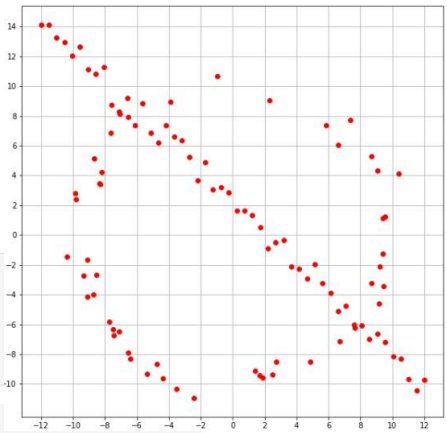
EN2550 Assignment 2 on Fitting and Alignment

Name – Wanshika W.A.R.

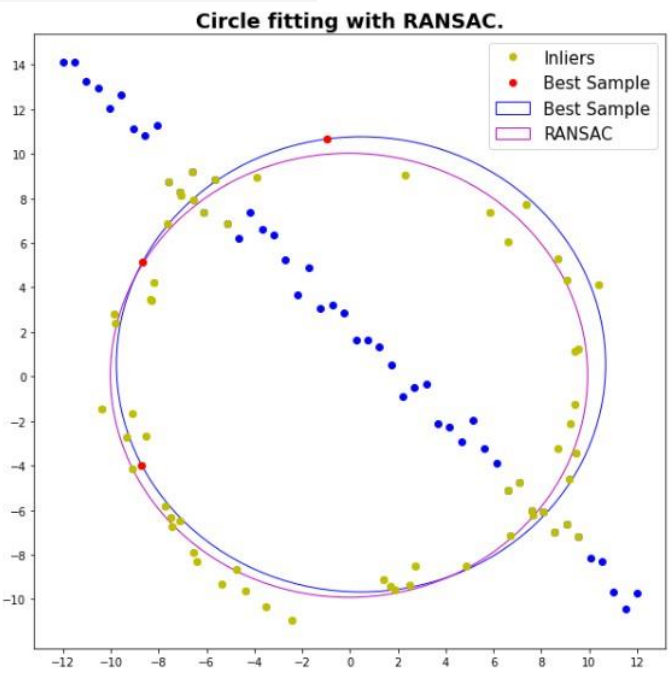
Index No. – 190663R

(01)



[1]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy import linalg
import circle_fit as cf
```

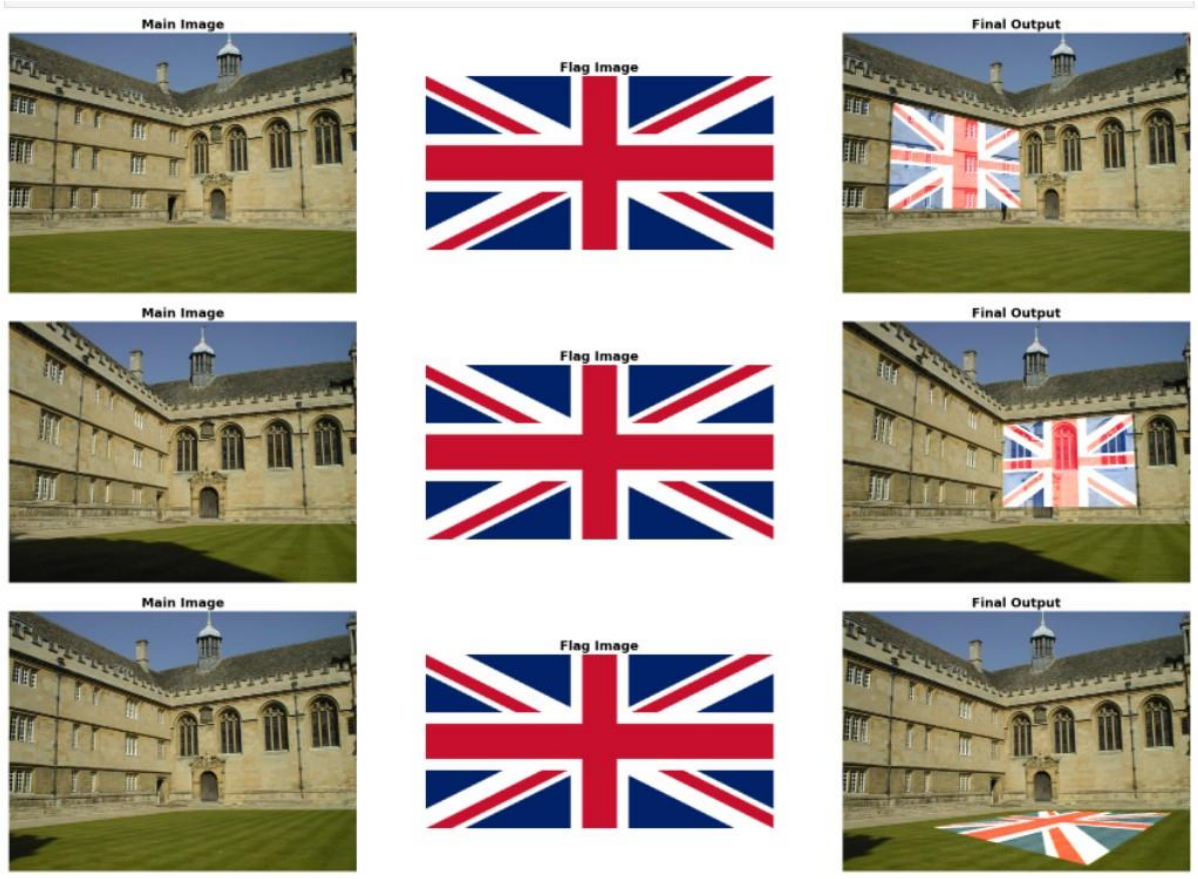
```
def RANSAC_circle(X):
    pts = 3
    t = 1.96
    d = 50
    N = 35

    best_circle = None
    best_x = None
    best_in_cnt = 0

    #Get random 3 points from data to estimate the circle

    for i in range(N):
        pt = []

        for j in range(pts):
            h_on = X[np.random.randint(0, 100), :]

            if len(pt) == 0:
                pt.append(h_on)
            elif np.array_equal(h_on, pt[-1]):
                while np.array_equal(h_on, pt[-1]):
                    h_on = X[np.random.randint(0, 100), :]

                pt.append(h_on)
            else:
                pt.append(h_on)

        a, b, r = est_circle(pt[0], pt[1], pt[2])

        if a == None:
            continue

        cnt, inliers = inlier_cnt(a, b, r, X, t)

        #Draw ransac circle

        if cnt > best_in_cnt:
            best_circle = plt.Circle((a, b), r, color='b', fill=False, label="Best Sample")
            best_pt = pt
            best_inliers = inliers
            best_in_cnt = cnt

    xc,yc,r,_ = cf.least_squares_circle(best_inliers)

    #Draw least square circle

    ransac_circle = plt.Circle((xc, yc), r, color='m', fill=False, label="RANSAC")

    return ransac_circle, best_circle, best_pt, best_inliers

ransac_circle, best_circle, best_pt, best_inliers = RANSAC_circle(X)
```



In this question, we selected three randomly selected points(s = 3) to draw the circle. The threshold value(t) is kept at 1.96 to achieve a 95% probability of getting all inliers. There are 100 data points in this graph. Out of them, 50 need to be inliers the other 50 need to be the outlier. Therefore, the consensus set size has been kept at 50 (d = 50) for accurate circle estimation. The number of samples (N) keep at 35 to have a probability of 99% of having at least one outlier random sample. (According to the equation of determining the number of iterations N)

(02)

```
cordinates = []
def click(event, x, y, flags, params):
    if event == cv.EVENT_LBUTTONDOWN:
        cordinates.append([x,y])


cv.imshow('image', im)
cv.setMouseCallback('image', click)
cv.waitKey(0)
cv.destroyAllWindows()

h, w = np.shape(im)[0], np.shape(im)[1]
zero_matrix = np.array([[0],[0],[0]])

for i in range (4):
    for j in range (2):
        globals()[["x","y"][j]+"_dash_"+str(i+1)] = cordinates[i][j]

flag_h , flag_w = np.shape(flag)[0] , np.shape(flag)[1]

x1T = np.array([[0,0,1]])
x2T = np.array([[flag_w-1, 0,1]])
x3T = np.array([[flag_w-1, flag_h-1, 1]])
x4T = np.array([[0,flag_h-1,1]])

for i in range(4):
    a = np.concatenate((zero_matrix.T,globals()["x"+str(i+1)+"T"], -globals()["y_dash_"+str(i+1)]*globals()["x"+str(i+1)+"T"]), axis=1)
    b = np.concatenate((globals()["x"+str(i+1)+"T"],zero_matrix.T, -globals()["x_dash_"+str(i+1)]*globals()["x"+str(i+1)+"T"]), axis=1)
    if i==0:
        A=np.concatenate((a,b), axis=0, dtype = np.float32)
    else:
        A=np.concatenate((A,a,b), axis=0, dtype = np.float32)

W,V = np.linalg.eig((A.T)@A)
temp = V[:, np.argmin(W)]
H = temp.reshape((3,3))

flag_trans = cv.warpPerspective(flag, H,(w,h))
output = cv.add(flag_trans,im)
```



In this question, we need to superimpose Merton college III with the British flag. At first, we need to set the coordinates of four points on the Merton college image. Then map the flag on the Merton college selected points by computing homography. Then wrap the flag image and blend it on the Merton college III architectural image using the cv.wrapPerspective built-in function in OpenCV.

(03)

```
im1 = cv.imread(r"E:\#####ACCA folders\acca 4 th sem\Fundamentals of Image Processing and Machine Vision\Assignment 02\Graf\img1.ppm", cv.IMREAD_GRAYS
im5 = cv.imread(r"E:\#####ACCA folders\acca 4 th sem\Fundamentals of Image Processing and Machine Vision\Assignment 02\Graf\img5.ppm", cv.IMREAD_GRAYS

sift = cv.SIFT_create()
kp1 , decs1 = sift.detectAndCompute(im1,None)
kp2 , decs2 = sift.detectAndCompute(im5,None)

FLANN_INDEX_KDTREE = 0
index_paras = dict(algorithm = FLANN_INDEX_KDTREE , trees =5)
search_paras = dict(checks = 100)

flann =  cv.FlannBasedMatcher(index_paras,search_paras)
matches = flann.knnMatch(decs1,decs2, k=2)

good = [[0,0] for i in range(len(matches))]

for i , (m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        good[i]=[1,0]

draw_params = dict(matchesMask = good, flags = 2)

im3 = cv.drawMatchesKnn(im1,kp1,im5,kp2,matches,None,**draw_params)

kp_im_1 = np.zeros(im1.shape)
kp_im_5 = np.zeros(im5.shape)

kp_im_1 = cv.drawKeypoints(im1, kp1, kp_im_1)
kp_im_5 = cv.drawKeypoints(im5, kp2, kp_im_5)
```

Image 1 Features

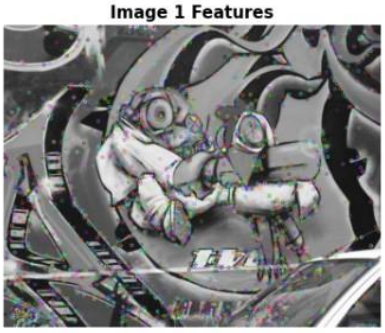
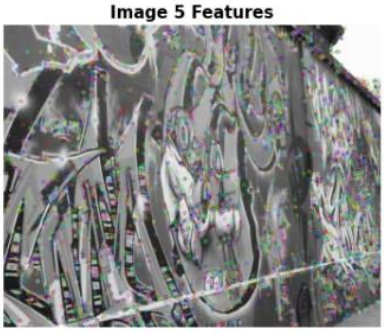


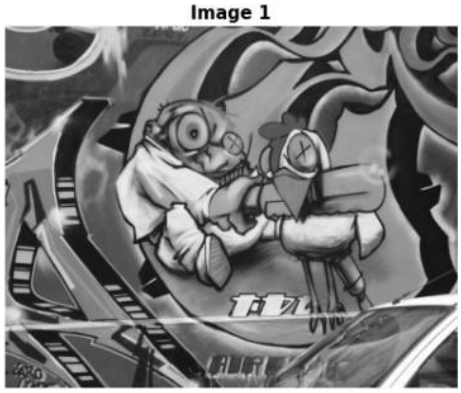
Image 5 Features



Matching Features in Image 01 & Image 05

```
def compute_Homography(from_Pt, to_Pt):

    X1, Y1, X2, Y2, X3, Y3, X4, Y4 = to_Pt[0], to_Pt[1], to_Pt[2], to_Pt[3], to_Pt[4], to_Pt[5], to_Pt[6], to_Pt[7]
    X1T, X2T, X3T, X4T = from_Pt[0], from_Pt[1], from_Pt[2], from_Pt[3]
    z_mat = np.array([[0],[0],[0]])

    m = np.concatenate((z_mat.T,X1T, -Y1*X1T), axis=1)
    n = np.concatenate((X1T,z_mat.T, -X1*X1T), axis=1)

    o = np.concatenate((z_mat.T,X2T, -Y2*X2T), axis=1)
    p = np.concatenate((X2T,z_mat.T, -X2*X2T), axis=1)

    q = np.concatenate((z_mat.T,X3T, -Y3*X3T), axis=1)
    r = np.concatenate((X3T,z_mat.T, -X3*X3T), axis=1)

    s = np.concatenate((z_mat.T,X4T, -Y4*X4T), axis=1)
    t = np.concatenate((X4T,z_mat.T, -X4*X4T), axis=1)

    A = np.concatenate((m,n,o,p,q,r,s,t), axis=0, dtype = np.float32)
    W,V = np.linalg.eig((A.T)@A)
    temp = V[:, np.argmin(W)]
    K = temp.reshape((3,3))
    return K
```

Image 1



```
t_h, I, b_H = 2, 0, 0

for k in range(N):
    rand_pts = randN(len(lst_kp1)-1,4)

    from_Pt = []
    for i in range(4):
        from_Pt.append(np.array([[lst_kp1[rand_pts[i]][0], lst_kp1[rand_pts[i]][1], 1]]))

    to_Pt = []
    for j in range(4):
        to_Pt.append(lst_kp2[rand_pts[j]][0])
        to_Pt.append(lst_kp2[rand_pts[j]][1])

    inliers = 0
    for i in range(len(lst_kp1)):
        X = [lst_kp1[i][0], lst_kp1[i][1], 1]
        hX = compute_Homography(from_Pt, to_Pt)@X
        hX /= hX[-1]
        error = np.sqrt(np.power(hX[0]-lst_kp2[i][0],2) + np.power(hX[1]-lst_kp2[i][1],2))
        if error < t_h: inliers+=1

    if inliers > I:
        I = inliers
        b_H = compute_Homography(from_Pt, to_Pt)

H.append(b_H)
```

Image 5

```
H_1_to_5 = H[3] @ H[2] @ H[1] @ H[0]
H_1_to_5 /= H_1_to_5[-1][-1]

trans = cv.warpPerspective(im1, H_1_to_5 ,(np.shape(im5)[1] ,np.shape(im5)[0]))
```

Transformed Image 1

Stitched Image 1 on Image 5

In This question part (a), We draw the features in image 01 and image 05. Then features are mapped using SIFT function and Flann-based matcher. In part (b), We compute the homography using the RANSAC algorithm to avoid perspective differences between 2 images and get better sift feature matches. Then the transformation is carried by the computed homography which is almost the same as the actual homography. Then in part (c), we stitched the transformed image 01 onto Image 05 using the cv.warpPerspective built-in function in OpenCV.

GitHub - https://github.com/wanshika99/Assignment-02.git