EN2550: Assignment 2

Name: B.G.D.T.Chathumini

Index Number: 190107T

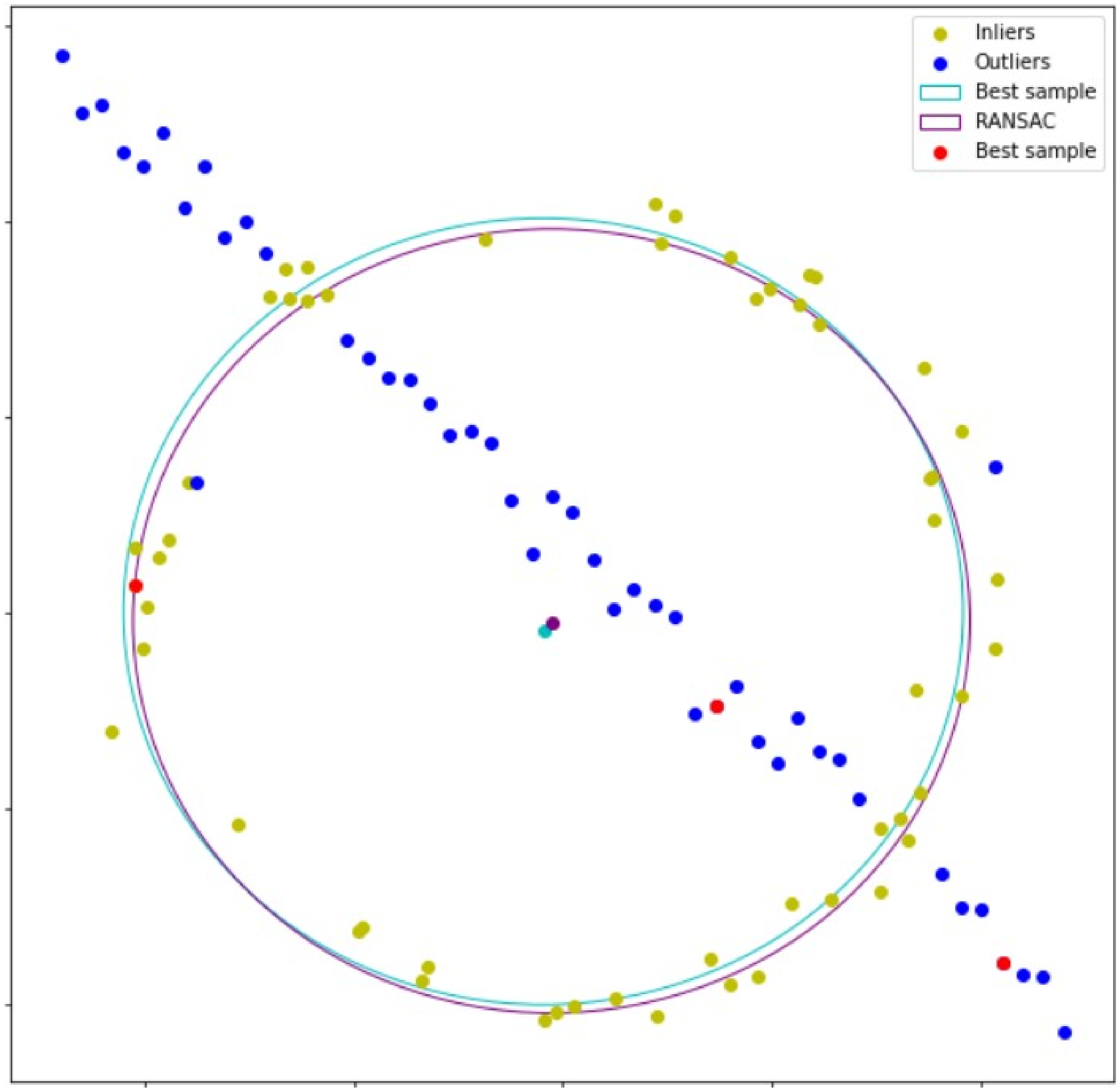
GitHub Link: `https://github.com/dulmi-19/Image-Processing-and-Machine-Vision`

Question 1 (a)

- S = 3 (The minimum number of points needed to estimate a circle)
- N = 35 (from no.of samples vs outlier ratio table, when probability of selecting at least one outlier free random sample is 0.99)

RANSAC circle estimation is done using the best model sample.

```python
class RANSAC: # RANSAC algorithm
    def __init__(self,x_values,y_values,thresh_dis,n_samples):
        # initialization of the variables used
        self.x_values=x_values
        self.y_values=y_values
        self.s = 3 #minimum no. of points needed
        self.t = thresh_dis # threshold distance
        self.N=n_samples
        self.outliers = []
        self.inliers = []
        self.points = []
        self.best_model = None
    def dis_point_point(self, p1,p2): # calculate distance from 2 points
        return ((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)**0.5
    def random_points(self): # select 3 random points
        self.points=[]
        i =1
        while i < 4:
            r_num =random.randint(0,len(self.x_values)-1)
            point = (self.x_values[r_num],self.y_values[r_num])
            if point not in self.points:
                self.points.append(point)
                i+=1
    def circle(self): # calculate the center point and radius of the circle
        p_1,p_2,p_3 = self.points
        Y = np.array([[p_2[0] - p_1[0], p_2[1] - p_1[1]], [p_3[0] - p_2[0], p_3[1] - p_2[1]]])
        Z = np.array([[p_2[0]**2 - p_1[0]**2 + p_2[1]**2 - p_1[1]**2], [p_3[0]**2 - p_2[0]**2 + p_3[1]**2 - p_2[1]**2]])
        inverse_Y = linalg.inv(Y)
        c_x, c_y = np.dot(inverse_Y, Z) / 2
        cx, cy = c_x[0], c_y[0]
        r = np.sqrt((cx - p_1[0])**2 + (cy - p_1[1])**2)
        return cx, cy, r
    def in_out_liers(self): # calculate inliers and outlier
        inliers_new =[]
        outliers_new = []
        cen_x, cen_y, r = self.circle()
        for i in range (len(self.x_values)):
            distance = self.dis_point_point((self.x_values[i],self.y_values[i]),(cen_x,cen_y))
            if abs(distance - r)<= self.t:
                inliers_new.append((self.x_values[i],self.y_values[i]))
            else:
                outliers_new.append((self.x_values[i],self.y_values[i]))
        if len(self.inliers)<len(inliers_new): # finding best model
            self.inliers=inliers_new
            self.outliers=outliers_new
            self.best_model=(cen_x, cen_y, r)
    def find_best_model(self): # find best model by repeating N times
        for j in range(self.N):
            self.random_points()
            self.in_out_liers()
        return self.best_model
```



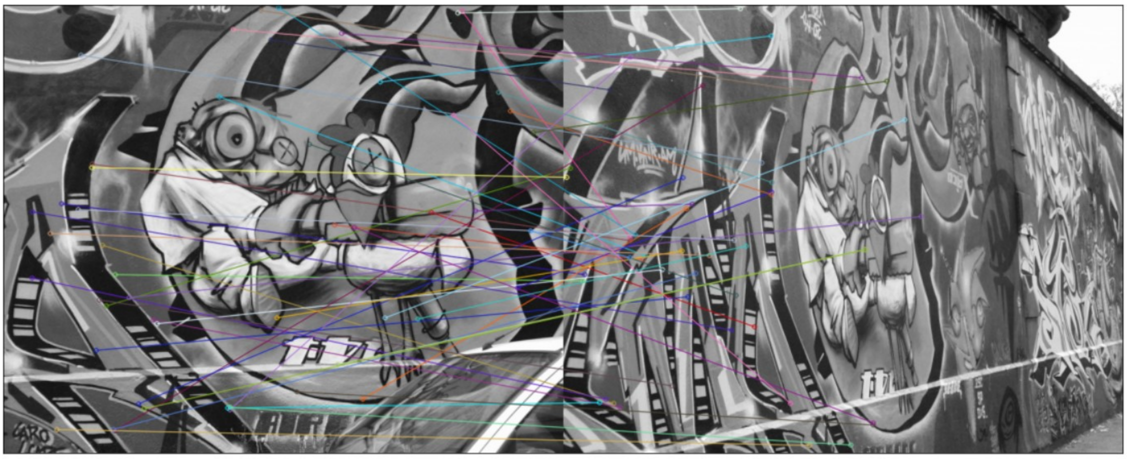
Question 2

```
mouse_points=[]
def get_mouse_points(image):
    # get co-ordinates of the mouse clicks
    global mouse_points
    img=cv.imread(image)
    count=0
    def click_event(event, x, y, flags, params):
        if event==cv.EVENT_LBUTTONDOWN:
            mouse_points.append([x,y])
            cv.circle(img, (x,y), 2, [0,0,255], 2) # drawing a small dot in the clicked position
            cv.imshow('image', img)
    cv.namedWindow('image',cv.WINDOW_AUTOSIZE)
    cv.imshow('image', img)
    cv.setMouseCallback('image', click_event)
    while count<4:
        cv.waitKey(1)
        count+=1
    cv.waitKey(0)
    cv.destroyAllWindows()
    return mouse_points
def homography(img_fg,img_bg):#homography and combine two images
    fh,fg=img_fg.shape[0],img_fg.shape[1]
    pts_src = np.array([[0, 0], [0, fg], [fh, 0],[fh, fg]])
    pts_dst = np.array(get_mouse_points('001.jpg'))
    h, status = cv.findHomography(pts_src, pts_dst)
    im_out = cv.warpPerspective(img_fg, h, (img_bg.shape[1],img_bg.shape[0]))
    return im_out
def plot_align(img_fg,img_bg): # only important code line is shown
    ax.imshow(cv.cvtColor(cv.addWeighted(img_bg,1,homography(img_fg,img_bg),0.5,0), cv.COLOR_BGR2RGB))
plot_align(flag,hall)
```

Question 3 (a)

SIFT Feature Mapping:

```
im_1 = cv.imread(r'./graf/img1.ppm')
im_2 = cv.imread(r'./graf/img5.ppm')
graf_1 = cv.cvtColor(im_1, cv.COLOR_BGR2GRAY)
graf_2 = cv.cvtColor(im_2, cv.COLOR_BGR2GRAY)
sift = cv.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(graf_1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(graf_2,None)
bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)
image= cv.drawMatches(graf_1, keypoints_1, graf_2, keypoints_2, matches[:50], graf_2, flags=2)
```



Question 3 (b)

Because the protective angle between images 1 and 5 is large, it is quite difficult to discover a decent number of properly matching features using SIFT. Therefore, to compute homography between images 1 and 5, homograpies between images 1,4 and 4,5 (H45, H14) are used.

- Actual Homography:

$$\begin{bmatrix} 6.2544644e-01 & 5.7759174e-02 & 2.2201217e+02 \\ 2.2240536e-01 & 1.1652147e+00 & -2.5605611e+01 \\ 4.9212545e-04 & -3.6542424e-05 & 1.0000000e+00 \end{bmatrix}$$

- Calculated Homography:

$$\begin{bmatrix} 6.22685625e-01 & 5.54507133e-02 & 2.22329787e+02 \\ 2.15499411e-01 & 1.15588616e+00 & -2.19799673e+01 \\ 4.85364432e-04 & -4.23870163e-05 & 1.0000000e+00 \end{bmatrix}$$

- SSD Value = 3.639549470953883

The general code for homography:

```python
def homography(X, Y):
    O = np.array([[0],[0],[0]])
    A = []
    for i in range(4):
        A.append(np.concatenate((O.T, np.expand_dims(X.T[i,:], axis=0), np.expand_dims(-1*Y[1, i]*X.T[i,:], axis=0) ), axis=1))
        A.append(np.concatenate((np.expand_dims(X.T[i,:], axis=0), O.T, np.expand_dims(-1*Y[0, i]*X.T[i,:], axis=0) ), axis=1))
    A = np.array(A).squeeze().astype(np.float64)
    eigen_values, eigen_vectors = np.linalg.eig(A.T @ A)
    Homo = eigen_vectors[:, np.argmin(eigen_values)]
    Homo  = Homo.reshape(3, -1)
    return Homo
```

Calculating the homography of image 1 to 5:

```python
H14, count14, count_db14, best_fit_X_inliers14, best_fit_Y_inliers14 = RANSAC(r'./graf/img1.ppm', r'./graf/img4.ppm', 1, 20, 10000)

H45, count45, count_db45, best_fit_X_inliers45, best_fit_Y_inliers45 = RANSAC(r'./graf/img4.ppm', r'./graf/img5.ppm', 1, 20, 10000)

computed_homography = H45 @ H14
computed_homography=computed_homography/computed_homography[2,2]  # making last element of the homography to 1
```

Question 3 (c)

Image Stitching:

```python
final_img = cv.warpPerspective(im_1, computed_homography, dsize=(im_1.shape[1], im_1.shape[0]+100))
transformed_im_1 = np.copy(final_img)

for i in range(final_img.shape[0]):
    for j in range(final_img.shape[1]):
        if np.all(final_img[i,j] == 0) and i < im_2.shape[0] and j < im_2.shape[1]:
            final_img[i,j] = im_2[i,j]
```