EN2550 Assignment 2 on Fitting and Alignment

Name: Hettihewa D.P.G.

Index No: 190231R

Github link:

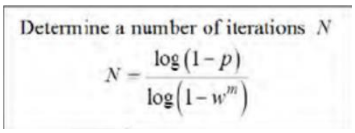
01)Development of the RANSAC algorithm

- Parameter Selection

S = 3 :- we need at least three point to randomly select circles

T = 1.96 :- Threshold of 1.96 gives us a probability 0.95 to catch all inliers

D = 50 :- Out of the 100 data points only 50 are corresponding to the circle. For accurate estimations circles should contain 50 inliers

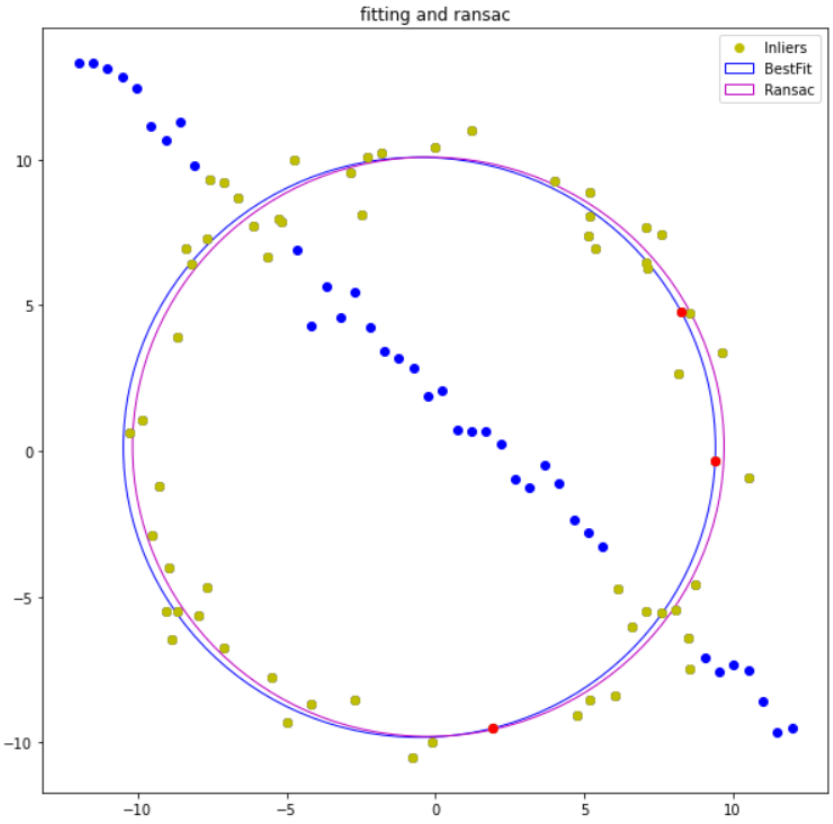


N = 35 :- By substituting for this equation we can find the No of iteration required to have a probability of 0.99 of having at least one outlier free sample

a) RANSAC Algorithm

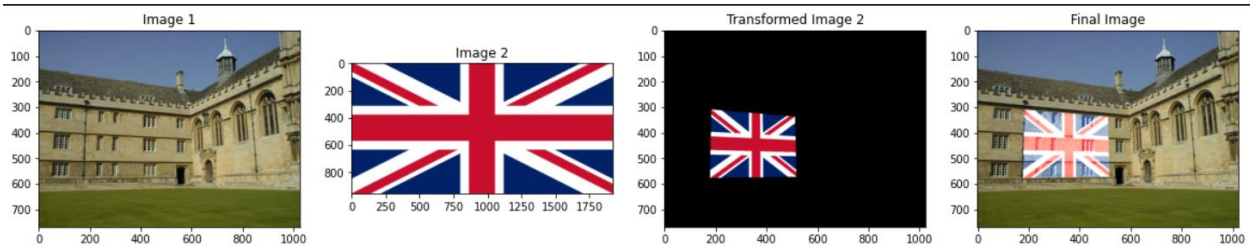
```
def RANSAC(X):
    s,  d, N = 3, 50, 35
    c1,c2 = 0,0
    nMax = 0
    bestInliers = []
    p_ = []
    for i in range(N):
        points = []
        while len(points) <3:
            p = X[np.random.randint(0, 100), :]
            while np.array_equal(p, x[-1]):
                    p = X[np.random.randint(0, 100), :]

            points.append(p)
        cx, cy, r =  get_circle(points[0][0], points[0][1], points[1][0], points[1][1], points[2][0], points[2][1])
        n, inliers = calc_inlier(cx, cy, r)
        if n > nMax:
            nmax = N
            bestInliers = inliers
            p_ = points
            bestFitCircle = plt.Circle((cx, cy), r, color = 'b', fill = False, label = "BestFit")
    if nMax < N:
        print("cannot find a suitable cicle")
    Cx, Cy, R, u = cf.least_squares_circle(bestInliers)

    ransacCircle = plt.Circle((Cx, Cy), R, color = 'm', fill = False, label = "Ransac")
    return bestInliers, ransacCircle, bestFitCircle, p_
```

b)

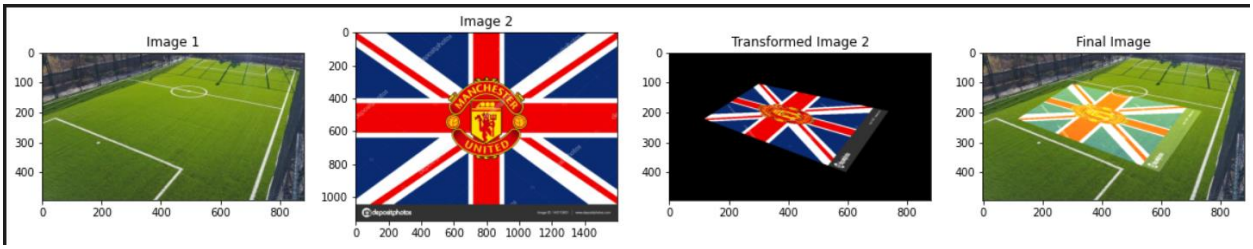


02)

1)Oxford university building and England flag

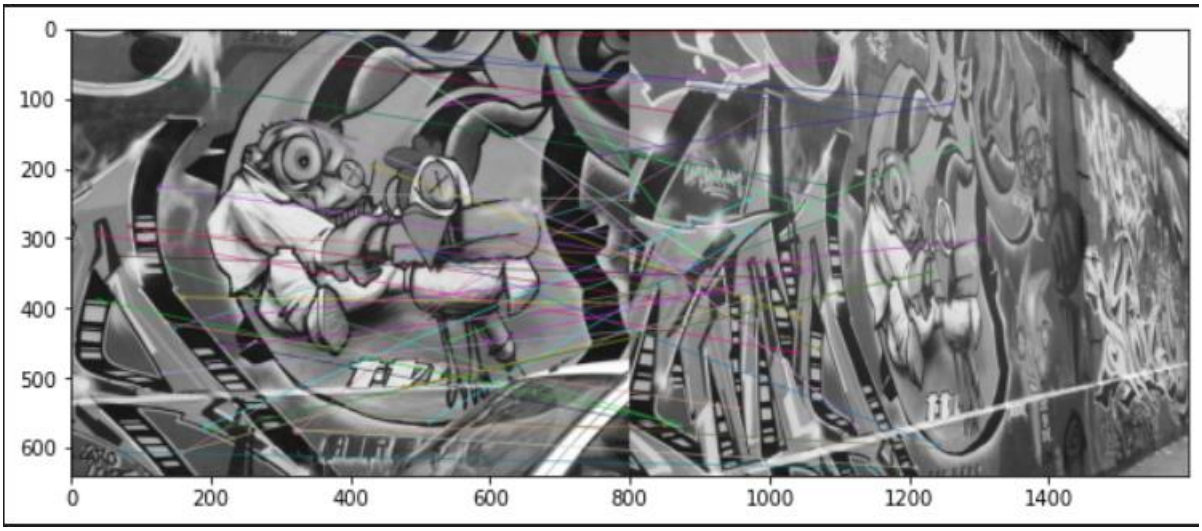


2) Football Field and Men United Flag



03)

a) SIFT Feature matching




```
im1Original = cv.imread(r'./images/img1.ppm')
im5Original = cv.imread(r'./images/img5.ppm')

im1 = cv.cvtColor(im1Original, cv.COLOR_BGR2GRAY)
im2 = cv.cvtColor(cv.imread(r'./images/img2.ppm'), cv.COLOR_BGR2GRAY)
im3 = cv.cvtColor(cv.imread(r'./images/img3.ppm'), cv.COLOR_BGR2GRAY)
im4 = cv.cvtColor(cv.imread(r'./images/img4.ppm'), cv.COLOR_BGR2GRAY)
im5 = cv.cvtColor(im5Original, cv.COLOR_BGR2GRAY)

images = [im1, im2, im3, im4, im5]
```

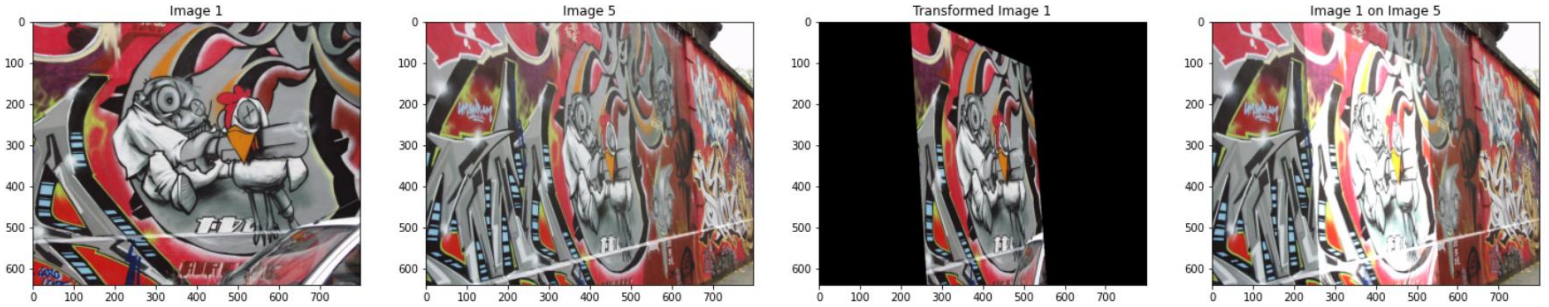
✓ 0.1s

```
sift = cv.SIFT_create()
kp1, dc1 = sift.detectAndCompute(im1,None)
kp5, dc5 = sift.detectAndCompute(im5,None)
bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
matches = bf.match(dc1,dc5)
matches = sorted(matches, key = lambda x:x.distance)
Matched = cv.drawMatches(im1, kp1, im5, kp5, matches[:50], im5, flags=2)
fig,ax = plt.subplots(figsize=(10,10))
ax.imshow(Matched)
```

✓ 2.6s

- Here we are computing and matching features using SIFT function and Flann-based matcher.
- Here in part b)we use RANSAC algorithm to calculate homography to outcome the perspective differences between the imagers and get a better SIFT feature match.
- In part c) we are using the cv.warpPerspective ti stitch the transformed images on to each other

b), c)




```python
def computeHomo(P1, P2):

    x1, y1, x2, y2, x3, y3, x4, y4 = P2[0], P2[1], P2[2], P2[3], P2[4], P2[5], P2[6], P2[7]
    x1T, x2T, x3T, x4T = P1[0], P1[1], P1[2], P1[3]
    zero_matrix = np.array([[0],[0],[0]])

    a = np.concatenate((zero_matrix.T,x1T, -y1*x1T), axis=1)
    b = np.concatenate((x1T,zero_matrix.T, -x1*x1T), axis=1)

    c = np.concatenate((zero_matrix.T,x2T, -y2*x2T), axis=1)
    d = np.concatenate((x2T,zero_matrix.T, -x2*x2T), axis=1)

    e = np.concatenate((zero_matrix.T,x3T, -y3*x3T), axis=1)
    f = np.concatenate((x3T,zero_matrix.T, -x3*x3T), axis=1)

    g = np.concatenate((zero_matrix.T,x4T, -y4*x4T), axis=1)
    h = np.concatenate((x4T,zero_matrix.T, -x4*x4T), axis=1)

    A = np.concatenate((a,b,c,d,e,f,g,h), axis=0, dtype = np.float64)


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

- general homography calculation

```python
list_kp1 = [kp1[mat.queryIdx].pt for mat in matches]
list_kp2 = [kp2[mat.trainIdx].pt for mat in matches]

threshold, best_inliers, best_H = 2, 0, 0

for k in range(N):
    four_random_points = randN(len(list_kp1)-1,4)

    fromPoints = []
    for i in range(4): fromPoints.append(np.array([[list_kp1[four_random_points[i]][0], list_kp1[four_random_points[i]][1], 1]]))

    toPoints = []
    for j in range(4):
        toPoints.append(list_kp2[four_random_points[j]][0])
        toPoints.append(list_kp2[four_random_points[j]][1])

    H = computeHomo(fromPoints, toPoints)

    inliers = 0
    for i in range(len(list_kp1)):
        X = [list_kp1[i][0], list_kp1[i][1], 1]
        hX = H@X
        hX /= hX[-1]
        error = np.sqrt(np.power(hX[0]-list_kp2[i][0],2) + np.power(hX[1]-list_kp2[i][1],2))
        if error < threshold: inliers+=1

    if inliers > best_inliers:
        best_inliers = inliers
        best_H = H

H_values.append(best_H)
H_1_to_5 = H_values[3] @ H_values
H_1_to_5 = H_values[3] @ H_values[2] @ H_values[1] @ H_values[0]
H_1_to_5 /= H_1_to_5[-1][-1]
```

- Calculating homography from images