EN2550 - Fundamentals of Image Processing and Machine Vision

Assignment – 2

M. D. A. J. Abeyratne – 190009U

Question 1

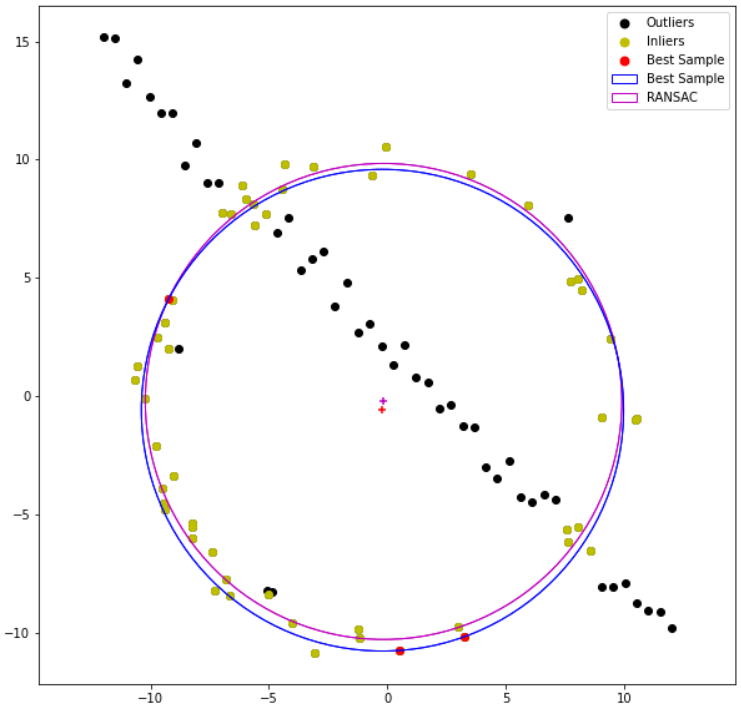
RANSAC, which is the short term for **RAN**dom **SA**mple **C**onsensus is an algorithm which is used for model fitting in the presence of outliers. Briefly what happens here is, we first take the required number of samples to make the required shape (2 for a line, 3 for a circle), draw the shape, and check how many data points are there as inliers (close to the shape) and outliers (far from the shape). As we define a lower threshold previously (which is being calculated from a trial-and-error method to improve the accuracy) all the inliers will be then taken into a different list. After that, a shape will e estimated for that given set of inliers. So, the initial points may or may not be on the final shape drawn.

As the required estimation was a circle, 3 points were needed for drawing the shape. Initially, the circle and the straight line consisted of random data were created using the given code snippet. Next, a code was taken from the internet, where it would give the radius and the center of a circle, when three points are given as the input.

```
x_cor1, y_cor1 = x_con[a1],y_con[a1]
x_cor2, y_cor2 = x_con[a2],y_con[a2]
x_cor3, y_cor3 = x_con[a3],y_con[a3]

x_points = (x_cor1, x_cor2, x_cor3)
y_points = (y_cor1, y_cor2, y_cor3)

c = (x_cor1-x_cor2)**2 + (y_cor1-y_cor2)**2
a = (x_cor2-x_cor3)**2 + (y_cor2-y_cor3)**2
b = (x_cor3-x_cor1)**2 + (y_cor3-y_cor1)**2

s = 2*(a*b + b*c + c*a) - (a*a + b*b + c*c)

cen_x = (a*(b+c-a)*x_cor1 + b*(c+a-b)*x_cor2 + c*(a+b-c)*x_cor3) / s
cen_y = (a*(b+c-a)*y_cor1 + b*(c+a-b)*y_cor2 + c*(a+b-c)*y_cor3) / s

ar = a**0.5
br = b**0.5
cr = c**0.5

r = ar*br*cr / ((ar+br+cr)*(-ar+br+cr)*(ar-br+cr)*(ar+br-cr))**0.5

center = (cen_x, cen_y)
```

Then 3 points were randomly chosen from the data set, and the number of inliers were calculated. The threshold value was set for 53 to improve the accuracy of the initial circle, which was drawn using the 3 points.

Next, the RANSAC circle was drawn using the inlier data set.



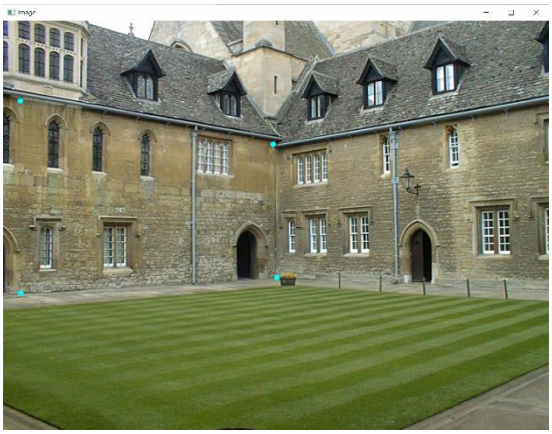

```
# Coordinates of the 2D points
x = in_list_x
y = in_list_y

# coordinates of the barycenter
x_m = np.mean(x)
y_m = np.mean(y)

# calculation of the reduced coordinates
u = x - x_m
v = y - y_m

Suv  = sum(u*v)
Suu  = sum(u**2)
Svv  = sum(v**2)
Suuv = sum(u**2 * v)
Suvv = sum(u*v**2)
Suuu = sum(u**3)
Svvv = sum(v**3)

# Solving the linear system
A = np.array([[Suu, Suv], [Suv, Svv]])
B = np.array([ Suuu + Suvv, Svvv + Suuv ])/2.0
uc, vc = linalg.solve(A, B)

xc_1 = x_m + uc
yc_1 = y_m + vc

# Calculation of all distances from the center (xc_1, yc_1)
Ri_1      = np.sqrt((x-xc_1)**2 + (y-yc_1)**2)
R_1       = np.mean(Ri_1)
residu_1  = sum((Ri_1-R_1)**2)
residu2_1 = sum((Ri_1**2-R_1**2)**2)

cen = (xc_1,yc_1)
```

Question 2

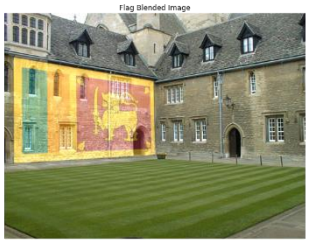
In the second question the task is to map a flag to an architectural structure, with four mouse clicks on the image. Initially a function was defined to get the coordinates of the click points, where it will give an array of the (x, y) coordinates of the four click points. After clicking, the flag is being homography transformed and warped in the correct manner using **cv.findHomography** and **cv.warpPerspective**. Finally, the image is being blended with the architectural image with a weight of 0.5. This amount of weight was given to show the building, as well as the flag clearly, and giving it a natural look.

```
def MouseHandling(event,x,y,f,pram):
    global temp2,click_points
    if event == cv.EVENT_LBUTTONDOWN:
        cv.circle(temp2 ,(x,y), 2, (255,255,0), 5, cv.LINE_AA)
        cv.imshow("Image",temp2)
        if len(click_points)<4:
            click_points = np.append(click_points,[(x,y)],axis=0)
    return
```

```
Homograpy_Matrix, status = cv.findHomography(click_points, point_coord)

trans_flag = cv.warpPerspective(flag, np.linalg.inv(Homograpy_Matrix), (h_w, h_h))

blend_image = cv.addWeighted(hall, 1, trans_flag, 0.5, 0)
```



Click Points (Cyan)



Outputs

Question 3

In this question, the outcome is to map SIFT features, transform the second image appropriately, and stitch it with the first image.

In the first part of the question, the function **key_point_des_point**(img1,img2) is defined, where it takes the two images as the input, and gives outputs of the key points. Then, the match points are connected, and lines are drawn to show the similar points.

```
def key_point_des_point(img1,img2):
    sift = cv.SIFT_create()
    kp1, desc1 = sift.detectAndCompute(img1,None)
    kp2, desc2 = sift.detectAndCompute(img2,None)
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks=1000)
    flann = cv.FlannBasedMatcher(index_params,search_params)
    matches = flann.knnMatch(desc1,desc2,k=2)

    good = []
    pts1 = []
    pts2 = []

    for m,n in matches:
        if m.distance < 0.7*n.distance:
            good.append([m])
            pts2.append(kp2[m.trainIdx].pt)
            pts1.append(kp1[m.queryIdx].pt)

    return pts1,pts2,good,kp1,kp2
```

In the second part of the question, the homography is needed to be calculated in our own code within RANSAC algorithm and it is needed to be compared with the homography in the given dataset for img1 and img 5. Rather than directly calculating the homography between img1 and img5, a sequential approach was taken to calculate first between img1 and img2, img2 and img3 and so and finally the homography matrices are then being multiplied each other to get the homography between img1 and img5, which is more accurate.

```
Cal_H = np.identity(3)
for r in range(1,5):
    img1 = cv.imread(r'img{}.ppm'.format(r),cv.IMREAD_COLOR)
    img2 = cv.imread(r'img{}.ppm'.format(r+1),cv.IMREAD_COLOR)

    pts1, pts2, good, kp1, kp2 = key_point_des_point(img1,img2)
    good = np.array(good)
    H, count = RANSAC(pts1,pts2,good, 1, 4, 10000)
    Cal_H=np.dot(H,Cal_H)
```

For the calculations, several functions were defined, such as **get_homo**(X, Y), **inlier_count**(X_f, Y_f, H, t), **ran_pts**(X, n) and **RANSAC**(pts1, pts2, matches, t, s, N).

The **RANSAC**(pts1, pts2, matches, t, s, N) function takes the matched points, along with the other RANSAC parameters, where

```
def get_homo(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)
    eig_val, eig_vect = np.linalg.eig(A.T @ A)
    H = eig_vect[:, np.argmin(eig_val)]
    H = H.reshape(3, -1)

    return H

def inlier_count(X_f, Y_f, H, t):
    count = 0
    t_X_f = H @ X_f
    t_X_f = t_X_f / t_X_f[2,:]
    error = np.sqrt(np.sum(np.square(t_X_f - Y_f), axis=0))
    count = np.where(error <= t)[0].shape[0]
    return count
```

the loop count is given as 10000 in the code, and it will give the output of the best fitting homography and the highest inlier count.

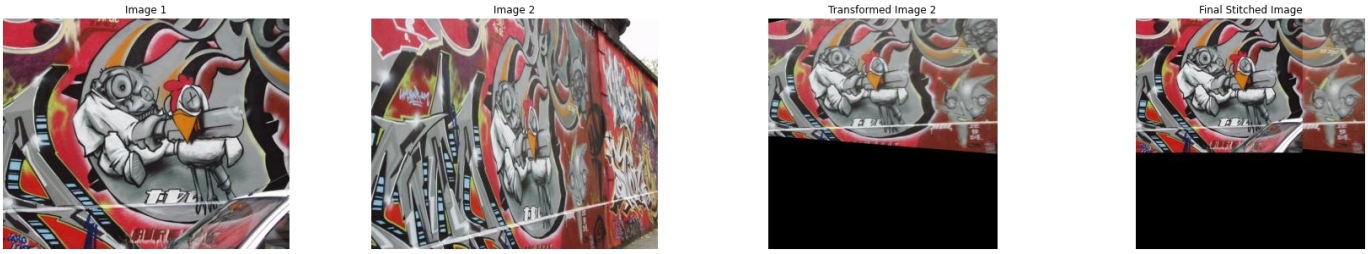
The `get_homo(X, Y)` function the homography is calculated. As the lists are not in the required shape, rows and columns of zeros are concatenated for the ease of calculations. Finally, the homography vector is given as the output.

The `inlier_count(X_f, Y_f, H, t)` functions takes the SIFT features of img1 and img5, homography matrix calculated from the get_homo function, and give the count of the inliers which are less than the error. The inlier count will be calculated on each loop. It will always be updated from the highest inlier count, and the corresponding homography matrix in the RANSAC function.

```
def ran_pts(X,n):
    sam=[]
    for i in range(0,n):
        ran_ind=np.random.randint(len(X))
        while True:
            if ran_ind not in sam:
                sam.append(ran_ind)
                break
            else:
                ran_ind=np.random.randint(len(X))
    return sam

def RANSAC(pts1, pts2, matches, t, s, N):
    best_fit_hom = None
    best_inl_count = 0
    X_f =np.concatenate((pts1,np.ones((len(pts1),1))),axis=1).T
    Y_f =np.concatenate((pts2,np.ones((len(pts2),1))),axis=1).T
    for r in range(N):
        x = ran_pts(matches,s)
        X = np.zeros((4,3))
        Y = np.zeros((4,3))
        for i, j in enumerate(x):
            X[i,:] = np.array([pts1[j][0], pts1[j][1], 1])
            Y[i,:] = np.array([pts2[j][0], pts2[j][1], 1])
        X = X.T
        Y = Y.T
        H = get_homo(X,Y)
        count = inlier_count(X_f, Y_f, H, t)
        if count > best_inl_count:
            best_fit_hom = H
            best_inl_count = count
    return best_fit_hom, best_inl_count
```

Finally, the predefined homography transform matrix is read using the “H1to5p” file to calculate the error of the transformation found by the written code. It is varying from time to time as the points selection is randomly done. Usually, it’s in the range from 1-5. The for the given transformation in the images, the error is `3.453721560431695`

```
f = open("H1to5p","r").read().split()
actual_H = np.zeros((3,3))
for i in range(3):
    for j in range(3):
        actual_H[i][j] += float(f[3*i + j])

d = len(good) * 0.8
error = np.sum(np.square(actual_H - Cal_H/Cal_H[2][2]))
print(error)
#print(Cal_H/Cal_H[2][2])
final_img = cv.warpPerspective(img5,linalg.inv(Cal_H/Cal_H[2][2]),(1100,1100))
t_img = np.copy(final_img)
final_img[0:img0.shape[0], 0:img0.shape[1]] = img0
```



Outputs

Github Link - https://github.com/akilaabeyratne/EN2550-Assignment-2