917523198401%, match count = 580

Top 60% -> 1.093637180141944%, match count = 621

Top 65% -> 1.0826627651792244%, match count = 666

Top 70% -> 1.065708635862756%, match count = 706

Top 75% -> 1.0552416912044407%, match count = 749

Top 80% -> 1.046083131909498%, match count = 792

Top 85% -> 1.0504332260109641%, match count = 845

Top 90% -> 1.0402113296154976%, match count = 886

Top 95% -> 1.0521983827733101%, match count = 946

Top 100% -> 1.0460803685584168%, match count = 990