EN3160: Assignment 02

Fitting and Alignment

Index No. : 200462U

Name : Randika Perera

GitHub : Link to Source Code

Question 1

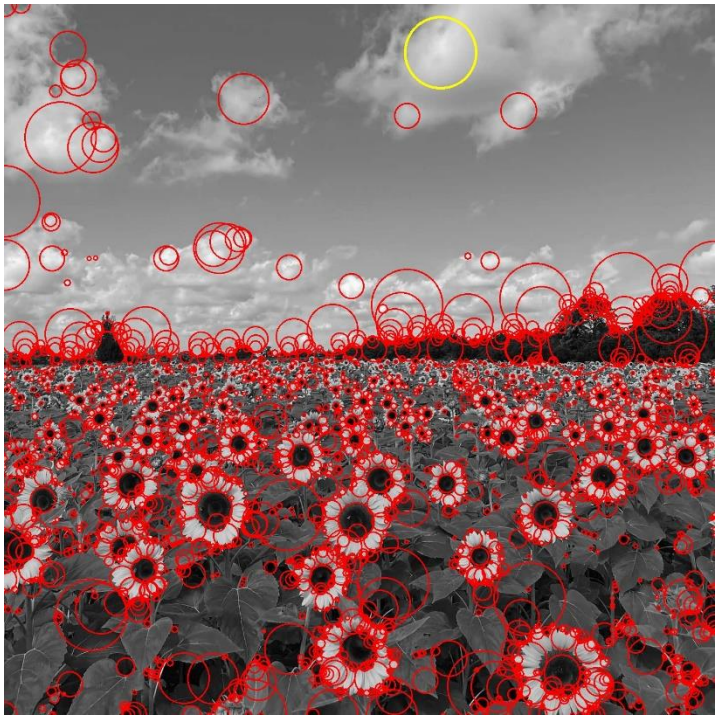
```python
def log_kernel(sigma, size):
    :
        tmp_kernel = np.multiply(kernel, np.square(x_idx) + np.square(y_idx) - 2 * sigma2) / (sigma2 ** 2) #Computes the Laplacian component of the LoG filter
        kernel = tmp_kernel - np.sum(tmp_kernel) / (size ** 2) # Final LoG filter
        return kernel
 :
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY) # Convert image to grayscale
 :
cv.normalize(gray, gray, 1, 0, cv.NORM_MINMAX) # Normalize the image
sigma0 = 0.4      # Initial sigma. Selected through trial and error to produce the best results
k = np.sqrt(2)    # Scale factor
num_scales = 15   # Number of scales
sigmas = sigma0 * np.power(k, np.arange(num_scales)) # Array of sigma values for each scale
 :
# LoG Filtering at Multiple Scales
img_stack = None
for i in range(num_scales):
        size = int(2 * np.ceil(4 * sigmas[i]) + 1) # Size of the kernel
        kernel = log_kernel(sigmas[i], size) * np.power(sigmas[i], 2) # LoG filter
        filtered = cv.filter2D(gray, cv.CV_32F, kernel) # Filter the image with the LoG filter
        filtered = pow(filtered, 2) # Square the filtered image
        # Filtered images are stored in img_stack
        if i == 0:
                img_stack = filtered
        else:
                img_stack = np.dstack((img_stack, filtered))
 :
# Maximum Response Extraction
scale_space = None
for i in range(num_scales):
        filtered = cv.dilate(img_stack[:, :, i], np.ones((3, 3)), cv.CV_32F, (-1, -1), 1, cv.BORDER_CONSTANT) # Dilate the image
        # Filtered images are stored in scale_space
        if i == 0:
                scale_space = filtered
        else:
                scale_space = np.dstack((scale_space, filtered))
max_stack = np.amax(scale_space, axis=2)  # Find the maximum response across all the dilated images along the third axis (axis=2)
max_stack = np.repeat(max_stack[:, :, np.newaxis], num_scales, axis=2)  # Expand the 2D image into a 3D stack of identical 2D images
max_stack = np.multiply((max_stack == scale_space), scale_space)  # Multiplies the 3D stack of images by a binary mask. The mask (max_stack == scale_space)
```

In this question, the Laplacian of the Gaussian (LoG) along with scale-space maxima detection was used to detect blobs.

The image was first converted into grayscale and then passed through the LoG filter.

The blobs were detected and circled on both the grayscale version of the image as well as the coloured version of the image. The largest blob was circled in red while other blobs were circled in red.

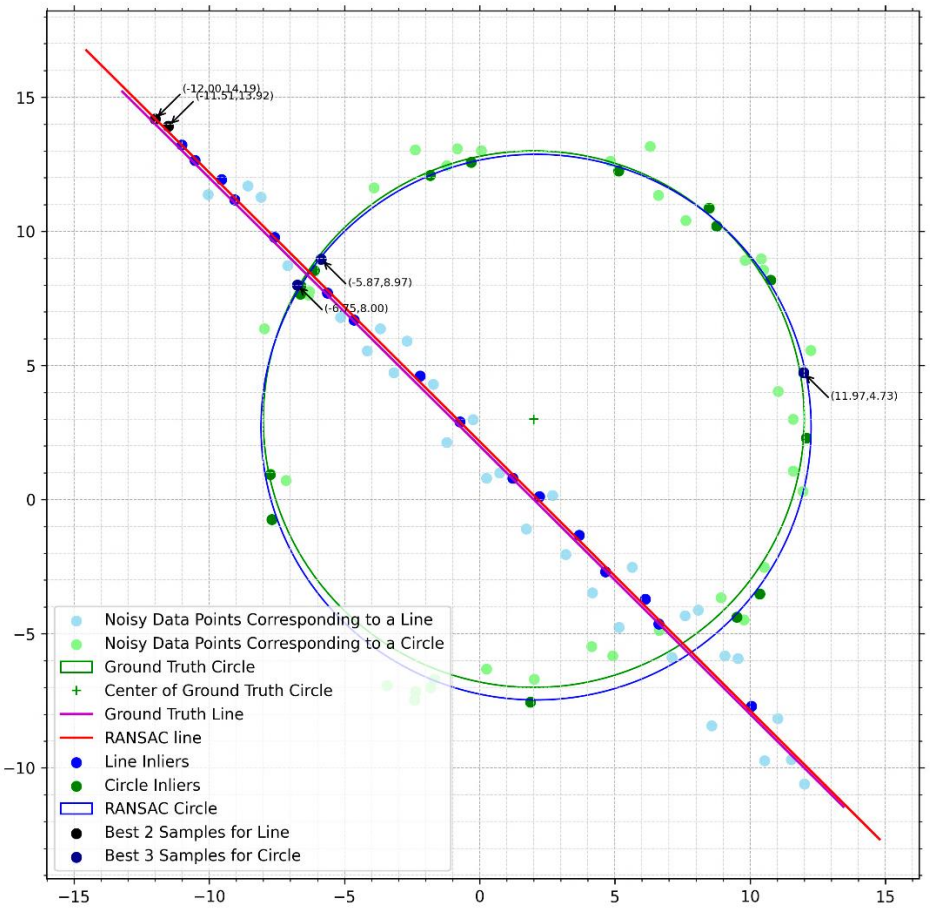
```
Largest Circle Parameters:
Radius: 72.40773439350254
Center Coordinates (x, y): (98, 884)

Range of σ values used:
Minimum σ: 0.4
Maximum σ: 51.200000000000045
```



Question 2

```python
def RANSAC_line_fitting(X, iterations, threshold, min_inliers):
    best_model = None
    best_inliers = []
    for i in range(iterations):
        sample_indices = np.random.choice(len(X),2,replace=False)
        x1,y1 = X[sample_indices[0]]
        x2,y2 = X[sample_indices[1]]
        a,b,d = find_line_parameters_using_2_points(x1,y1,x2,y2)
        magnitude = np.sqrt(a**2 + b**2)
        a = a/magnitude
        b = b/magnitude
        distances = np.abs(a*X[:,0]+b*X[:,1]-d)
        inliers  = np.where(distances<threshold)[0]
        if len(inliers) >= min_inliers:
            if len(inliers) > len(best_inliers):
                best_model   = (a,b,d)
                best_inliers = inliers
    return best_model,best_inliers


def RANSAC_circle_fitting(X,iterations,threshold,min_inliers):
    best_model   = None
    best_inliers = []
    for i in range(iterations):
        sample_indices = np.random.choice(len(X),3,replace=False)
        x1,y1 = X[sample_indices[0]]
        x2,y2 = X[sample_indices[1]]
        x3,y3 = X[sample_indices[2]]
        x_center,y_center,radius =
find_circle_parameters_using_3_points(x1,y1,x2,y2,x3,y3)
        errors  = np.abs(np.sqrt((X[:,0]-x_center)**2+(X[:,1]-
y_center)**2)-radius)
        inliers = np.where(errors<threshold)[0]
        if len(inliers) >= min_inliers:
            if len(inliers) > len(best_inliers):
                best_model   = (x_center,y_center,radius)
                best_inliers = inliers
    return best_model,best_inliers
```

```python
# RANSAC parameters for line estimation
iterations  = 100000
threshold   = 0.2
min_inliers = 15

# Estimate the line using RANSAC
best_RANSAC_line,line_inlier_indices_array =
RANSAC_line_fitting(X_line,iterations,threshold,min_inliers)



# RANSAC parameters for circle estimation
circle_iterations  = 100000
circle_threshold   = 0.2
circle_min_inliers = 15

remnant_indices = [i for i in range(len(X)) if i not in
line_inlier_indices_array]
remnant_points  = X[remnant_indices]

# Estimate the circle using RANSAC
best_RANSAC_circle, circle_inlier_indices_array =
RANSAC_circle_fitting(remnant_points,circle_iterations,circle_thr
eshold,circle_min_inliers)


# Plot the line estimated by RANSAC
x_min,x_max = ax.get_xlim()
x_ = np.array([x_min,x_max])
y_ = (-
best_RANSAC_line[0]*x_+best_RANSAC_line[2])/best_RANSAC_line[1]
plt.plot(x_,y_,label='RANSAC line',color='red')


# Plot the circle estimated by RANSAC
x_center,y_center,radius = best_RANSAC_circle
circle_estimated =
plt.Circle((x_center,y_center),radius,color='blue',fill=False,lab
el='RANSAC Circle')
ax.add_patch(circle_estimated)
```



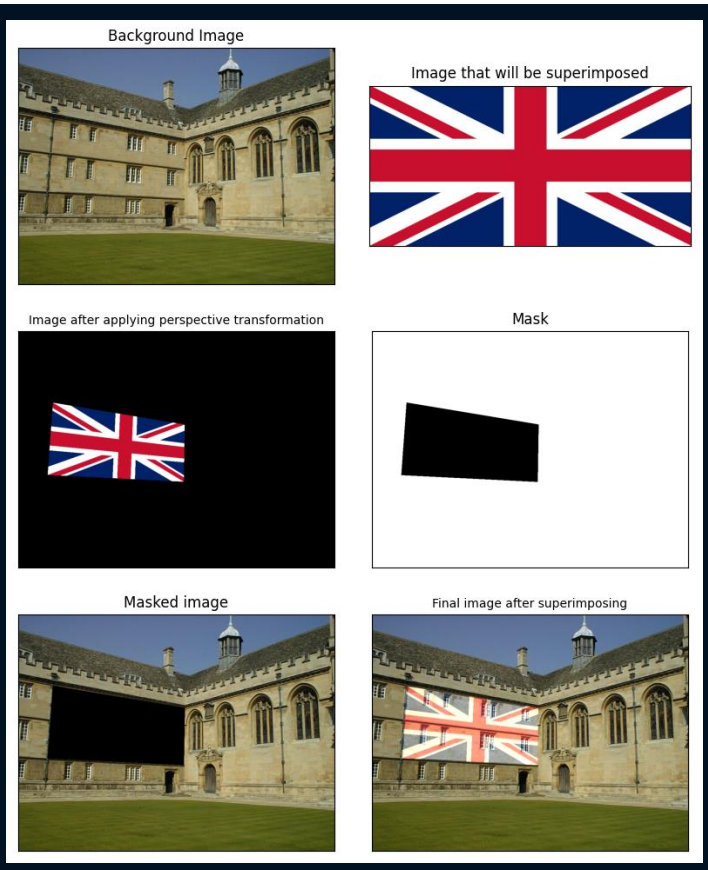
What will happen if we fit the circle first?

If we fit the circle first, it might not lead to accurate results due to the following reasons,

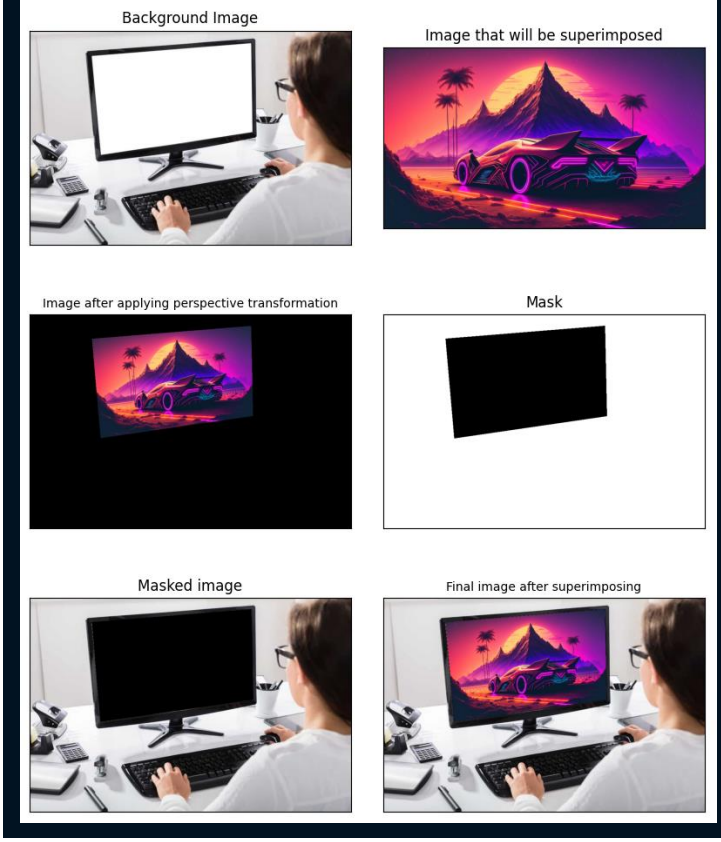
1. Since the circle fitting is done first, the estimated circle might include points that belong to the line. This can *distort the circle estimation.*
2. When we subsequently try to fit a line to the remaining points (after subtracting the consensus of the best circle), the performance of RANSAC may be compromised. RANSAC relies on the assumption that there is a dominant model in the data. If we remove a significant portion of the points that belong to the line while fitting the circle, *the performance of RANSAC for line fitting might degrade.*

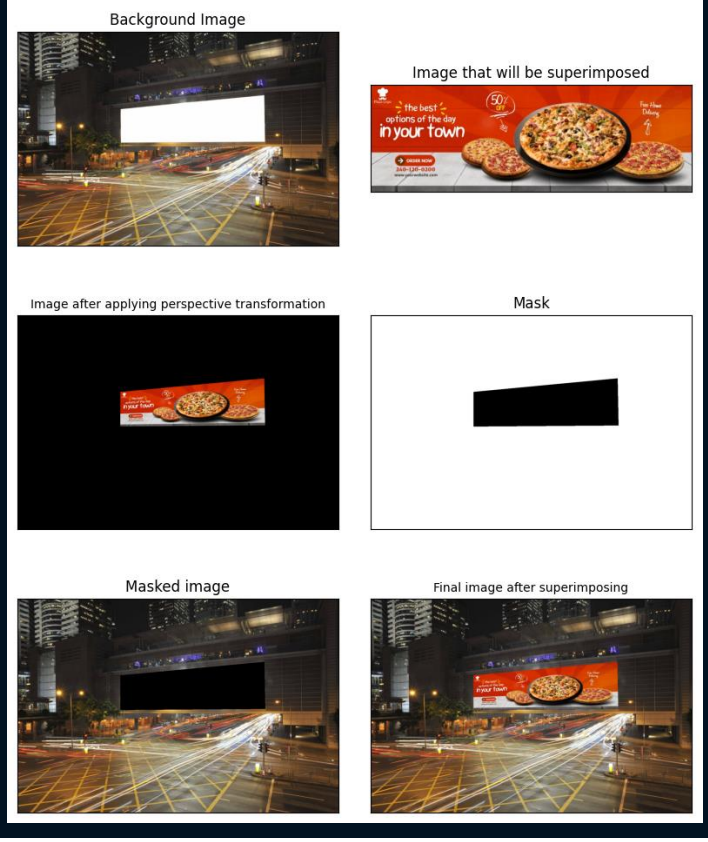
Question 3

```python
def selecting_points(event,x,y,flags,param):
# When the left mouse button is clicked (EVENT_LBUTTONUP), it appends the
coordinate (x,y) to the positions list and increments the count variable.
    global positions,count
    if event == cv.EVENT_LBUTTONUP:
          cv.circle(background_image,(x,y),5,(255,100,0),-1)
          positions.append([x,y])
          count = count + 1

positions = []  # List to store the coordinates of the points where the
second image has to be placed
count = 0       # Variable to keep track of the number of points selected

cv.namedWindow('Select Points') # Window to display the background image

cv.setMouseCallback('Select Points',selecting_points)
# Sets the mouse callback function to selecting_points.
# When the user interacts with this window using the mouse, the
selecting_points function is called.

while(True):
    cv.imshow('Select Points',background_image)
    k = cv.waitKey(20)
    if k == 27:  # Pressing the escape key will break the
                   loop (ASCII value of escape key is 27)
          break
cv.destroyAllWindows()

pts1 =
np.float32([[0,0],[width_superimposing_image,0],[0,height_superimposing_im
age],[width_superimposing_image,height_superimposing_image]])

pts2 = np.float32(positions[:4])

# Find the perspective transformation matrix (h) that aligns
image_to_superimpose with the user-selected points on the
background_image.
h,mask = cv.findHomography(pts1,pts2,cv.RANSAC,5.0)

# Apply the calculated perspective transformation to image_to_superimpose,
creating a new image that matches the perspective of background_image
perspective_transformed_superimposing_image =
cv.warpPerspective(image_to_superimpose,h,(width_background,height_backgro
und))
```




```python
# Use this line to superimpose the image with some transparency      (I used this line of code for the above image to have some transparency)
final = cv.addWeighted(background_image,1,perspective_transformed_superimposing_image,0.8,0)

# Use this line to superimpose the image without any transparency    (I used this line of code for the below 2 images since transparency is not required)
final = cv.bitwise_or(perspective_transformed_superimposing_image,masked_image)
```





Question 4

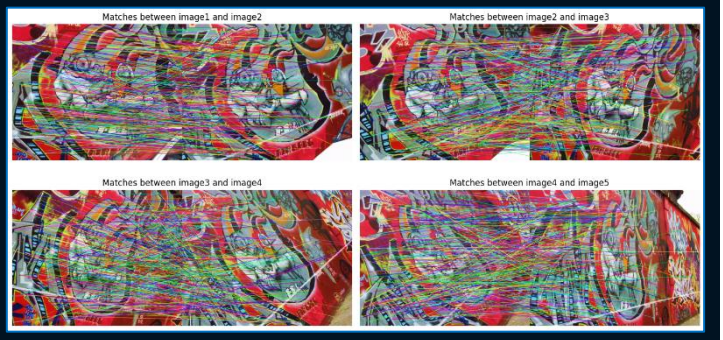
```python
def Find_Features(image):
    sift = cv.SIFT_create()
    keypoint, descriptor = sift.detectAndCompute(image, None)
    return keypoint, descriptor

def Find_Matches_By_Brute_Force(des1, des5):
    brute_force_matcher = cv.BFMatcher()
    matches = brute_force_matcher.knnMatch(des1, des5, k=2)
    return matches

def Process_Image(image_path):
    :
    return image, keypoint, descriptor

def Calculate_Homography_Matrix(sample):
    M = []
    for i in range(len(sample)):
        x1, y1, x2, y2 = sample[i, 0], sample[i, 1], sample[i, 2],
sample[i, 3]
        M.append([-x1, -y1, -1, 0, 0, 0, x2*x1, x2*y1, x2])
        M.append([0, 0, 0, -x1, -y1, -1, y2*x1, y2*y1, y2])
    M = np.matrix(M)
    U, S, V = np.linalg.svd(M)
    H = np.reshape(V[-1], (3, 3))
    H = (1/H.item(8))*H
    return H

def Calculate_Geometric_Distance(H, correspondence):
     :
     return np.linalg.norm(error)

def Final_Homography_Matrix_Using_RANSAC(corres,threshold):
    max_inliers = []
    homography = []
    for i in range(100):
        corr1 = corres[np.random.randint(0, len(corres))]
        corr2 = corres[np.random.randint(0, len(corres))]
        sample = np.vstack((corr1, corr2))
        corr3 = corres[np.random.randint(0, len(corres))]
        sample = np.vstack((sample, corr3))
        corr4 = corres[np.random.randint(0, len(corres))]
        sample = np.vstack((sample, corr4))
        h = Calculate_Homography_Matrix(sample)
        inliers = np.zeros((1,4))
        for j in range(len(corres)):
            distance = Calculate_Geometric_Distance(h, corres[j])
            if distance < 5:
                inliers = np.vstack((inliers,corres[j]))
        inliers = np.delete(inliers,0,0)
        if len(inliers) > threshold:
            max_inliers = inliers
            homography = Calculate_Homography_Matrix(max_inliers)
    return homography,max_inliers
```

```
Calculated Homography
 [[ 7.50767642e-01  1.44871582e-01  1.89851849e+02]
 [ 3.06867140e-01  1.32819838e+00 -7.02388117e+01]
 [ 6.90913005e-04  7.49002820e-05  1.00000000e+00]]
Number of inliers 255

Original Homography
   6.2544644e-01   5.7759174e-02   2.2201217e+02
  2.2240536e-01   1.1652147e+00  -2.5605611e+01
  4.9212545e-04  -3.6542424e-05   1.0000000e+00
```

We can see that the calculated homography matrix is close to the actual homography matrix (given in the dataset).

We can see that the matches between img1 and img5 are not very good. If we calculate the homography matrix using these matches, we will get a bad result and the stitched image may get distorted.

So instead of using only img1 and img5, I will use all the images and calculate the homography matrix using the matches between img1 and img2, img2 and img3, img3 and img4, and img4 and img5. Then I will combine all the homography matrices to get the final homography matrix.




```python
image_array = []
for i in range(1,6):
    image = cv.imread('../Data/graf/img'+str(i)+'.ppm')
    image_array.append(image)

homographs = []
threshold = 100
for i in range(4):
    correspondence = []
    key1, des1 = Find_Features(image_array[i])
    key5, des5 = Find_Features(image_array[i+1])
    keypoints = [key1, key5]
    matches = Find_Matches_By_Brute_Force(des1, des5)
    for match in matches:
        (x1, y1) = keypoints[0][match[0].queryIdx].pt
        (x2, y2) = keypoints[1][match[0].trainIdx].pt
        correspondence.append([x1, y1, x2, y2])
    corres = np.matrix(correspondence)
    H,inliers =Final_Homography_Matrix_Using_RANSAC(corres,threshold)
    homographs.append(H)

final_homograph = homographs[3]@homographs[2]@homographs[1]@homographs[0]
final_homograph = (1/final_homograph.item(8))*final_homograph
```